The COCONUT API

Version 4.0.0005

Reference Manual

Hermann Schichl
University of Vienna
Faculty of Mathematics
Nordbergstr. 15
A-1090 Wien, Austria
email: Hermann.Schichl@univie.ac.at

Technical Report
July 2013

Contents

1 Introduction

The COCONUT environment is a modular solver environment for nonlinear continuous global optimization problems with an open-source kernel, which can be expanded by commercial and open-source solver components (inference modules).

The application programmer's interface (API) is designed to make the development of the various module types independent of each other and independent of the internal model representation. It is be a collection of open-source C++ classes protected by the LGPL and GPL license models, so that most of it could be used as part of commercial software (special license regulations are contained in the distribution). It uses the `FILIB++` library for interval computations and the Vienna matrix template library (`VMTL`) for the internal representation of various matrix classes. The graphs are implemented using the `VGTL` (Vienna Graph Template Library), and the search database is based on the `VDBL` (Vienna DataBase Library). - Support for dynamic linking relieves the user from recompilation when modules are added or removed. In addition, it is designed for distributed computing, and will probably be developed further (in the upcoming years) to support parallel computing as well.

1.1 Search Graph

The solution algorithm is an advanced branch-and-bound scheme which proceeds by working on the **search graph**, a **directed acyclic graph** (DAG) of search nodes, each representing an optimization problem, a **model**. The **search nodes** come in two flavors: **full nodes** which record the complete description of a model, and **delta nodes** which only contain the difference between the model represented by the node and its (then only) parent.

1.2 Models, Expressions

The optimization problems (models) stored in the **work nodes**, which are passed to the various inference engines, are kept as directed acyclic graphs (DAG), as well. This representation has big advantages. - Hereby, a complete optimization problem is always represented by a single DAG. The vertices of the graph represent operators similar to computational trees. Constants and variables are sources, objective and constraints are sinks of the DAG.

Every vertex represents a real valued function of n variables. Predefined functions include sum, product, max, min, elementary real functions (exp, log, pow, sqrt, ...), and also some discrete operators like `all_diff` and count.

1.3 Evaluators

For expression graphs (DAG or tree), special **forward** and **backward evaluators** are provided. Currently implemented are real function values, function ranges, gradients (real, interval), and slopes. In the near future evaluators for Hessians (real, interval) and second order slopes will be provided, as well.

1.4 Module Interfaces

The strategy calls the various model classes to perform clearly distinguished tasks. The **management modules** and the **initializers** change the internal structure, the **report modules** produce output, and the **inference modules** (**inference engines**, **graph analyzers**) calculate information about the models and the search graph.

2 Bug List

Member coco::coco::full_node::get_database () const

This method needs to be replaced, once the COCONUT environment really gets distributed.

Member coco::coco::full_node::get_database_ptr () const

This method needs to be replaced, once the COCONUT environment really gets distributed.

Member coco::coco::model::simplify_thin ()

The simplify_thin method does not yet work properly.

Member coco::coco::work_node::get (unsigned int __type)

The get function is not yet properly implemented.

Member coco::full_node::get_database () const

This method needs to be replaced, once the COCONUT environment really gets distributed.

Member coco::full_node::get_database_ptr () const

This method needs to be replaced, once the COCONUT environment really gets distributed.

Member coco::model::simplify_thin ()

The simplify_thin method does not yet work properly.

Member coco::work_node::get (unsigned int __type)

The get function is not yet properly implemented.

3 Module Index

3.1 Modules

Here is a list of all modules:

4 Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

5 Class Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

6 Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

7 File Index

7.1 File List

Here is a list of all files with brief descriptions:

8 Module Documentation

8.1 Classes and types for basic data management

Classes

- class coco::basic_alltype

  The basic alltype which can hold any of a number of basic types.
- class coco::control_data

  The class for communicating parameter information to COCONUT modules.
- class coco::datamap

  The base class for communicating with COCONUT modules.
- class coco::info_contents

  The class for returning additional information from inference modules.
- struct coco::interval_st

Constructor-free interval.

- class coco::interval

  Interval wrapper class.
- class coco::termination_reason

  This class holds the reason of termination of inference and management modules.
- class gptr

  Global pointer class.
- class ptr

  Local global pointer class.
- class counted_ptr

Defines

- #define BASIC_ALLTYPE_CHECK 1
- #define BASIC_ALLTYPE_ASSERT(A)
- #define BASIC_ALLTYPE_ASSERT2(A, B)
- #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)

Typedefs

- typedef basic_alltype coco::additional_info_u

  type definition for backwards compatibility with earlier API versions

Functions

- void coco::set_translation_map (const std::map< std::string, std::string > &tr_msg_map)

  A datamap function for setting the message translation map.
- const std::string & coco::get_translated_message (const std::string &msg)

  A datamap function for message translation.

8.1.1 Detailed Description

The classes and types in this section are the basics for the internal data management of the COCONUT environment.

8.1.2 Define Documentation

8.1.2.1 #define BASIC_ALLTYPE_ASSERT(*A*)

Value:

```
do { \
      if(__cont_type != (A)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/1: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

Compare the requested type with the stored type and throw an exception if they do not coincide.

Definition at line 124 of file search_graph.cc.

8.1.2.2 #define BASIC_ALLTYPE_ASSERT2(*A, B*)

Value:

```
do { \
      if(__cont_type != (A) && __cont_type != (B)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/2: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

Compare the requested type with the stored type and a fallback type and throw an exception if it does not coincide with either.

Definition at line 135 of file search_graph.cc.

8.1.2.3 #define BASIC_ALLTYPE_CHECK 1

If BASIC_ALLTYPE_CHECK is non-null, the exact contents of the basic_alltype are checked, and an exception is thrown if a different type is requested than is contained.

Definition at line 50 of file search_graph.cc.

8.1.2.4 #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(*CT, T*)

Value:

```
do { \
      if (__cont_type != (CT) && is_empty_vector()) \
        /* The const_cast is ugly, but there's no other option. */ \
        const_cast<basic_alltype *>(this)->operator=(std::vector<T>()); \
    } while(0)
```

Convert any size 0 vector to any other. This is needed because [] in Python is untyped.

Definition at line 154 of file search_graph.cc.

8.1.3 Typedef Documentation

8.1.3.1 typedef basic_alltype coco::additional_info_u

This is a definition to keep backwards compatibility with earlier versions of the API (before 2.35).

Definition at line 43 of file addinfo.h.

8.1.4 Function Documentation

8.1.4.1 const std::string & coco::get_translated_message (const std::string & *msg*)

This map can be used during the registration phase for producing highly descriptive termr messages or translations.

Definition at line 64 of file termreason.cc.

8.1.4.2 void coco::set_translation_map (const std::map< std::string, std::string > & *tr_msg_map*)

This function is used to set a new message translation map.

Definition at line 53 of file termreason.cc.

8.2 Basic API utilities

Classes

- class coco::api_exception

  API exception class.
- class coco::nyi_exception

  Not Yet Implemented exception class.
- class coco::graph_analyzer_exception

  Graph analyzer exception class.
- class coco::inference_engine_exception

  Inference engine exception class.
- class coco::initializer_exception

  Initializer exception class.
- class coco::management_module_exception

  Management module exception class.
- class coco::report_module_exception

  Report module exception class.

Defines

- #define PURE_VIRTUAL { throw "Pure virtual function called!"; }

Enumerations

- enum coco::api_exception_type { coco::apiee_internal = 1, coco::apiee_evaluator = 2, coco::apiee_io = 3, coco::apiee_delta = 4, coco::apiee_search_graph = 5, coco::apiee_communication_data = 6, coco::apiee_inference_engine = 7, coco::apiee_graph_analyzer = 8, coco::apiee_management_module = 9, coco::apiee_initializer = 10, coco::apiee_report_module = 11, coco::apiee_oom = 12, coco::apiee_nyi = 13, coco::apiee_other = 14 }

  Enum for classifying api_exceptions.
- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer =

10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

Enum for classifying api_exceptions.

- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

  Enum for classifying api_exceptions.

- enum coco::coco::coco::api_exception_type { coco::coco::coco::apiee_internal = 1, coco::coco::coco::apiee_evaluator = 2, coco::coco::coco::apiee_io = 3, coco::coco::coco::apiee_delta = 4, coco::coco::coco::apiee_search_graph = 5, coco::coco::coco::apiee_communication_data = 6, coco::coco::coco::apiee_inference_engine = 7, coco::coco::coco::apiee_graph_analyzer = 8, coco::coco::coco::apiee_management_module = 9, coco::coco::coco::apiee_initializer = 10, coco::coco::coco::apiee_report_module = 11, coco::coco::coco::apiee_oom = 12, coco::coco::coco::apiee_nyi = 13, coco::coco::coco::apiee_other = 14 }

  Enum for classifying api_exceptions.

- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi =

13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

Enum for classifying api_exceptions.

- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

  Enum for classifying api_exceptions.

- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer =

10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

Enum for classifying api_exceptions.

Functions

- template<class _TH >
  std::string coco::coco::convert_to_str (const _TH &_h)

  Convert an object of any printable class to a string.
- template<typename _TK , typename _Ve , typename _TC , typename _TA >
  ostream & std::operator<< (ostream &o, const map< _TK, _Ve, _TC, _TA > &a)
- template<typename _TK , typename _Ve , typename _TC , typename _TA >
  ostream & std::operator<< (ostream &o, const multimap< _TK, _Ve, _TC, _TA > &a)
- template<typename _Ve , typename _TA >
  ostream & std::operator<< (ostream &o, const vector< _Ve, _TA > &a)
- template<typename _Ve , typename _TA >
  ostream & std::operator<< (ostream &o, const list< _Ve, _TA > &a)
- template<typename _Ve , typename _TA >
  ostream & std::operator<< (ostream &o, const deque< _Ve, _TA > &a)
- template<typename _Ve , typename _TC , typename _TA >
  ostream & std::operator<< (ostream &o, const set< _Ve, _TC, _TA > &a)
- template<typename _Ve , typename _TC , typename _TA >
  ostream & std::operator<< (ostream &o, const multiset< _Ve, _TC, _TA > &a)
- template<typename _TA , typename _TB >
  ostream & std::operator<< (ostream &o, const pair< _TA, _TB > &a)
- template<typename _TA , typename _TB , typename _TC >
  ostream & std::operator<< (ostream &o, const triple< _TA, _TB, _TC > &a)

8.2.1 Detailed Description

The classes and functions in this section are basic utilities used throughout the COCONUT environment, like conversion routines and exception classes.

8.2.2 Define Documentation

8.2.2.1 #define PURE_VIRTUAL { throw "Pure virtual function called!"; }

This macro is used to circumvent certain bugs connected with pure virtual methods in GNU C++ prior version 3.

Definition at line 37 of file coconut_config.h.

8.2.3 Enumeration Type Documentation

8.2.3.1 enum coco::api_exception_type

This enum contains type values for the different api_exceptions.

Enumerator:

apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)

Definition at line 44 of file api_exception.h.

8.2.3.2 enum coco::coco::api_exception_type

This enum contains type values for the different api_exceptions.

Enumerator:

apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines

apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related

apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)

Definition at line 44 of file search_graph.cc.

8.2.3.3 enum coco::coco::api_exception_type

This enum contains type values for the different api_exceptions.

Enumerator:

apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory

apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules

apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)

Definition at line 44 of file search_graph.cc.

8.2.3.4 enum coco::coco::api_exception_type

This enum contains type values for the different api_exceptions.

Enumerator:

apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error

apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related

apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)

Definition at line 44 of file search_graph.cc.

8.2.3.5 enum coco::coco::coco::api_exception_type

This enum contains type values for the different api_exceptions.

Enumerator:

apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)

Definition at line 44 of file search_graph.cc.

8.2.3.6 enum coco::coco::api_exception_type

This enum contains type values for the different api_exceptions.

Enumerator:

apiee_internal internal stuff
apiee_evaluator evaluator related

apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff

apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)

Definition at line 44 of file search_graph.cc.

8.2.3.7 enum coco::coco::api_exception_type

This enum contains type values for the different api_exceptions.

Enumerator:

apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines

apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related

apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)
apiee_internal internal stuff
apiee_evaluator evaluator related
apiee_io I/O error
apiee_delta delta related
apiee_search_graph search graph related
apiee_communication_data communication data related
apiee_inference_engine for inference engines
apiee_graph_analyzer for graph analyzers
apiee_management_module for management modules
apiee_initializer for initializers
apiee_report_module for report modules
apiee_oom out of memory
apiee_nyi not yet implemented
apiee_other other exception (info in msg)

Definition at line 44 of file expression.h.

8.2.4 Function Documentation

8.2.4.1 template<class _TH > std::string coco::coco::convert_to_str (const _TH & *_h*) [inline]

This function converts an object _h of class _TH into a std::string, if it is printable.

Definition at line 46 of file expression.h.

8.2.4.2 template<typename _TA , typename _TB > ostream& std::operator<< (ostream & *o,* const pair< _TA, _TB > & *a*)

This function defines the stream output operator for pairs.

Definition at line 40 of file print_tuple.h.

8.2.4.3 template<typename _Ve , typename _TC , typename _TA > ostream& std::operator<< (ostream & *o,* const set< _Ve, _TC, _TA > & *a*)

This function defines the stream output operator for standard sets.

Definition at line 40 of file print_set.h.

8.2.4.4 template<typename _TK , typename _Ve , typename _TC , typename _TA > ostream& std::operator<< (ostream & *o,* const map< _TK, _Ve, _TC, _TA > & *a*)

This function defines the stream output operator for standard maps.

Definition at line 42 of file print_map.h.

8.2.4.5 template<typename _Ve , typename _TA > ostream& std::operator<< (ostream & *o,* const vector< _Ve, _TA > & *a*)

This function defines the stream output operator for standard vectors.

Definition at line 42 of file print_seq.h.

8.2.4.6 template<typename _TA , typename _TB , typename _TC > ostream& std::operator<< (ostream & *o,* const triple< _TA, _TB, _TC > & *a*)

This function defines the stream output operator for triples.

Definition at line 48 of file print_tuple.h.

8.2.4.7 template<typename _Ve , typename _TC , typename _TA > ostream& std::operator<< (ostream & *o,* const multiset< _Ve, _TC, _TA > & *a*)

This function defines the stream output operator for standard multisets.

Definition at line 56 of file print_set.h.

8.2.4.8 template<typename _TK , typename _Ve , typename _TC , typename _TA > ostream& std::operator<< (ostream & *o,* const multimap< _TK, _Ve, _TC, _TA > & *a*)

This function defines the stream output operator for standard multimaps.

Definition at line 58 of file print_map.h.

8.2.4.9 template<typename _Ve , typename _TA > ostream& std::operator<< (ostream & *o,* const list< _Ve, _TA > & *a*)

This function defines the stream output operator for standard lists.

Definition at line 58 of file print_seq.h.

8.2.4.10 template<typename _Ve , typename _TA > ostream& std::operator<< (ostream & *o,* const deque< _Ve, _TA > & *a*)

This function defines the stream output operator for standard double ended queues.

Definition at line 75 of file print_seq.h.

8.3 Models

Classes

- class coco::annotation

  Annotations for Models.
- class coco::expression_node

  The base class for a node in the expression DAGs.
- class coco::expression_print_visitor
- class coco::graphorder_visitor

  This visitor class is used for computing a graph order.
- class coco::model

  The model class (an attributed DAG of expression nodes, lowest class in the model hierarchy)
- class coco::model_iddata

  The model id-data class (the topmost in the model class hierarchy)
- class coco::model_gid

  Model Group Data Class (middle class in the model hierarchy)

Typedefs

- typedef model::walker coco::expression_walker

  Walker to the expression DAG.
- typedef model::const_walker coco::expression_const_walker

  Const walker to the expression DAG.
- typedef model::walker coco::coco::expression_walker

  Walker to the expression DAG.
- typedef model::const_walker coco::coco::expression_const_walker

  Const walker to the expression DAG.

Functions

- void coco::graphorder (const model &DAG, std::vector< unsigned int > &order)

  Compute a graph order.
- void coco::graphorder (const model &DAG, std::vector< unsigned int > &order, std::vector< unsigned int > &inv_order)

  Compute a graph order.

Variables

- const char * coco::expr_names [ ]

Operator types of standard expressions

These definitions define the standard operator types for the expression_node class. In the operator descriptions y_i is used for the value of the ith child of the node, while $x_i = c_i y_i$ stands for the value of the ith child of the node multiplied by the ith entry of the `coeffs` array.

- #define EXPRINFO_GHOST 0
- #define EXPRINFO_CONSTANT -1
- #define EXPRINFO_VARIABLE -2
- #define EXPRINFO_SUM -3
- #define EXPRINFO_MEAN -4
- #define EXPRINFO_PROD -5
- #define EXPRINFO_MAX -6
- #define EXPRINFO_MIN -7
- #define EXPRINFO_MONOME -8
- #define EXPRINFO_SCPROD -9
- #define EXPRINFO_NORM -10
- #define EXPRINFO_INVERT -11
- #define EXPRINFO_SQUARE -12
- #define EXPRINFO_SQROOT -13
- #define EXPRINFO_ABS -14
- #define EXPRINFO_INTPOWER -15
- #define EXPRINFO_EXP -16
- #define EXPRINFO_LOG -17
- #define EXPRINFO_SIN -18
- #define EXPRINFO_COS -19
- #define EXPRINFO_SINH -20
- #define EXPRINFO_COSH -21
- #define EXPRINFO_GAUSS -22
- #define EXPRINFO_POLY -23
- #define EXPRINFO_POW -24
- #define EXPRINFO_DIV -25
- #define EXPRINFO_ATAN2 -26
- #define EXPRINFO_LIN -27
- #define EXPRINFO_QUAD -28
- #define EXPRINFO_IQUAD -29
- #define EXPRINFO_WSUMSQR -30
- #define EXPRINFO_LINSQR -31
- #define EXPRINFO_RE -32
- #define EXPRINFO_IM -33
- #define EXPRINFO_ARG -34
- #define EXPRINFO_CPLXCONJ -35
- #define EXPRINFO_LOOKUP -36
- #define EXPRINFO_PWLIN -37
- #define EXPRINFO_SPLINE -38
- #define EXPRINFO_PWCONSTLC -39
- #define EXPRINFO_PWCONSTRC -40
- #define EXPRINFO_IN -41
- #define EXPRINFO_IF -42
- #define EXPRINFO_AND -43

- #define EXPRINFO_OR -44
- #define EXPRINFO_NOT -45
- #define EXPRINFO_IMPLIES -46
- #define EXPRINFO_COUNT -47
- #define EXPRINFO_ALLDIFF -48
- #define EXPRINFO_HISTOGRAM -49
- #define EXPRINFO_LEVEL -50
- #define EXPRINFO_NEIGHBOR -51
- #define EXPRINFO_NOGOOD -52
- #define EXPRINFO_EXPECTATION -53
- #define EXPRINFO_INTEGRAL -54
- #define EXPRINFO_DET -55
- #define EXPRINFO_COND -56
- #define EXPRINFO_PSD -57
- #define EXPRINFO_MPROD -58
- #define EXPRINFO_FEM -59
- #define EXPRINFO_CMPROD -60
- #define EXPRINFO_CGFEM -61
- #define EXPRINFO_UNDEFINED -65
- #define EXPRINFO_NUMOFPREDEF -(EXPRINFO_UNDEFINED)
- #define EXPRINFO_ASQR -62
- #define EXPRINFO_D1FUNC -63
- #define EXPRINFO_BLACKBOX -64

8.3.1 Detailed Description

The classes in this section define the internal representation of models (optimization problems) in the COCONUT environment.

8.3.2 Define Documentation

8.3.2.1 #define EXPRINFO_ABS -14

The absolute value node is $|x_0 + q|$, where q is stored in `params.nd()`.

Definition at line 144 of file search_graph.cc.

8.3.2.2 #define EXPRINFO_ALLDIFF -48

This node returns 1 if all children are pairwise different. Here different means $|x_i - x_j| > \delta$ with δ in `params.nd()`.

Definition at line 314 of file search_graph.cc.

8.3.2.3 #define EXPRINFO_AND -43

The value is 1 if all $x_i \in \boldsymbol{x}_i$ where the $\boldsymbol{x}_i$ are stored in `params.i()`, and 0 otherwise.

Definition at line 291 of file search_graph.cc.

8.3.2.4 #define EXPRINFO_ARG -34

This node returns the argument of a complex number.

Definition at line 241 of file search_graph.cc.

8.3.2.5 #define EXPRINFO_ASQR -62

This is a virtual expression needed for the `dag_delta` class. Ghost nodes are exceptions to the rule that node numbers have to be unique throughout all models in a model group. The node number of a ghost node is the same as the node number of the node, of which it is a ghost.

Definition at line 379 of file search_graph.cc.

8.3.2.6 #define EXPRINFO_ATAN2 -26

The atan2 node has no parameters, and its definition is $\mathrm{atan2}(x_0, x_1) = \mathrm{atan}(x_0/x_1)$.

Definition at line 197 of file search_graph.cc.

8.3.2.7 #define EXPRINFO_BLACKBOX -64

This is a virtual expression needed for the `dag_delta` class. Ghost nodes are exceptions to the rule that node numbers have to be unique throughout all models in a model group. The node number of a ghost node is the same as the node number of the node, of which it is a ghost.

Definition at line 383 of file search_graph.cc.

8.3.2.8 #define EXPRINFO_CGFEM -61

This node computes information relevant for FEM computations with constant A.

Definition at line 377 of file search_graph.cc.

8.3.2.9 #define EXPRINFO_CMPROD -60

This node computes a matrix vector product with constant A.

Definition at line 373 of file search_graph.cc.

8.3.2.10 #define EXPRINFO_COND -56

This node computes the condition of a matrix.

Definition at line 359 of file search_graph.cc.

8.3.2.11 #define EXPRINFO_CONSTANT -1

This node represents a constant, whose value is contained in `params.nd()`.

Definition at line 80 of file search_graph.cc.

8.3.2.12 #define EXPRINFO_COS -19

The cosine node is $\cos(x_0 + q)$, where q is stored in `params.nd()`.

Definition at line 164 of file search_graph.cc.

8.3.2.13 #define EXPRINFO_COSH -21

The cosine node is $\cosh(x_0 + q)$, where q is stored in `params.nd()`.

Definition at line 172 of file search_graph.cc.

8.3.2.14 #define EXPRINFO_COUNT -47

The count node returns for how many i we have $x_i \in \boldsymbol{x}_i$, where $*$ the intervals $\boldsymbol{x}_i$ are stored in `params.i()`.

Definition at line 309 of file search_graph.cc.

8.3.2.15 #define EXPRINFO_CPLXCONJ -35

This node returns the complex conjugate of a complex number.

Definition at line 246 of file search_graph.cc.

8.3.2.16 #define EXPRINFO_D1FUNC -63

This is a virtual expression needed for the `dag_delta` class. Ghost nodes are exceptions to the rule that node numbers have to be unique throughout all models in a model group. The node number of a ghost node is the same as the node number of the node, of which it is a ghost.

Definition at line 381 of file search_graph.cc.

8.3.2.17 #define EXPRINFO_DET -55

This node computes the determinant of a matrix.

Definition at line 356 of file search_graph.cc.

8.3.2.18 #define EXPRINFO_DIV -25

The general division node is $a * y_0/y_1$ and a is stored in `params.nd()`.

Definition at line 192 of file search_graph.cc.

8.3.2.19 #define EXPRINFO_EXP -16

The exponential node is e^{x_0+q}, where q is stored in `params.nd()`.

Definition at line 152 of file search_graph.cc.

8.3.2.20 #define EXPRINFO_EXPECTATION -53

This node computes the expectation value of a function depending on stochastic nodes.

Definition at line 344 of file search_graph.cc.

8.3.2.21 #define EXPRINFO_FEM -59

This node computes information relevant for FEM computations.

Definition at line 368 of file search_graph.cc.

8.3.2.22 #define EXPRINFO_GAUSS -22

The gaussian function node is $e^{-(x_0-q_0)^2/q_1^2}$, where the two dimensional vector q is stored in `params.d()`.

Definition at line 177 of file search_graph.cc.

8.3.2.23 #define EXPRINFO_GHOST 0

This is a virtual expression needed for the `dag_delta` class. Ghost nodes are exceptions to the rule that node numbers have to be unique throughout all models in a model group. The node number of a ghost node is the same as the node number of the node, of which it is a ghost.

Definition at line 75 of file search_graph.cc.

8.3.2.24 #define EXPRINFO_HISTOGRAM -49

The histogram node counts how many children x_i are in which interval $\boldsymbol{x}_{2k}$. The numbers n_k are collected, and if for all k the value $n_k \in \boldsymbol{x}_{2k+1}$, the result is 1. Otherwise, it is 0.

Definition at line 320 of file search_graph.cc.

8.3.2.25 #define EXPRINFO_IF -42

We get the interval $\boldsymbol{x}$ from `params.ni()` and define the result n of the node as $n = \begin{cases} x_1 & \text{if } x_0 \in \boldsymbol{x} \\ x_2 & \text{otherwise.} \end{cases}$

Definition at line 287 of file search_graph.cc.

8.3.2.26 #define EXPRINFO_IM -33

This node returns the imaginary part of a complex number.

Definition at line 238 of file search_graph.cc.

8.3.2.27 #define EXPRINFO_IMPLIES -46

The value is 1 if the following is true: $x_0 \in \boldsymbol{x}_0 \implies x_1 \in \boldsymbol{x}_1$, where the $\boldsymbol{x}_i$ are stored in `params.i()`. Otherwise, the value is 0.

Definition at line 305 of file search_graph.cc.

8.3.2.28 #define EXPRINFO_IN -41

The point defined by $x = (x_i)$ is tested against the box $\boldsymbol{x}$ stored in `params.i()`. If $x \in \text{int}(\boldsymbol{x})$ the value is 1, if $x \in \partial\boldsymbol{x}$, the return value is 0, and if $x \notin \boldsymbol{x}$ the node's value is -1.

Definition at line 276 of file search_graph.cc.

8.3.2.29 #define EXPRINFO_INTEGRAL -54

This node computes the definite integral of a function depending on free nodes over the box stored in `params.i()`.

Definition at line 348 of file search_graph.cc.

8.3.2.30 #define EXPRINFO_INTPOWER -15

The integer power node is x_0^n, where the integer n is in `params.nn()`.

Definition at line 148 of file search_graph.cc.

8.3.2.31 #define EXPRINFO_INVERT -11

Inversion is defined as a/y_0 with a in `params.nd()`.

Definition at line 132 of file search_graph.cc.

8.3.2.32 #define EXPRINFO_IQUAD -29

This node is a quadratic form in **all variables**. The `params.m()` stores the enhanced matrix $(A|b)$ for the function $x^T Ax + b^T x$. A is assumed symmetric and sparse.

Definition at line 217 of file search_graph.cc.

8.3.2.33 #define EXPRINFO_LEVEL -50

The matrix A in `params.mi()` contains a set of boxes $A_{i:}$ contained within another. The result is the smallest row index j such that the vector $x = (x_i)$ is contained in $A_{j:}$.

Definition at line 326 of file search_graph.cc.

8.3.2.34 #define EXPRINFO_LIN -27

This node is a linear combination of **all variables**. In `params.nn()` the row number in the matrix of linear constraints is stored.

Definition at line 204 of file search_graph.cc.

8.3.2.35 #define EXPRINFO_LINSQR -31

This node is a quadratic form in **all variables**. The `params.m()` stores the enhanced matrix $(A|b)$ for the function $x^T Ax + b^T x$. A is assumed symmetric and sparse.

Definition at line 230 of file search_graph.cc.

8.3.2.36 #define EXPRINFO_LOG -17

The logarithm node is $\log(x_0 + q)$, where q is stored in `params.nd()`.

Definition at line 156 of file search_graph.cc.

8.3.2.37 #define EXPRINFO_LOOKUP -36

This node computes a function with a table lookup procedure. The table is stored in `params.m()`.

Definition at line 252 of file search_graph.cc.

8.3.2.38 #define EXPRINFO_MAX -6

The maximum node is defined as $\max(c, x_i \quad i = 0, \ldots, n)$ with c in `params.nd()`.

Definition at line 108 of file search_graph.cc.

8.3.2.39 #define EXPRINFO_MEAN -4

This is a sum node where it is guaranteed that all coefficients are positive, and their sum is \$1\$. So this is a convex linear combination without constant term: $\sum_{i=0}^{n} x_i$.

Definition at line 100 of file search_graph.cc.

8.3.2.40 #define EXPRINFO_MIN -7

The minimum node is defined as $\min(c, x_i \quad i = 0, \ldots, n)$ with c in `params.nd()`.

Definition at line 112 of file search_graph.cc.

8.3.2.41 #define EXPRINFO_MONOME -8

The general monomial node is defined as $\prod_{i=0}^{n-1} x_i^{n_i}$, where the n_i are contained in the vector `params.n()`.

Definition at line 117 of file search_graph.cc.

8.3.2.42 #define EXPRINFO_MPROD -58

This node computes a matrix-vector product.

Definition at line 365 of file search_graph.cc.

8.3.2.43 #define EXPRINFO_NEIGHBOR -51

This returns 1 if x_0 and x_1 are neighbors, i.e., if (x_0, x_1) is in the list of allowed value pairs (v_{2k}, v_{2k+1}) where the vector v is stored in `params.n()`. The x_i and v_i are integers.

Definition at line 334 of file search_graph.cc.

8.3.2.44 #define EXPRINFO_NOGOOD -52

This returns 1 if $x = v$ with v in `params.n()`. Both $x = (x_i)$ and v are integer vectors.

Definition at line 338 of file search_graph.cc.

8.3.2.45 #define EXPRINFO_NORM -10

The norm node has the form $\|x\|_k$, where k is stored in `params.nd()`, and x is the vector built from the values of all children.

Definition at line 126 of file search_graph.cc.

8.3.2.46 #define EXPRINFO_NOT -45

The value is 0 if $x_0 \in \boldsymbol{x}$ and 1 otherwise. Here $\boldsymbol{x}$ is in `params.ni()`.

Definition at line 299 of file search_graph.cc.

8.3.2.47 #define EXPRINFO_NUMOFPREDEF -(EXPRINFO_UNDEFINED)

This is the number of standard operator types.

Definition at line 392 of file search_graph.cc.

8.3.2.48 #define EXPRINFO_OR -44

The value is 0 if all $x_i \notin \boldsymbol{x}_i$ where the $\boldsymbol{x}_i$ are stored in `params.i()`, and 1 otherwise.

Definition at line 295 of file search_graph.cc.

8.3.2.49 #define EXPRINFO_POLY -23

The polynomial node is $\sum_{i=0}^{n} \alpha_i x_0^i$, where the coefficients α_i are stored in `params.d()`.

Definition at line 182 of file search_graph.cc.

8.3.2.50 #define EXPRINFO_POW -24

The general power node is $(x_0 + p)^{x_1}$ with p in `params.nd()`.

Definition at line 188 of file search_graph.cc.

8.3.2.51 #define EXPRINFO_PROD -5

The product node has the form $c \prod_{i=0}^{n} y_i$, where c is in `params.nd()`.

Definition at line 104 of file search_graph.cc.

8.3.2.52 #define EXPRINFO_PSD -57

This node checks whether a matrix is positive semidefinite.

Definition at line 362 of file search_graph.cc.

8.3.2.53 #define EXPRINFO_PWCONSTLC -39

This node computes a piecewise constant left-continuous function from a table. The table is stored in `params.m()`.

Definition at line 264 of file search_graph.cc.

8.3.2.54 #define EXPRINFO_PWCONSTRC -40

This node computes a piecewise constant right-continuous function from a table. The table is stored in `params.m()`.

Definition at line 268 of file search_graph.cc.

8.3.2.55 #define EXPRINFO_PWLIN -37

This node computes a piecewise linear function from a table. The table is stored in `params.m()`.

Definition at line 256 of file search_graph.cc.

8.3.2.56 #define EXPRINFO_QUAD -28

This node is a quadratic form in **all variables**. The `params.m()` stores the enhanced matrix $(A|b)$ for the function $x^T Ax + b^T x$. A is assumed symmetric and sparse.

Definition at line 210 of file search_graph.cc.

8.3.2.57 #define EXPRINFO_RE -32

This node returns the real part of a complex number.

Definition at line 235 of file search_graph.cc.

8.3.2.58 #define EXPRINFO_SCPROD -9

The scalar product is defined as $\sum_{i=0}^{k-1} x_{2i}x_{2i+1}$ without parameters.

Definition at line 121 of file search_graph.cc.

8.3.2.59 #define EXPRINFO_SIN -18

The sine node is $\sin(x_0 + q)$, where q is stored in `params.nd()`.

Definition at line 160 of file search_graph.cc.

8.3.2.60 #define EXPRINFO_SINH -20

The sine node is $\sinh(x_0 + q)$, where q is stored in `params.nd()`.

Definition at line 168 of file search_graph.cc.

8.3.2.61 #define EXPRINFO_SPLINE -38

This node computes a cubic spline from a table. The table is stored in `params.m()`.

Definition at line 260 of file search_graph.cc.

8.3.2.62 #define EXPRINFO_SQROOT -13

The square root node is $\sqrt{x_0 + q}$, where q is stored in `params.nd()`.

Definition at line 140 of file search_graph.cc.

8.3.2.63 #define EXPRINFO_SQUARE -12

The square node is $(x_0 + q)^2$ where q is stored in `params.nd()`.

Definition at line 136 of file search_graph.cc.

8.3.2.64 #define EXPRINFO_SUM -3

This is a general linear combination of the form $\sum_{i=1}^{n} x_i + c$ where c is stored in `params.nd()`. Here x_i as in all operator types stands for $k_i y_i$, where y_i is the result of the ith child and k_i stands for `coeff[i]`.

Definition at line 95 of file search_graph.cc.

8.3.2.65 #define EXPRINFO_UNDEFINED -65

This is the artifical number for undefined nodes.

Definition at line 389 of file search_graph.cc.

8.3.2.66 #define EXPRINFO_VARIABLE -2

The variable number of this node is contained in `params.nn()`.

Definition at line 83 of file search_graph.cc.

8.3.2.67 #define EXPRINFO_WSUMSQR -30

This node is a quadratic form in **all variables**. The `params.m()` stores the enhanced matrix $(A|b)$ for the function $x^T Ax + b^T x$. A is assumed symmetric and sparse.

Definition at line 223 of file search_graph.cc.

8.3.3 Typedef Documentation

8.3.3.1 typedef model::const_walker coco::expression_const_walker

This type defines a const walker (generalized const pointer) to the expression DAG.

Definition at line 1235 of file model.h.

8.3.3.2 typedef model::const_walker coco::coco::expression_const_walker

This type defines a const walker (generalized const pointer) to the expression DAG.

Definition at line 1235 of file search_graph.cc.

8.3.3.3 typedef model::walker coco::coco::expression_walker

This type defines a walker (generalized pointer) to the expression DAG.

Definition at line 1230 of file search_graph.cc.

8.3.3.4 typedef model::walker coco::expression_walker

This type defines a walker (generalized pointer) to the expression DAG.

Definition at line 1230 of file model.h.

8.3.4 Function Documentation

8.3.4.1 void coco::graphorder (const model & *DAG,* std::vector< unsigned int > & *order*)

This function computes a node order in `order`, which is compatible with the DAG structure defined by `DAG`.

Definition at line 132 of file graphorder.cc.

8.3.4.2 void coco::graphorder (const model & *DAG,* std::vector< unsigned int > & *order,* std::vector< unsigned int > & *inv_order*)

This function computes a node order in `order`, which is compatible with the DAG structure defined by `DAG`, and an inverse order map.

Definition at line 137 of file graphorder.cc.

8.3.5 Variable Documentation

8.3.5.1 const char * coco::expr_names

This array of strings keeps the names of the various operator types in the .dag format.

Definition at line 37 of file expression.cc.

8.4 Search Graph

Classes

- class coco::work_node_comp_hook

  The work_node_comp_hook class (work node computation hook)
- class coco::diameter_comp_hook

  The log-volume computation hook (work node computation hook)
- class coco::inference_engine_comp_hook

  The inference engine meta computation hook (work node computation hook)
- class coco::logvol_comp_hook

  The log-volume computation hook (work node computation hook)
- class coco::memory_comp_hook

  The memory computation hook (work node computation hook)
- class coco::objbounds_comp_hook

  The objective-bounds computation hook (work node computation hook)
- class coco::pending_status_comp_hook

  The pending status computation hook (work node computation hook)
- class coco::pfstar_hook

  The pfstar computation hook (work node computation hook)
- class coco::coco::search_node

  Base type of the nodes in the search graph.
- class coco::coco::delta_node

  Class holding the delta nodes in the search graph.
- class coco::coco::full_node

  Class holding the full nodes in the search graph.
- class coco::coco::work_node

  Work node, which is passed to the inference engines.
- class coco::coco::work_node::constraint_iterator_base

  The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.
- class coco::sum_deltas

  Pre-post visitor for summing up all the deltas during work node extraction.
- class coco::search_graph

  The search graph.
- class coco::vol_comp_hook

  The volume computation hook (work node computation hook)

Typedefs

- typedef uint32_t coco::search_node_id

  Type of the unique search node identifier.
- typedef uint32_t coco::coco::search_node_id

  Type of the unique search node identifier.
- typedef search_graph::const_walker coco::search_inspector

  The search inspector for graph analysis.
- typedef search_graph::walker coco::search_focus

  The search focus for work node selection.

Enumerations

- enum coco::coco::search_node_relation { coco::coco::snr_root, coco::coco::snr_reduction, coco::coco::snr_relaxation, coco::coco::snr_split, coco::coco::snr_glue, coco::coco::snr_worknode, coco::coco::snr_virtual }

  Enum specifying node relations in the search graph.
- enum coco::search_node_relation { coco::snr_root, coco::snr_reduction, coco::snr_relaxation, coco::snr_split, coco::snr_glue, coco::snr_worknode, coco::snr_virtual }

  Enum specifying node relations in the search graph.

Functions

- work_node coco::full_node_to_work_node (full_node &n_full, gptr< search_node > &ground)

  This function converts a full_node to a work_node.
- work_node full_node_to_work_node (full_node &n_full, gptr< search_node > &ground)

  This function converts a full_node to a work_node.

8.4.1 Detailed Description

The classes and types in this section are used to build the search graph structure used during the solution process.

8.4.2 Typedef Documentation

8.4.2.1 typedef search_graph::walker coco::search_focus

There are only few variables of the search_focus type which are used to select nodes in the search graph for analysis.

Definition at line 343 of file search_graph.h.

8.4.2.2 typedef search_graph::const_walker coco::search_inspector

Variables of the search_inspector type are used to analyze the search graph.

Definition at line 336 of file search_graph.h.

8.4.2.3 typedef uint32_t coco::coco::search_node_id

This is the type of the unique identifier generated for a work node inside a search graph.

Definition at line 44 of file coconut_types.h.

8.4.2.4 typedef uint32_t coco::coco::search_node_id

This is the type of the unique identifier generated for a work node inside a search graph.

Definition at line 45 of file search_graph.cc.

8.4.3 Enumeration Type Documentation

8.4.3.1 enum coco::coco::search_node_relation

This enum is used to specify the relation of a node to its parent(s) in the search graph.

Enumerator:

snr_root this is the root node of a search graph
snr_reduction this is a reduction of the parent node
snr_relaxation this is a relaxation of the parent node
snr_split this is one of a number of splits of the parent
snr_glue this is a glueing of a number of parent nodes
snr_worknode this is a **standalone** work-node
snr_virtual this is a virtual node of the search graph

Definition at line 62 of file search_graph.cc.

8.4.3.2 enum coco::search_node_relation

This enum is used to specify the relation of a node to its parent(s) in the search graph.

Enumerator:

snr_root this is the root node of a search graph
snr_reduction this is a reduction of the parent node
snr_relaxation this is a relaxation of the parent node
snr_split this is one of a number of splits of the parent
snr_glue this is a glueing of a number of parent nodes
snr_worknode this is a **standalone** work-node
snr_virtual this is a virtual node of the search graph

Definition at line 62 of file search_node.h.

8.4.4 Function Documentation

8.4.4.1 work_node coco::full_node_to_work_node (full_node & *n_full*, gptr< search_node > & *ground*) [inline]

This function converts the full_node `n_full` to a work_node. The parameter `ground` provides a reference to the ground of the search graph.

Definition at line 40 of file search_graph.cc.

8.4.4.2 work_node full_node_to_work_node (full_node & *n_full*, gptr< search_node > & *ground*) [inline]

This function converts the full_node `n_full` to a work_node. The parameter `ground` provides a reference to the ground of the search graph.

Definition at line 40 of file sum_deltas.h.

8.5 Deltas

Classes

- class coco::annotation_delta

  the delta class for annotation changes
- class coco::no_certificate

  The not-certified certificate.
- class coco::split_certificate

  The certificate for deltas formed by splits.
- class coco::compound_certificate

  The certificate for deltas formed by compressing bound_delta entries.
- class coco::rigorous_module_certificate

  The certificate for deltas computed by rigorous inference engines.
- class coco::certificate

  The certificate class (certifies deltas for rigorous mode operation)
- class coco::certificate_base

  Base class for the certificates.
- class coco::delta_get_action

  Stored procedure class for computing the delta action specifier.
- class coco::delta

  The delta class (updates to work nodes)
- class coco::delta_base

  Base class for the deltas.
- class coco::undelta

  The undelta class (undo of updates to work nodes)
- class coco::undelta_base

  Base class for the undeltas.
- class coco::bound_undelta

  The bound undelta class for undoing changes to the node bounds in a model.
- class coco::bound_delta

  The bound delta class for changing the node bounds within a model.
- class coco::boxes_delta

  A delta class which adds new boxes to the search database.
- class coco::dag_undelta

  The DAG undelta class for undoing changes to the DAG of a model.
- class coco::dag_delta

  The DAG delta class for performing changes to the DAG of a model.
- class coco::infeasible_undelta

  The infeasible undelta class for undoing changes to the feasibility of a model.
- class coco::infeasible_delta

  The infeasible delta class for marking a model as infeasible.
- class coco::point_delta

  A delta class which adds new points to the search database.
- class coco::semantics_undelta

  The semantics undelta class for undoing changes to the node semantics in a model.
- class coco::semantics_delta

The semantics delta class for changing the node semantics within a model.

- class coco::split_undelta

  The split undelta class for removing proposed splits from a work_node.
- class coco::split_delta

  The split delta class for proposing useful splits.
- class coco::table_delta

  The base class for all deltas adding information to the search database.

Typedefs

- typedef vdbl::rowid coco::delta_id

  The class for delta ids.
- typedef vdbl::rowid coco::coco::delta_id

  The class for delta ids.

Functions

- std::ostream & coco::operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.
- std::ostream & coco::operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.
- std::ostream & coco::coco::operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.
- std::ostream & coco::coco::operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.

8.5.1 Detailed Description

The classes and types in this section define the deltas used for updating the internal representation of the models.

8.5.2 Typedef Documentation

8.5.2.1 typedef vdbl::rowid coco::coco::delta_id

The delta_id type is essentially a vdbl::rowid pointing to a row in the `deltas` table of the search database.

Definition at line 46 of file api_deltabase.h.

8.5.2.2 typedef vdbl::rowid coco::coco::delta_id

The delta_id type is essentially a vdbl::rowid pointing to a row in the `deltas` table of the search database.

Definition at line 46 of file search_graph.cc.

8.5.3 Function Documentation

8.5.3.1 std::ostream& coco::operator<< (std::ostream & *o,* const certificate & *t*) [inline]

Stream output operator for certificates, which calls the get_contents method of the certificate.

Definition at line 112 of file api_certbase.h.

8.5.3.2 std::ostream & coco::coco::operator<< (std::ostream & *o,* const certificate & *t*) [inline]

Stream output operator for certificates, which calls the get_contents method of the certificate.

Definition at line 112 of file search_graph.cc.

8.5.3.3 std::ostream & coco::coco::operator<< (std::ostream & *o,* const delta & *t*) [inline]

Stream output operator for deltas, which calls the get_action method of the delta.

Definition at line 137 of file search_graph.cc.

8.5.3.4 std::ostream& coco::operator<< (std::ostream & *o,* const delta & *t*) [inline]

Stream output operator for deltas, which calls the get_action method of the delta.

Definition at line 137 of file api_deltabase.h.

8.6 Evaluators

Classes

- class coco::bbfunc

  The bbfunc class (black box function)
- class coco::bbfunc_expr

  The bbfunc_expr class (black box function expression)
- class coco::d1func

  The d1func class (one dimensional function)
- class coco::d1func_expr

  The d1func_expr class (one dimensional function expression)
- class coco::dag_d1func

  The dag_d1func class (one dimensional function)
- struct coco::dd1f_interval_eval_type

  Visitor data for dd1f_interval_eval.
- class coco::dd1f_interval_eval

  Forward function range evaluation.
- struct coco::polsppt_ret_type

  Return type for polsppt_eval.
- struct coco::polsppt_eval_type

  Visitor data for polsppt_eval.
- class coco::polsppt_eval

  Forward function range evaluation.
- struct coco::defdom_ret_type

  Return type for defdom_eval.
- struct coco::defdom_eval_type

  Visitor data for defdom_eval.
- class coco::defdom_eval

  Forward function range evaluation.
- class coco::prep_d_eval

  Preparation Evaluator for derivatives.
- struct coco::func_d_eval_type

  Visitor data for func_d_eval.
- class coco::func_d_eval

  Forward function evaluation with preparation of derivative data.
- struct coco::der_eval_type

  Visitor data for der_eval.
- class coco::der_eval

  Backward gradient evaluation with prepared derivative data.
- class coco::variable_indicator

  Bitmap class used to indicate variable occurrence.
- class coco::_evaluator_base

  Base class of all evaluators.
- class coco::evaluator_base

  Base class of all (non-caching) evaluators.
- class coco::cached_evaluator_base

Base class of all caching evaluators.

- class coco::forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class coco::backward_evaluator_base

  Base class of all (non-caching) backward evaluators.
- class coco::cached_forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class coco::cached_backward_evaluator_base

  Base class of all caching backward evaluators.
- struct coco::func_eval_type

  Visitor data for func_eval.
- class coco::func_eval

  Forward function evaluation.
- class coco::hsf_func

  The hsf_func class (one dimensional function)
- class coco::prep_id_eval

  Preparation Evaluator for interval derivatives.
- struct coco::func_id_eval_type

  Visitor data for func_id_eval.
- class coco::func_id_eval

  Forward function range evaluation with preparation of interval derivative data.
- struct coco::ider_eval_type

  Visitor data for ider_eval.
- class coco::ider_eval

  Backward interval gradient evaluation with prepared interval derivative data.
- struct coco::iderf_ret_type

  Visitor data for iderf_eval return value.
- struct coco::iderf_eval_type

  Visitor data for iderf_eval.
- class coco::iderf_eval

  Forward function and derivative range evaluation.
- struct coco::interval_eval_type

  Visitor data for interval_eval.
- class coco::interval_eval

  Forward function range evaluation.
- struct coco::Islope_eval_type

  Visitor data for Islope_eval.
- class coco::Islope_eval

  Forward function range evaluation.
- struct coco::func_islp_return_type

  The return type of the func_islp_eval evaluator.
- class coco::prep_islp_eval

  Preparation Evaluator for first order slopes.
- struct coco::func_islp_eval_type

  Visitor data for func_id_eval.
- class coco::func_islp_eval

Forward function range evaluation with preparation of first order slope data.

- struct coco::islp_eval_type_INTERNAL

  Visitor data for islp_eval.
- class coco::islp_eval_INTERNAL

  Backward first order slope evaluation with prepared first order slope data.
- struct coco::xxxNumber_eval_type

  Visitor data for xxxNumber_eval.
- class coco::xxxNumber_eval

  Forward function range evaluation.
- class coco::xexp_func

  The xexp_func class (one dimensional function)

Enumerations

- enum coco::d1func_monotonicity_t { coco::d1fpmon_dec = -1, coco::d1fpmon_const = 0, coco::d1fpmon_inc = 1, coco::d1fpmon_unclear = 2, coco::d1fpmon_nonmon = -2 }

  The d1func_monotonicity_t enum.

Functions

- d1func_monotonicity_t coco::operator- (d1func_monotonicity_t x)

  The d1func_monotonicity_t unary minus.
- template<class _Walker , class _Visitor >
  _Visitor::return_value coco::recursive_short_cut_walk (_Walker __w, _Visitor __f)

  Perform a recursive graph walk with possible caching and short-cuts.
- template<class _Walker , class _Visitor >
  _Visitor::return_value coco::_recursive_short_cut_walk (_Walker __w, _Visitor __f)

  Perform a recursive graph walk with possible caching and short-cuts (internal)
- template<class _Visitor , class _Walker >
  _Visitor::return_value coco::evaluate (_Visitor __v, _Walker __start)

  Evaluate an evaluator on a DAG.

8.6.1 Detailed Description

The classes and types in this section define the evaluators which can be used for gaining varios information from the models like evaluation of function and constraints, automatic differentiation,...

8.6.2 Enumeration Type Documentation

8.6.2.1 enum coco::d1func_monotonicity_t

The d1func_monotonicity_t enum encodes monotonicity information for a one-dimensional function.

Enumerator:

d1fpmon_dec

d1fpmon_const

d1fpmon_inc

d1fpmon_unclear

d1fpmon_nonmon

Definition at line 54 of file d1func.h.

8.6.3 Function Documentation

8.6.3.1 template<class _Walker , class _Visitor > _Visitor::return_value coco::coco::_recursive_short_cut_walk (_Walker __w, _Visitor __f)

This **internal** function performs a recursive pre+post order walk starting at `__w`. At every node various methods of the visitor `__f` are called:

`vinit` is called before walking for every virtual node

`vcollect` is called after a child of a virtual node has finished

`vvalue` is called to compute the return value of a virtual node

`preorder` is called before the children are visited. The return value of this method determines the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

`collect` is called everytime a child has finished. The return value of this method determines the further graph walk:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip the specified number of children.

`postorder` is called after all children have been visited

`value` is called to compute the return value for this node

`short_cut_to` is called to provide the target of the short cut if a short cut is signalled by `preorder`.

This function does not check for hitting the virtual ground node.

Definition at line 1079 of file evaluator.h.

8.6.3.2 template<class _Visitor , class _Walker > _Visitor::return_value coco::coco::evaluate (_Visitor __v, _Walker __start)

This function evaluates the evaluator `__v` on the graph, calculating the result for the node pointed at by `__start`.

Definition at line 1127 of file evaluator.h.

8.6.3.3 d1func_monotonicity_t coco::operator- (d1func_monotonicity_t *x*) [inline]

This operator inverts monotonicity for a one-dimensional function.

Definition at line 62 of file d1func.h.

8.6.3.4 template<class _Walker , class _Visitor > _Visitor::return_value coco::coco::recursive_short_cut_walk (_Walker *__w*, _Visitor *__f*)

Perform a recursive pre+post order walk starting at __w. At every node various methods of the visitor __f are called:

vinitis called before walking for every virtual node

vcollectis called after a child of a virtual node has finished

vvalueis called to compute the return value of a virtual node

preorderis called before the children are visited. The return value of this method determines the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

collectis called everytime a child has finished. The return value of this method determines the further graph walk:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip the specified number of children.

postorderis called after all children have been visited

valueis called to compute the return value for this node

short_cut_tois called to provide the target of the short cut if a short cut is signalled by preorder.

Definition at line 989 of file evaluator.h.

8.7 Module Base Classes

Classes

- class coco::graph_analyzer

  Graph analyzer base class.
- class coco::ie_return_type

  The return class of all inference engines.
- class coco::statistic_info

  Base class for all inference engine statistics classes.
- class coco::inference_module_cache_autogen

  Inference module cache auto-generate method base class.
- class coco::inference_module_cache

  Inference module cache class.
- class coco::inference_engine

  Inference engine base class.
- class coco::initializer

  Initializer base class.
- class coco::management_module

  Management module base class.
- class coco::report_module

  Report module base class.
- class coco::coco::semantics

  Expression Semantics.

Defines

- #define new_cb_ref() _new_cb_ref((void∗)this)

Typedefs

- typedef unsigned int cb_ref_handle
- typedef ie_return_type coco::ga_return_type

  Graph analyzer return type.

Functions

- cb_ref_handle _new_cb_ref (void ∗dest)
- int delete_cb_ref (cb_ref_handle ref)
- void ∗ cb_ref (cb_ref_handle ref)
- bool coco::operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.
- bool coco::operator== (const convex_e &__c, const convex_info &__d)
- bool coco::operator== (const convex_info &__c, const convex_e &__d)
- bool coco::operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.
- bool coco::operator!= (const convex_e &__c, const convex_info &__d)

- bool coco::operator!= (const convex_info &__c, const convex_e &__d)
- std::ostream & coco::operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.
- bool coco::coco::operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.
- bool coco::coco::operator== (const convex_e &__c, const convex_info &__d)
- bool coco::coco::operator== (const convex_info &__c, const convex_e &__d)
- bool coco::coco::operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.
- bool coco::coco::operator!= (const convex_e &__c, const convex_info &__d)
- bool coco::coco::operator!= (const convex_info &__c, const convex_e &__d)
- std::ostream & coco::coco::operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.
- std::ostream & coco::coco::operator<< (std::ostream &o, const semantics &__s)

  C++ stream output operator for semantics.
- std::ostream & coco::operator<< (std::ostream &o, const termination_reason &__x)

  C++ stream output operator for the termination_reason.

8.7.1 Detailed Description

The classes and types in this section define the base classes for the five COCONUT module types inference engines, graph analyzers, management modules, initializers, and report modules.

8.7.2 Define Documentation

8.7.2.1 #define new_cb_ref( ) _new_cb_ref((void*)this)

This function returns a new cb_ref_handle which points to the inference engine object which calls this macro.

Definition at line 42 of file callback_ref.h.

8.7.3 Typedef Documentation

8.7.3.1 typedef unsigned int cb_ref_handle

This is the type of the handles handed out for backreference from C or Fortran code to C++ objects in wrapper headers for inference engines in C or Fortran.

Definition at line 36 of file callback_ref.h.

8.7.3.2 typedef ie_return_type coco::ga_return_type

This is the type which is returned by the analyze method of any graph_analyzer.

Definition at line 74 of file graph_analyzer.h.

8.7.4 Function Documentation

8.7.4.1 cb_ref_handle _new_cb_ref (void * *dest*)

This function returns a new cb_ref_handle which points to the `dest` parameter provided. This function should not be called directly. Rather the new_cb_ref macro should be used, which inserts the `this` pointer of the inference engine object into the `dest` parameter.

Definition at line 46 of file callback_ref.cc.

8.7.4.2 void* cb_ref (cb_ref_handle *ref*)

This function returns the pointer to the inference engine object corresponding to the cb_ref_handle `ref`. The pointer is of type `void*` and has to be cast to the appropriate C++ type.

Definition at line 65 of file callback_ref.cc.

8.7.4.3 int delete_cb_ref (cb_ref_handle *ref*)

This function deletes the cb_ref_handle `ref`. It returns 0 on success and <0 on failure.

Definition at line 53 of file callback_ref.cc.

8.7.4.4 bool coco::operator!= (const convex_e & *__c*, const convex_e & *__d*) [inline]

Disequality comparison operator. Note that only the convex_info part counts!

Definition at line 195 of file convex_info.h.

8.7.4.5 bool coco::coco::operator!= (const convex_e & *__c*, const convex_e & *__d*) [inline]

Disequality comparison operator. Note that only the convex_info part counts!

Definition at line 195 of file search_graph.cc.

8.7.4.6 bool coco::operator!= (const convex_e & *__c*, const convex_info & *__d*) [inline]

Disequality comparison operator with a convex_info.

Definition at line 203 of file convex_info.h.

8.7.4.7 bool coco::coco::operator!= (const convex_e & *__c*, const convex_info & *__d*) [inline]

Disequality comparison operator with a convex_info.

Definition at line 203 of file search_graph.cc.

8.7.4.8 bool coco::coco::operator!= (const convex_info & *__c*, const convex_e & *__d*) [inline]

Disequality comparison operator with a convex_info.

Definition at line 211 of file search_graph.cc.

8.7.4.9 bool coco::operator!= (const convex_info & __c, const convex_e & __d) [inline]

Disequality comparison operator with a convex_info.

Definition at line 211 of file convex_info.h.

8.7.4.10 std::ostream& coco::operator<< (std::ostream & o, const termination_reason & __x) [inline]

This operator writes a termination_reason (the message) to an ostream.

Definition at line 90 of file termreason.h.

8.7.4.11 std::ostream& coco::coco::operator<< (std::ostream & o, const convex_e & __s) [inline]

This operator writes a convex_e in .dag format to an ostream.

Definition at line 219 of file search_graph.cc.

8.7.4.12 std::ostream& coco::operator<< (std::ostream & o, const convex_e & __s) [inline]

This operator writes a convex_e in .dag format to an ostream.

Definition at line 219 of file convex_info.h.

8.7.4.13 std::ostream & coco::coco::operator<< (std::ostream & o, const semantics & __s)

This operator writes a semantics expression in .dag format to an ostream.

Definition at line 224 of file semantics.cc.

8.7.4.14 bool coco::operator== (const convex_e & __c, const convex_e & __d) [inline]

Equality comparison operator. Note that only the convex_info part counts!

Definition at line 171 of file convex_info.h.

8.7.4.15 bool coco::coco::operator== (const convex_e & __c, const convex_e & __d) [inline]

Equality comparison operator. Note that only the convex_info part counts!

Definition at line 171 of file search_graph.cc.

8.7.4.16 bool coco::operator== (const convex_e & __c, const convex_info & __d) [inline]

Equality comparison operator with a convex_info.

Definition at line 179 of file convex_info.h.

8.7.4.17 bool coco::coco::operator== (const convex_e & __c, const convex_info & __d) [inline]

Equality comparison operator with a convex_info.

Definition at line 179 of file search_graph.cc.

8.7.4.18 bool coco::operator== (const convex_info & __c, const convex_e & __d) [inline]

Equality comparison operator with a convex_info.

Definition at line 187 of file convex_info.h.

8.7.4.19 bool coco::coco::operator== (const convex_info & __c, const convex_e & __d) [inline]

Equality comparison operator with a convex_info.

Definition at line 187 of file search_graph.cc.

8.8 Database Interface

Classes

- class coco::dbt_row

  This type is used to hold one row in some table of the search database.
- class coco::point_check_feasibility

  Stored procedure checking the feasibility of a point.
- class coco::box_check_intersection

  Stored procedure checking whether a box intersects the work_node's box.
- class coco::coco::work_node_context

  The evaluation context when retrieving from the search database.
- class coco::search_graph_context

  An evaluation context when retrieving from the search database.

Functions

- vdbl::tableid coco::get_point_table (vdbl::database *ptr, vdbl::standard_table *&ptb, vdbl::userid)

  Get the point from the search database.

8.8.1 Detailed Description

The classes and types in this section provide the interface to the VDBL (Vienna DataBase Library).

8.8.2 Function Documentation

8.8.2.1 vdbl::tableid coco::get_point_table (vdbl::database * *ptr*, vdbl::standard_table *& *ptb*, vdbl::userid)

This function returns the tableid of and gets a pointer `ptb` to the "point" table of the search database `ptr`. If the point table does not yet exist, it is created. The point table has the following column structure: name type default sem x vector<double> N/A the coordinates L_mult vector<double> empty the multipliers f double +COCO_INF the objective value kappa double 1. the Kar.-John mult. best bool false best point in box verified bool false verified point feasible bool stored proc. feasible point optimal bool false local optimum global bool false from global box relaxation bool false from a relaxation class unsigned int 0 point class

8.9 Advanced Utilities

Classes

- struct coco::coconut_random_f
- class coco::wnc_hook_base

  Base class for the work node computation hooks.
- class coco::sparsity_visitor

  Preorder visitor which calculates sparsity structures.
- class coco::sparsity3_visitor

  Preorder visitor which calculates sparsity structures.

Defines

- #define coconut_random random
- #define coconut_seed(A) srandom(A)
- #define COCONUT_RAND_MAX RAND_MAX
- #define COCONUT_RRAND_MIN -1.e08
- #define COCONUT_RRAND_MAX +1.e08
- #define coconut_init_random() coconut_seed(INIT_SEED)
- #define COCONUT_RRAND_MIN_BETA 0.01
- #define COCONUT_RRAND_MIN_ALPHA 0.5
- #define COCONUT_RRAND_MAX_ALPHA 0.99

Typedefs

- typedef long int coco::rand_t
- typedef std::pair< unsigned int, unsigned int > coco::ipair

  Row-column index pair.
- typedef std::set< ipair > coco::splist

  Sparsity information.
- typedef std::triple< unsigned int, unsigned int, unsigned int > coco::itriple

  Row-column-depth index triple.
- typedef std::set< itriple > coco::stlist

  Sparsity information.

Functions

- double coco::d_random ()
- double coco::r_random (double l, double u, double beta=10.0, double alpha=0.9)
- double coco::r_random (const interval &_i, double beta=10.0, double alpha=0.9)
- interval coco::i_random (double l, double r, double beta=10.0, double alpha=0.9)
- interval coco::i_random (const interval &_i, double beta=10.0, double alpha=0.9)
- interval coco::i_random ()
- void coco::get_leaves_vector (const search_graph &__sgraph, std::vector< si_ri_pair > &v, search-_inspector *sfoc=NULL)

  Sparsity calculation (Hessian and Jacobian)

- void coco::sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity (const model &DAG, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity (const model &DAG, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity3 (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity3 (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity3 (const model &DAG, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)

- void coco::sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)

- void coco::sparsity3 (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)

- void coco::sparsity3 (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)

- void coco::sparsity3 (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity3 (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)

- void coco::sparsity3 (const model &DAG, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity3 (const model &DAG, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)

8.9.1 Detailed Description

The classes, types, and functions in this section provide additional utilities, which can be used to support the solution process.

8.9.2 Define Documentation

8.9.2.1 #define coconut_init_random( ) coconut_seed(INIT_SEED)

This function initializes the random number generator

Definition at line 85 of file coconut_random.h.

8.9.2.2 #define COCONUT_RAND_MAX RAND_MAX

This constant is the maximum number returned by coconut_random.

Definition at line 63 of file coconut_random.h.

8.9.2.3 #define coconut_random random

The function coconut_random returns a random integer number between 0 and COCONUT_RAND_MAX.

Definition at line 48 of file coconut_random.h.

8.9.2.4 #define COCONUT_RRAND_MAX +1.e08

This constant defines the maximal `double` number returned by r_random.

Definition at line 74 of file coconut_random.h.

8.9.2.5 #define COCONUT_RRAND_MAX_ALPHA 0.99

This constant defines the largest allowed value of the parameter `alpha` in r_random.

Definition at line 120 of file coconut_random.h.

8.9.2.6 #define COCONUT_RRAND_MIN -1.e08

This constant defines the minimal `double` number returned by r_random.

Definition at line 69 of file coconut_random.h.

8.9.2.7 #define COCONUT_RRAND_MIN_ALPHA 0.5

This constant defines the smallest allowed value of the parameter `alpha` in r_random.

Definition at line 115 of file coconut_random.h.

8.9.2.8 #define COCONUT_RRAND_MIN_BETA 0.01

This constant defines the smallest allowed value of the parameter `beta` in r_random.

Definition at line 110 of file coconut_random.h.

8.9.2.9 #define coconut_seed(*A*) srandom(A)

The function coconut_seed sets its `unsigned int` argument as the seed for a new sequence of pseudo-random numbers returned by coconut_random.

Definition at line 58 of file coconut_random.h.

8.9.3 Typedef Documentation

8.9.3.1 typedef std::pair<unsigned int, unsigned int> coco::ipair

This type holds a row-column index pair for a sparse matrix.

Definition at line 48 of file sparsity.cc.

8.9.3.2 typedef std::triple<unsigned int, unsigned int, unsigned int> coco::itriple

This type holds a row-column-depth index triple for a sparse 3-tensor.

Definition at line 48 of file sparsity3.cc.

8.9.3.3 typedef long int coco::rand_t

This is the type returned by the coconut_random function.

Definition at line 90 of file coconut_random.h.

8.9.3.4 typedef std::set<ipair> coco::splist

This type holds a list of row-column index pairs for a sparse matrix, the matrix' sparsity structure.

Definition at line 54 of file sparsity.cc.

8.9.3.5 typedef std::set<itriple> coco::stlist

This type holds a list of row-column-depth index triples for a sparse 3-tensor, the tensor's sparsity structure.

Definition at line 54 of file sparsity3.cc.

8.9.4 Function Documentation

8.9.4.1 double coco::d_random ( ) [inline]

The function d_random returns a random `double` number in [0,1[.

Definition at line 95 of file coconut_random.h.

8.9.4.2 void coco::get_leaves_vector (const search_graph & *__sgraph*, std::vector< si_ri_pair > & *v*, search_inspector * *sfoc*)

Definition at line 43 of file search_graph_funcs.cc.

8.9.4.3 interval coco::i_random (double *l*, double *r*, double *beta* = `10.0`, double *alpha* = `0.9`) [inline]

The function i_random returns a random `interval` subset of [l,r]. The role of the parameters `beta` and `alpha` is the same as in the workhorse function r_random(double,double,double,double).

Definition at line 219 of file coconut_random.h.

8.9.4.4 interval coco::i_random (const interval & *_i*, double *beta* = `10.0`, double *alpha* = `0.9`) [inline]

The function i_random returns a random `interval` subset of `_i`. The role of the parameters `beta` and `alpha` is the same as in the workhorse function r_random(double,double,double,double).

Definition at line 231 of file coconut_random.h.

8.9.4.5 interval coco::i_random ( ) [inline]

The function i_random returns a random `interval` subset of [0,1].

Definition at line 240 of file coconut_random.h.

8.9.4.6 double coco::r_random (double *l*, double *u*, double *beta* = `10.0`, double *alpha* = `0.9`) [inline]

The function r_random returns a random `double` number in the interval {[l,r]}. Actually, there are never numbers generated, which are outside the interval [COCONUT_RRAND_MIN,COCONUT_RRAND_MAX]. The numbers are generated along the graph of the function h(t) = (-alpha∗beta + t) / ((1/alpha-2)∗beta + t), if t > beta, = alpha∗t/beta, if t [-beta,beta], = (alpha∗beta + t) / ((1/alpha-2)∗beta - t), if t < -beta, where alpha is a positive parameter close to, but smaller than 1 (COCONUT_RRAND_MIN_ALPHA <= alpha <= COCONUT_RRAND_MAX_ALPHA < 1), and beta is a positive parameter (beta >= COCONUT_RRAND_MIN_BETA > 0). h is a smooth probability distribution function in [0,inf], and behaves similarly in [-inf,0] because of h(-t)=-h(t). The parameter `beta` specifies that in the interval [-beta,beta] the point is chosen from uniform distribution. Outside this interval, h(t)->1 from below when t->inf, and h(t)->-1

from above when t->-inf. The parameter `alpha` specifies how fast |h(t)|-1 decays while |t|->inf. Also, since h(beta) = alpha, alpha specifies that if a random point is generated in [-inf,inf] (or [0,inf] or [-inf,0], resp.), then it falls in [-beta,beta] with probability alpha .

Definition at line 168 of file coconut_random.h.

8.9.4.7 double coco::r_random (const interval & *_i*, double *beta* = `10.0`, double *alpha* = `0.9`) [inline]

The function r_random returns a random `double` number in the interval `_i`. Actually, there are never numbers generated, which are outside the interval [COCONUT_RRAND_MIN,COCONUT_RRAND_-MAX]. The role of the parameters `beta` and `alpha` is the same as in the workhorse function r_-random(double,double,double,double).

Definition at line 208 of file coconut_random.h.

8.9.4.8 void coco::sparsity (const model & *DAG*, unsigned int & *nnz_H*, unsigned int *& *H_ridx*, unsigned int *& *H_cidx*, bool *org_only*, bool *H_upper_only*, sp_indexing *indexing*, bool *var_one_based*)

This function computes the sparsity structure of the Hessian of the Lagrangean. Its parameters are:

- DAG: The expression DAG (input),
- nnz_H: The number of nonzero entries in the Hessian (output),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output, the array is allocated using `new` by the function),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output, the array is allocated using `new` by the function),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.9 void coco::sparsity (const model & *DAG*, std::vector< unsigned int > & *H_ridx*, std::vector< unsigned int > & *H_cidx*, std::vector< unsigned int > & *J_funidx*, std::vector< unsigned int > & *J_varidx*, bool *J_with_obj*, bool *J_obj_is_first*, bool *org_only*, bool *H_upper_only*, sp_indexing *indexing*, bool *fun_one_based*, bool *var_one_based*)

This function computes the sparsity structures of the Hessian of the Lagrangean and the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output),

- J_funidx: The function index vector of the Jacobian of the constraints (output),
- J_varidx: The variable index vector of the Jacobian of the constraints (output),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

Definition at line 646 of file sparsity.cc.

8.9.4.10 void coco::sparsity (const model & *DAG*, std::vector< unsigned int > & *H_ridx*, std::vector< unsigned int > & *H_cidx*, bool *org_only*, bool *H_upper_only*, sp_indexing *indexing*, bool *var_one_based*)

This function computes the sparsity structure of the Hessian of the Lagrangean. Its parameters are:

- DAG: The expression DAG (input),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

Definition at line 659 of file sparsity.cc.

8.9.4.11 void coco::sparsity (const model & *DAG*, std::vector< unsigned int > & *J_funidx*, std::vector< unsigned int > & *J_varidx*, bool *J_with_obj*, bool *J_obj_is_first*, bool *org_only*, sp_indexing *indexing*, bool *fun_one_based*, bool *var_one_based*)

This function computes the sparsity structure of the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),

- J_funidx: The function index vector of the Jacobian of the constraints (output),
- J_varidx: The variable index vector of the Jacobian of the constraints (output),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

Definition at line 669 of file sparsity.cc.

8.9.4.12 void coco::sparsity (const model & *DAG,* unsigned int & *nnz_H,* unsigned int *& *H_ridx,* unsigned int *& *H_cidx,* unsigned int & *nnz_J,* unsigned int *& *J_funidx,* unsigned int *& *J_varidx,* bool *J_with_obj,* bool *J_obj_is_first,* bool *org_only,* bool *H_upper_only,* sp_indexing *indexing,* bool *fun_one_based,* bool *var_one_based*)

This function computes the sparsity structures of the Hessian of the Lagrangean and the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- nnz_H: The number of nonzero entries in the Hessian (output),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output, the array is allocated using `new` by the function),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output, the array is allocated using `new` by the function),
- nnz_J: The number of nonzero entries in the Jacobian (output),
- J_funidx: The function index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_varidx: The variable index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),

- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

Definition at line 680 of file sparsity.cc.

8.9.4.13 void coco::sparsity (const model & *DAG,* int & *nnz_H,* int *& *H_ridx,* int *& *H_cidx,* bool *org_only,* bool *H_upper_only,* sp_indexing *indexing,* bool *var_one_based*)

This function computes the sparsity structure of the Hessian of the Lagrangean. Its parameters are:

- DAG: The expression DAG (input),
- nnz_H: The number of nonzero entries in the Hessian (output),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output, the array is allocated using `new` by the function),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output, the array is allocated using `new` by the function),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

Definition at line 747 of file sparsity.cc.

8.9.4.14 void coco::sparsity (const model & *DAG,* unsigned int & *nnz_J,* unsigned int *& *J_funidx,* unsigned int *& *J_varidx,* bool *J_with_obj,* bool *J_obj_is_first,* bool *org_only,* sp_indexing *indexing,* bool *fun_one_based,* bool *var_one_based*)

This function computes the sparsity structures of the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- nnz_J: The number of nonzero entries in the Jacobian (output),
- J_funidx: The function index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_varidx: The variable index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),

- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

Definition at line 756 of file sparsity.cc.

8.9.4.15 void coco::sparsity3 (const model & *DAG*, std::vector< unsigned int > & *H_ridx*, std::vector< unsigned int > & *H_cidx*, std::vector< unsigned int > & *J_funidx*, std::vector< unsigned int > & *J_varidx*, bool *J_with_obj*, bool *J_obj_is_first*, bool *org_only*, bool *H_upper_only*, sp_indexing *indexing*, bool *fun_one_based*, bool *var_one_based*)

This function computes the sparsity3 structures of the Hessian of the Lagrangean and the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output),
- J_funidx: The function index vector of the Jacobian of the constraints (output),
- J_varidx: The variable index vector of the Jacobian of the constraints (output),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.16 void coco::sparsity3 (const model & *DAG*, std::vector< unsigned int > & *H_ridx*, std::vector< unsigned int > & *H_cidx*, bool *org_only*, bool *H_upper_only*, sp_indexing *indexing*, bool *var_one_based*)

This function computes the sparsity3 structure of the Hessian of the Lagrangean. Its parameters are:

- DAG: The expression DAG (input),

- H_ridx: The row index vector of the Hessian of the Lagrangean (output),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.17 void coco::sparsity3 (const model & *DAG,* std::vector< unsigned int > & *J_funidx,* std::vector< unsigned int > & *J_varidx,* bool *J_with_obj,* bool *J_obj_is_first,* bool *org_only,* sp_indexing *indexing,* bool *fun_one_based,* bool *var_one_based*)

This function computes the sparsity3 structure of the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- J_funidx: The function index vector of the Jacobian of the constraints (output),
- J_varidx: The variable index vector of the Jacobian of the constraints (output),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.18 void coco::sparsity3 (const model & *DAG,* unsigned int & *nnz_H,* unsigned int *& *H_ridx,* unsigned int *& *H_cidx,* unsigned int & *nnz_J,* unsigned int *& *J_funidx,* unsigned int *& *J_varidx,* bool *J_with_obj,* bool *J_obj_is_first,* bool *org_only,* bool *H_upper_only,* sp_indexing *indexing,* bool *fun_one_based,* bool *var_one_based*)

This function computes the sparsity3 structures of the Hessian of the Lagrangean and the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- nnz_H: The number of nonzero entries in the Hessian (output),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output, the array is allocated using `new` by the function),

- H_cidx: The column index vector of the Hessianof the Lagrangean (output, the array is allocated using `new` by the function),
- nnz_J: The number of nonzero entries in the Jacobian (output),
- J_funidx: The function index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_varidx: The variable index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.19 void coco::sparsity3 (const model & *DAG,* int & *nnz_H,* int *& *H_ridx,* int *& *H_cidx,* int & *nnz_J,* int *& *J_funidx,* int *& *J_varidx,* bool *J_with_obj,* bool *J_obj_is_first,* bool *org_only,* bool *H_upper_only,* sp_indexing *indexing,* bool *fun_one_based,* bool *var_one_based*)

This function computes the sparsity3 structures of the Hessian of the Lagrangean and the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- nnz_H: The number of nonzero entries in the Hessian (output),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output, the array is allocated using `new` by the function),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output, the array is allocated using `new` by the function),
- nnz_J: The number of nonzero entries in the Jacobian (output),
- J_funidx: The function index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_varidx: The variable index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),

- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.20 void coco::sparsity3 (const model & *DAG,* unsigned int & *nnz_H,* unsigned int *** & *H_ridx,* unsigned int *** & *H_cidx,* bool *org_only,* bool *H_upper_only,* sp_indexing *indexing,* bool *var_one_based*)

This function computes the sparsity3 structure of the Hessian of the Lagrangean. Its parameters are:

- DAG: The expression DAG (input),
- nnz_H: The number of nonzero entries in the Hessian (output),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output, the array is allocated using `new` by the function),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output, the array is allocated using `new` by the function),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.21 void coco::sparsity3 (const model & *DAG,* int & *nnz_H,* int *** & *H_ridx,* int *** & *H_cidx,* bool *org_only,* bool *H_upper_only,* sp_indexing *indexing,* bool *var_one_based*)

This function computes the sparsity3 structure of the Hessian of the Lagrangean. Its parameters are:

- DAG: The expression DAG (input),
- nnz_H: The number of nonzero entries in the Hessian (output),
- H_ridx: The row index vector of the Hessian of the Lagrangean (output, the array is allocated using `new` by the function),
- H_cidx: The column index vector of the Hessianof the Lagrangean (output, the array is allocated using `new` by the function),

- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- H_upper_only: This bool specifies whether of the (symmetric!) Hessian only the row-column pairs above the diagonal shall be given (input),
- indexing: This enum specifies the requested indexing structure (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.22 void coco::sparsity3 (const model & *DAG,* unsigned int & *nnz_J,* unsigned int *& *J_funidx,* unsigned int *& *J_varidx,* bool *J_with_obj,* bool *J_obj_is_first,* bool *org_only,* sp_indexing *indexing,* bool *fun_one_based,* bool *var_one_based*)

This function computes the sparsity3 structures of the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- nnz_J: The number of nonzero entries in the Jacobian (output),
- J_funidx: The function index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_varidx: The variable index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

8.9.4.23 void coco::sparsity3 (const model & *DAG,* int & *nnz_J,* int *& *J_funidx,* int *& *J_varidx,* bool *J_with_obj,* bool *J_obj_is_first,* bool *org_only,* sp_indexing *indexing,* bool *fun_one_based,* bool *var_one_based*)

This function computes the sparsity3 structures of the Jacobian of the constraints. Its parameters are:

- DAG: The expression DAG (input),
- nnz_J: The number of nonzero entries in the Jacobian (output),
- J_funidx: The function index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),

- J_varidx: The variable index vector of the Jacobian of the constraints (output, the array is allocated using `new` by the function),
- J_with_obj: This bool determines whether the objective function is included in the Jacobian (input),
- J_obj_is_first: This bool determines whether the objective function is has the first function index (rather than the last) (input),
- org_only: This bool determines whether constraints and variables from the (symbolic!) KJ-conditions are left out (input),
- indexing: This enum specifies the requested indexing structure (input),
- fun_one_based: This boolean determines whether the function index in the Jacobian starts with 1 or with 0 (input),
- var_one_based: This boolean determines whether all indices depending on variables start with 1 or with 0 (input).

9 Namespace Documentation

9.1 coco Namespace Reference

the main namespace of the COCONUT API

Namespaces

- namespace coco
- namespace num

Classes

- struct analyticd_eval_type
- class analyticd_eval
- class annotation

  Annotations for Models.
- class annotation_undelta

  the undelta class for annotation changes
- class annotation_delta

  the delta class for annotation changes
- class no_certificate

  The not-certified certificate.
- class split_certificate

  The certificate for deltas formed by splits.
- class compound_certificate

  The certificate for deltas formed by compressing bound_delta entries.
- class rigorous_module_certificate

  The certificate for deltas computed by rigorous inference engines.
- class certificate

  The certificate class (certifies deltas for rigorous mode operation)
- class certificate_base

  Base class for the certificates.
- class delta_get_action

  Stored procedure class for computing the delta action specifier.
- class delta

  The delta class (updates to work nodes)
- class delta_base

  Base class for the deltas.
- class undelta

  The undelta class (undo of updates to work nodes)
- class undelta_base

  Base class for the undeltas.
- class api_exception

  API exception class.
- class nyi_exception

Not Yet Implemented exception class.

- class basic_alltype

  The basic alltype which can hold any of a number of basic types.
- class bbfunc

  The bbfunc class (black box function)
- class bbfunc_expr

  The bbfunc_expr class (black box function expression)
- struct b_interval_eval_type
- class b_interval_eval
- class bound_undelta

  The bound undelta class for undoing changes to the node bounds in a model.
- class bound_delta

  The bound delta class for changing the node bounds within a model.
- class boxes_delta

  A delta class which adds new boxes to the search database.
- struct cinterval_eval_type
- class cinterval_eval
- struct coconut_random_f
- class work_node_comp_hook

  The work_node_comp_hook class (work node computation hook)
- class wnc_hook_base

  Base class for the work node computation hooks.
- class control_data

  The class for communicating parameter information to COCONUT modules.
- class convex_e

  Convexity information.
- struct cmp_point_info_i
- struct cmp_point_info_s
- class d1func

  The d1func class (one dimensional function)
- class d1func_expr

  The d1func_expr class (one dimensional function expression)
- union double_to_uint64_u
- class testpoly_func
- class one_over_x_func
- class dag_undelta

  The DAG undelta class for undoing changes to the DAG of a model.
- class dag_delta

  The DAG delta class for performing changes to the DAG of a model.
- class dag_d1func

  The dag_d1func class (one dimensional function)
- class compare_intervals
- struct dd1f_interval_eval_type

  Visitor data for dd1f_interval_eval.
- class dd1f_interval_eval

  Forward function range evaluation.
- struct polsppt_ret_type

Return type for polsppt_eval.

- struct polsppt_eval_type

  Visitor data for polsppt_eval.
- class polsppt_eval

  Forward function range evaluation.
- class datamap

  The base class for communicating with COCONUT modules.
- class dbt_row

  This type is used to hold one row in some table of the search database.
- class dbccmps_gt
- class dbccmps_absgt
- class dbccmps_lt
- class dbccmps_abslt
- class dbccmps_true
- class dbccmps_false
- struct dbccmp_true
- struct dbccmp_false
- class point_check_feasibility

  Stored procedure checking the feasibility of a point.
- class box_check_intersection

  Stored procedure checking whether a box intersects the work_node's box.
- struct defdom_ret_type

  Return type for defdom_eval.
- struct defdom_problem_point
- struct defdom_eval_type

  Visitor data for defdom_eval.
- class defdom_eval

  Forward function range evaluation.
- class prep_d_eval

  Preparation Evaluator for derivatives.
- struct func_d_eval_type

  Visitor data for func_d_eval.
- class func_d_eval

  Forward function evaluation with preparation of derivative data.
- struct der_eval_type

  Visitor data for der_eval.
- class der_eval

  Backward gradient evaluation with prepared derivative data.
- struct dfunc_eval_rettype
- struct dfunc_eval_type
- class dfunc_eval
- class diameter_comp_hook

  The log-volume computation hook (work node computation hook)
- class diffI
- class diffNumber
- class variable_indicator

  Bitmap class used to indicate variable occurrence.
- class _evaluator_base

Base class of all evaluators.

- class evaluator_base

  Base class of all (non-caching) evaluators.
- class cached_evaluator_base

  Base class of all caching evaluators.
- class forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class backward_evaluator_base

  Base class of all (non-caching) backward evaluators.
- class cached_forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class cached_backward_evaluator_base

  Base class of all caching backward evaluators.
- class expression_node

  The base class for a node in the expression DAGs.
- class expression_print_visitor
- struct func_eval_type

  Visitor data for func_eval.
- class func_eval

  Forward function evaluation.
- class graph_analyzer_exception

  Graph analyzer exception class.
- class graph_analyzer

  Graph analyzer base class.
- class graphorder_visitor

  This visitor class is used for computing a graph order.
- struct hessPreparationEvaluatorType
- class hessPreparationEvaluator
- struct hessForwardEvaluatorReturnValue
- struct hessForwardEvaluatorType
- class hessForwardEvaluator
- struct hessBackwardEvaluatorType
- class hessBackwardEvaluator
- class hessNumber
- class hsf_func

  The hsf_func class (one dimensional function)
- class prep_id_eval

  Preparation Evaluator for interval derivatives.
- struct func_id_eval_type

  Visitor data for func_id_eval.
- class func_id_eval

  Forward function range evaluation with preparation of interval derivative data.
- struct ider_eval_type

  Visitor data for ider_eval.
- class ider_eval

  Backward interval gradient evaluation with prepared interval derivative data.
- struct iderf_ret_type

Visitor data for iderf_eval return value.

- struct iderf_eval_type

  Visitor data for iderf_eval.
- class iderf_eval

  Forward function and derivative range evaluation.
- class ie_return_type

  The return class of all inference engines.
- class statistic_info

  Base class for all inference engine statistics classes.
- struct ihessPreparationEvaluatorType
- class ihessPreparationEvaluator
- struct ihessForwardEvaluatorReturnValue
- struct ihessForwardEvaluatorType
- class ihessForwardEvaluator
- struct ihessBackwardEvaluatorType
- class ihessBackwardEvaluator
- class ihessNumber
- class inference_module_cache_autogen

  Inference module cache auto-generate method base class.
- class inference_module_cache

  Inference module cache class.
- struct infbound_eval_type
- class infbound_eval
- class infeasible_undelta

  The infeasible undelta class for undoing changes to the feasibility of a model.
- class infeasible_delta

  The infeasible delta class for marking a model as infeasible.
- class inference_engine_comp_hook

  The inference engine meta computation hook (work node computation hook)
- class inference_engine_exception

  Inference engine exception class.
- class inference_engine

  Inference engine base class.
- class info_contents

  The class for returning additional information from inference modules.
- class initializer_exception

  Initializer exception class.
- class initializer

  Initializer base class.
- struct interval_eval_type

  Visitor data for interval_eval.
- class interval_eval

  Forward function range evaluation.
- struct checking_my
- struct my_rounded_math
- struct interval_st

  Constructor-free interval.

- class interval

  Interval wrapper class.
- class interval_set
- struct Islope_eval_type

  Visitor data for Islope_eval.
- class Islope_eval

  Forward function range evaluation.
- struct prep_islp2_eval_type
- class prep_islp2_eval
- struct func_islp2_eval_ret_type
- struct func_islp2_eval_type
- class func_islp2_eval
- struct islp2_eval_type_INTERNAL
- class islp2_eval_INTERNAL
- struct func_islp_return_type

  The return type of the func_islp_eval evaluator.
- class prep_islp_eval

  Preparation Evaluator for first order slopes.
- struct func_islp_eval_type

  Visitor data for func_id_eval.
- class func_islp_eval

  Forward function range evaluation with preparation of first order slope data.
- struct islp_eval_type_INTERNAL

  Visitor data for islp_eval.
- class islp_eval_INTERNAL

  Backward first order slope evaluation with prepared first order slope data.
- struct ithirdderPreparationEvaluatorType
- class ithirdderPreparationEvaluator
- struct ithirdderForwardEvaluatorReturnValue
- struct ithirdderForwardEvaluatorType
- class ithirdderForwardEvaluator
- struct ithirdderBackwardEvaluatorType
- class ithirdderBackwardEvaluator
- class locopt_ret_record
- class logvol_comp_hook

  The log-volume computation hook (work node computation hook)
- class memory_comp_hook

  The memory computation hook (work node computation hook)
- class management_module_exception

  Management module exception class.
- class management_module

  Management module base class.
- class model

  The model class (an attributed DAG of expression nodes, lowest class in the model hierarchy)
- class model_iddata

  The model id-data class (the topmost in the model class hierarchy)
- class model_gid

  Model Group Data Class (middle class in the model hierarchy)

- class d1func_visitor_0
- class d1func_visitor_1
- struct xxxNumber_eval_type

  Visitor data for xxxNumber_eval.
- class xxxNumber_eval

  Forward function range evaluation.
- class objbounds_comp_hook

  The objective-bounds computation hook (work node computation hook)
- class pending_status_comp_hook

  The pending status computation hook (work node computation hook)
- class calc_pf_star

  Stored procedure calculating the pf∗ value of a box.
- class pfstar_hook

  The pfstar computation hook (work node computation hook)
- class point_delta

  A delta class which adds new points to the search database.
- class proj_rational
- class projective_interval
- class report_module_exception

  Report module exception class.
- class report_module

  Report module base class.
- class sum_deltas

  Pre-post visitor for summing up all the deltas during work node extraction.
- class __sg_anc_visitor
- class search_graph

  The search graph.
- class search_node

  Base type of the nodes in the search graph.
- class delta_node

  Class holding the delta nodes in the search graph.
- class full_node

  Class holding the full nodes in the search graph.
- class work_node

  Work node, which is passed to the inference engines.
- class semantics

  Expression Semantics.
- class semantics_undelta

  The semantics undelta class for undoing changes to the node semantics in a model.
- class semantics_delta

  The semantics delta class for changing the node semantics within a model.
- class search_graph_context

  An evaluation context when retrieving from the search database.
- class sparsity_visitor

  Preorder visitor which calculates sparsity structures.
- class sparsity3_visitor

  Preorder visitor which calculates sparsity structures.

- class split_undelta

  The split undelta class for removing proposed splits from a work_node.
- class split_delta

  The split delta class for proposing useful splits.
- class table_delta

  The base class for all deltas adding information to the search database.
- class termination_reason

  This class holds the reason of termination of inference and management modules.
- struct thirdderPreparationEvaluatorType
- class thirdderPreparationEvaluator
- struct thirdderForwardEvaluatorReturnValue
- struct thirdderForwardEvaluatorType
- class thirdderForwardEvaluator
- struct thirdderBackwardEvaluatorType
- class thirdderBackwardEvaluator
- class vol_comp_hook

  The volume computation hook (work node computation hook)
- class work_node_context

  The evaluation context when retrieving from the search database.
- class xexp_func

  The xexp_func class (one dimensional function)

Typedefs

- typedef basic_alltype additional_info_u

  type definition for backwards compatibility with earlier API versions
- typedef analyticd(∗ analyticd_evaluator)(const std::vector< analyticd > ∗__x, const variable_indicator &__v)
- typedef std::vector< std::pair < vdbl::tableid, vdbl::rowid > > trvec
- typedef vdbl::rowid delta_id

  The class for delta ids.
- typedef b_interval(∗ b_interval_evaluator)(const std::vector< b_interval > ∗__x, const variable_indicator &__v)
- typedef cinterval(∗ cinterval_evaluator)(const std::vector< cinterval > ∗__x, const variable_indicator &__v)
- typedef long int rand_t
- typedef enum coco::tristate_e tristate
- typedef uint32_t search_node_id

  Type of the unique search node identifier.
- typedef interval(∗ dd1f_interval_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)
- typedef polsppt_ret_type(∗ polsppt_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)
- typedef std::triple < d1func::d1func_point_t, diffI, int > polsppt_info
- typedef std::multimap < unsigned int, polsppt_info > polsppt_info_map
- typedef std::map< interval, polsppt_info_map, compare_intervals > polsppt_interval_map
- typedef std::multimap < unsigned int, polsppt_info_map::const_iterator > polsppt_map_map
- typedef std::multimap < unsigned int, defdom_problem_point > defdom_map

- typedef defdom_ret_type(∗ defdom_evaluator)(const std::vector< interval > ∗__x, defdom_map &__m, const variable_indicator &__v)
- typedef double(∗ func_evaluator)(const std::vector< double > ∗__x, const variable_indicator &__v)
- typedef interval rhs_t
- typedef std::vector< void ∗ > evaluator_v
- typedef ie_return_type ga_return_type

  Graph analyzer return type.
- typedef iderf_ret_type(∗ iderf_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)
- typedef infbound(∗ infbound_evaluator)(const std::vector< infbound > ∗__x, const variable_indicator &__v)
- typedef interval(∗ interval_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)
- typedef boost::numeric::interval_lib::policies < my_rounded_math< double > , checking_my< double >> my_policies
- typedef Islope(∗ Islope_evaluator)(const std::vector< Islope > ∗__x, const variable_indicator &__v)
- typedef std::map< int, locopt_ret_record > locopt_ret
- typedef model::walker expression_walker

  Walker to the expression DAG.
- typedef model::const_walker expression_const_walker

  Const walker to the expression DAG.
- typedef xxxNumber_t(∗ xxxNumber_evaluator)(const std::vector< xxxNumber_t > ∗__x, const variable_indicator &__v)
- typedef search_graph::const_walker search_inspector

  The search inspector for graph analysis.
- typedef search_graph::walker search_focus

  The search focus for work node selection.
- typedef std::pair< vdbl::rowid, search_node_id > si_ri_pair
- typedef std::pair< unsigned int, unsigned int > ipair

  Row-column index pair.
- typedef std::set< ipair > splist

  Sparsity information.
- typedef std::triple< unsigned int, unsigned int, unsigned int > itriple

  Row-column-depth index triple.
- typedef std::set< itriple > stlist

  Sparsity information.

- typedef bool(∗ prep_d_evaluator)()
- typedef double(∗ func_d_evaluator)(const std::vector< double > ∗__x, const variable_indicator &__v, std::vector< double > &__d_data)
- typedef std::vector< double > &(∗ der_evaluator)(const std::vector< double > &__d_dat, const variable_indicator &__v)

- typedef bool(∗ prep_id_evaluator)()
- typedef interval(∗ func_id_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v, std::vector< interval > &__id_data)
- typedef std::vector< interval > &(∗ ider_evaluator)(const std::vector< interval > &__d_dat, const variable_indicator &__v)

- typedef bool(∗ prep_islp_evaluator)()
- typedef func_islp_return_type(∗ func_islp_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v, std::vector< interval > &__islp_data)
- typedef std::vector< interval > &(∗ islp_evaluator)(const std::vector< interval > &__d_dat, const variable_indicator &__v)

Enumerations

- enum api_exception_type { apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14 }

  Enum for classifying api_exceptions.
- enum tristate_e { t_true = 1, t_false = -1, t_maybe = 0 }
- enum convex_info { c_convex = 1, c_linear = 0, c_concave = -1, c_maybe = 2, c_not = -2, c_constant = 3 }

  Convexity information enum.
- enum d1func_monotonicity_t { d1fpmon_dec = -1, d1fpmon_const = 0, d1fpmon_inc = 1, d1fpmon_unclear = 2, d1fpmon_nonmon = -2 }

  The d1func_monotonicity_t enum.
- enum dbc_method { dbc_cmp_eq = DB_COMPARE_CMP_EQ, dbc_cmp_ne = DB_COMPARE_CMP_NE, dbc_cmp_gt = DB_COMPARE_CMP_GT, dbc_cmp_ge = DB_COMPARE_CMP_GE, dbc_cmp_lt = DB_COMPARE_CMP_LT, dbc_cmp_le = DB_COMPARE_CMP_LE, dbc_cmp_in = DB_COMPARE_CMP_IN, dbc_cmp_abseq = DB_COMPARE_CMP_ABSEQ, dbc_cmp_absne = DB_COMPARE_CMP_ABSNE, dbc_cmp_absgt = DB_COMPARE_CMP_ABSGT, dbc_cmp_absge = DB_COMPARE_CMP_ABSGE, dbc_cmp_abslt = DB_COMPARE_CMP_ABSLT, dbc_cmp_absle = DB_COMPARE_CMP_ABSLE, dbc_cmp_absin = DB_COMPARE_CMP_ABSIN, dbc_sort_min = DB_COMPARE_SORT_MIN, dbc_sort_max = DB_COMPARE_SORT_MAX, dbc_sort_absmin = DB_COMPARE_SORT_ABSMIN, dbc_sort_absmax = DB_COMPARE_SORT_ABSMAX, dbc_sort_true = DB_COMPARE_SORT_TRUE, dbc_sort_false = DB_COMPARE_SORT_FALSE }
- enum e_expression_type { ex_bound = 1, ex_linear = 1<<1, ex_quadratic = 1<<2, ex_polynomial = 1<<3, ex_other = 1<<4, ex_kj = 1<<7, ex_org = 1<<8, ex_redundant = 1<<9, ex_notredundant = 1<<10, ex_active_lo = 1<<11, ex_inactive_lo = 1<<12, ex_active_hi = 1<<13, ex_inactive_hi = 1<<14, ex_active = ex_active_lo|ex_active_hi, ex_inactive = ex_inactive_lo|ex_inactive_hi, ex_integer = 1<<15, ex_exists = 1<<16, ex_forall = 1<<17, ex_free = 1<<18, ex_stochastic = 1<<19, ex_convex = 1<<20, ex_concave = 1<<21, ex_inequality = 1<<28, ex_equality = 1<<29, ex_leftbound = 1<<30, ex_rightbound = 1<<31, ex_atmlin = ex_bound|ex_linear, ex_atmquad = ex_atmlin|ex_quadratic, ex_atmpoly = ex_atmquad|ex_polynomial, ex_nonlin = ex_quadratic|ex_polynomial|ex_other, ex_nonbnd = ex_linear|ex_nonlin, ex_any = ex_atmlin|ex_nonlin, ex_bothbound = ex_leftbound|ex_rightbound }

- enum search_node_relation { snr_root, snr_reduction, snr_relaxation, snr_split, snr_glue, snr_worknode, snr_virtual }

  Enum specifying node relations in the search graph.
- enum type_annotation { v_exists = 0, v_forall = 1, v_free = 2, v_stochastic = 3 }

  Node type information enum.
- enum activity_descr { a_redundant = 1, a_active_lo = 2, a_active_lo_red = a_active_lo|a_redundant, a_active_hi = 4, a_active_hi_red = a_active_hi|a_redundant, a_active = a_active_lo|a_active_hi, a_active_red = a_active|a_redundant }

  Constraint activity information enum.
- enum sp_indexing { full_indexing = 0, row_wise_indexing = 1, column_wise_indexing = 2, full_indexing = 0, row_wise_indexing = 1, column_wise_indexing = 2 }
- enum sp_indexing { full_indexing = 0, row_wise_indexing = 1, column_wise_indexing = 2, full_indexing = 0, row_wise_indexing = 1, column_wise_indexing = 2 }

Functions

- std::ostream & operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.
- std::ostream & operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.
- interval flb (const interval &x1, const interval &x2, const interval &x3)
- interval asqr_find_f1min (const interval &c1, const interval &c3, const interval &x2)
- interval eval_asqr (const interval &x1, const interval &x2, const interval &x3)
- const basic_alltype & basic_alltype_empty ()
- std::ostream & operator<< (std::ostream &os, const basic_alltype &b)
- bool less_than (const basic_alltype &a, const basic_alltype &b)
- bool less_equal (const basic_alltype &a, const basic_alltype &b)
- double d_random ()
- double r_random_h_eval (double t, double b, double a)
- double r_random_hinv_eval (double x, double b, double a)
- double r_random (double l, double u, double beta=10.0, double alpha=0.9)
- double r_random (const interval &_i, double beta=10.0, double alpha=0.9)
- interval i_random (double l, double r, double beta=10.0, double alpha=0.9)
- interval i_random (const interval &_i, double beta=10.0, double alpha=0.9)
- interval i_random ()
- bool operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.
- bool operator== (const convex_e &__c, const convex_info &__d)
- bool operator== (const convex_info &__c, const convex_e &__d)
- bool operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.
- bool operator!= (const convex_e &__c, const convex_info &__d)
- bool operator!= (const convex_info &__c, const convex_e &__d)
- std::ostream & operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.
- d1func_monotonicity_t operator- (d1func_monotonicity_t x)

  The d1func_monotonicity_t unary minus.

- interval _internal_imperfect_eval (const interval &x, unsigned int k, const expression_const_walker &result, const model *mod, const variable_indicator &vind, const std::map< unsigned int, d1func * > *p_infos)
- interval_set intersect_inv_hlp (d1func *d1help, const interval &xx, const interval_set &r, int order)
- template<typename _TR >
  bool eval_in (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, _TR, const basic_alltype &I)
- template<>
  bool eval_in< interval > (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, interval, const basic_alltype &I)
- template<>
  bool eval_in< std::string > (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, std::string, const basic_alltype &I)
- template<typename _TR >
  bool eval_s (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, _TR cmpval)
- template<>
  bool eval_s< interval > (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, interval cmpval)
- template<typename _TR >
  bool eval_u (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, _TR cmpval)
- bool eval (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, const basic_alltype &cmpval)
- void parse_dbcompare_expression (const std::string &s, std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, std::vector< std::vector< int > > &expr)
- void parse_dbcompare_sortorder (const std::string &s, std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, std::vector< int > &order)
- template<typename _AT >
  void filter_from_view (typename std::vector< std::pair< vdbl::rowid, _AT > > &riv, vdbl::view *iv, const std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, const std::vector< std::vector< int > > &expr)
- template<typename _AT >
  void list_from_view (typename std::vector< std::pair< vdbl::rowid, _AT > > &riv, vdbl::view *iv, const std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, const std::vector< int > &sort, unsigned int n)
- template<typename _TR >
  _TR dbccmps_abs (const _TR &x)
- template<typename _TR , typename _AT >
  std::vector< std::pair < vdbl::rowid, _AT > >::iterator filter_s (typename std::vector< std::pair< vdbl::rowid, _AT > > &rid, typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator rid_b, const std::string &colname, dbc_method tp, vdbl::view *iv, unsigned int n, _TR, double rel, double abs)
- template<typename _TR , typename _AT >
  std::vector< std::pair < vdbl::rowid, _AT > >::iterator filter_u (typename std::vector< std::pair< vdbl::rowid, _AT > > &rid, typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator rid_b, const std::string &colname, dbc_method tp, vdbl::view *iv, unsigned int n, _TR)
- template<typename _AT >
  std::vector< std::pair < vdbl::rowid, _AT > >::iterator filter (typename std::vector< std::pair< vdbl::rowid, _AT > > &rid, typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator rid_b, const std::string &colname, dbc_method tp, vdbl::view *iv, unsigned int n, std::pair< double, double > rel_abs)

- tableid get_point_table (database *ptr, standard_table *&ptb, userid _uid)
- vdbl::tableid get_point_table (vdbl::database *ptr, vdbl::standard_table *&ptb, vdbl::userid)

  Get the point from the search database.
- std::string get_diff_class (int i)
- diffI operator+ (const diffI &a, const diffI &b)
- diffI operator+ (const double &a, const diffI &b)
- diffI operator+ (const diffI &a, const double &b)
- diffI operator+ (const interval &a, const diffI &b)
- diffI operator+ (const diffI &a, const interval &b)
- diffI operator- (const diffI &a, const diffI &b)
- diffI operator- (const double &a, const diffI &b)
- diffI operator- (const diffI &a, const double &b)
- diffI operator- (const interval &a, const diffI &b)
- diffI operator- (const diffI &a, const interval &b)
- diffI operator- (const diffI &a)
- diffI operator* (const double &a, const diffI &b)
- diffI operator* (const diffI &a, const double &b)
- diffI operator/ (const diffI &a, const double &b)
- diffI operator* (const interval &a, const diffI &b)
- diffI operator* (const diffI &a, const interval &b)
- diffI operator/ (const diffI &a, const interval &b)
- diffI operator* (const diffI &a, const diffI &b)
- diffI operator/ (const diffI &a, const diffI &b)
- diffI operator/ (const double &a, const diffI &b)
- diffI operator/ (const interval &a, const diffI &b)
- diffI exp (const diffI &a)
- diffI sin (const diffI &a)
- diffI cos (const diffI &a)
- diffI sqrt (const diffI &a)
- diffI abs (const diffI &a)
- diffI power (const diffI &a, int n)
- diffI pow (const diffI &a, const diffI &b)
- diffI max (const diffI &a, const diffI &b)
- diffI min (const diffI &a, const diffI &b)
- diffI log (const diffI &a)
- diffI sinh (const diffI &a)
- diffI cosh (const diffI &a)
- diffI atan2 (const diffI &a, const diffI &b)
- diffI atan (const diffI &a)
- diffI sqr (const diffI &a)
- std::ostream & operator<< (std::ostream &s, const diffI &a)
- diffNumber operator* (const diffNumber &a, const diffNumber &b)
- diffNumber operator/ (const diffNumber &a, const diffNumber &b)
- diffNumber operator/ (const double &a, const diffNumber &b)
- diffNumber exp (const diffNumber &a)
- diffNumber sin (const diffNumber &a)
- diffNumber cos (const diffNumber &a)
- diffNumber sqrt (const diffNumber &a)
- diffNumber abs (const diffNumber &a)
- diffNumber max (const diffNumber &a, const diffNumber &b)

- diffNumber min (const diffNumber &a, const diffNumber &b)
- diffNumber power (const diffNumber &a, int n)
- diffNumber pow (const diffNumber &a, const diffNumber &b)
- diffNumber log (const diffNumber &a)
- diffNumber sinh (const diffNumber &a)
- diffNumber cosh (const diffNumber &a)
- diffNumber atan2 (const diffNumber &a, const diffNumber &b)
- diffNumber atan (const diffNumber &a)
- diffNumber square (const diffNumber &a)
- std::ostream & operator<< (std::ostream &s, const diffNumber &a)
- diffNumber operator+ (const diffNumber &a, const diffNumber &b)
- diffNumber operator+ (const double &a, const diffNumber &b)
- diffNumber operator+ (const diffNumber &a, const double &b)
- diffNumber operator- (const diffNumber &a, const diffNumber &b)
- diffNumber operator- (const double &a, const diffNumber &b)
- diffNumber operator- (const diffNumber &a, const double &b)
- diffNumber operator- (const diffNumber &a)
- diffNumber operator∗ (const double &a, const diffNumber &b)
- diffNumber operator∗ (const diffNumber &a, const double &b)
- diffNumber operator/ (const diffNumber &a, const double &b)
- std::vector< std::vector< void ∗ > > standard_evaluators (NUM_EVALUATORS)
- template<class _Walker , class _Visitor >
  _Visitor::return_value recursive_short_cut_walk (_Walker __w, _Visitor __f)

  Perform a recursive graph walk with possible caching and short-cuts.
- template<class _Walker , class _Visitor >
  _Visitor::return_value _recursive_short_cut_walk (_Walker __w, _Visitor __f)

  Perform a recursive graph walk with possible caching and short-cuts (internal)
- template<class _Visitor , class _Walker >
  _Visitor::return_value evaluate (_Visitor __v, _Walker __start)

  Evaluate an evaluator on a DAG.
- std::ostream & operator<< (std::ostream &o, const expression_node &__x)
- std::ostream & __wr_interval (std::ostream &o, const interval &__i)
- void _internal_graphorder (const model &DAG, std::vector< unsigned int > &order, std::vector< unsigned int > ∗inv_order)
- void graphorder (const model &DAG, std::vector< unsigned int > &order)

  Compute a graph order.
- void graphorder (const model &DAG, std::vector< unsigned int > &order, std::vector< unsigned int > &inv_order)

  Compute a graph order.
- void hess_star_color1 (const unsigned int n, const vmtl::sparse_matrix< int > &H, std::vector< std::vector< unsigned int > > &parts, std::vector< vmtl::sparse_vector< unsigned int > > &provides)
- hessNumber operator∗ (const hessNumber &a, const hessNumber &b)
- hessNumber operator/ (const hessNumber &a, const hessNumber &b)
- hessNumber operator/ (const double &a, const hessNumber &b)
- hessNumber exp (const hessNumber &a)
- hessNumber sin (const hessNumber &a)
- hessNumber cos (const hessNumber &a)
- hessNumber sqrt (const hessNumber &a)
- hessNumber abs (const hessNumber &a)
- hessNumber power (const hessNumber &a, int n)

- hessNumber pow (const hessNumber &a, const hessNumber &b)
- hessNumber log (const hessNumber &a)
- hessNumber sinh (const hessNumber &a)
- hessNumber cosh (const hessNumber &a)
- hessNumber atan2 (const hessNumber &a, const hessNumber &b)
- hessNumber atan (const hessNumber &a)
- hessNumber square (const hessNumber &a)
- std::ostream & operator<< (std::ostream &s, const hessNumber &a)
- hessNumber operator+ (const hessNumber &a, const hessNumber &b)
- hessNumber operator+ (const double &a, const hessNumber &b)
- hessNumber operator+ (const hessNumber &a, const double &b)
- hessNumber operator- (const hessNumber &a, const hessNumber &b)
- hessNumber operator- (const double &a, const hessNumber &b)
- hessNumber operator- (const hessNumber &a, const double &b)
- hessNumber operator- (const hessNumber &a)
- hessNumber operator∗ (const double &a, const hessNumber &b)
- hessNumber operator∗ (const hessNumber &a, const double &b)
- hessNumber operator/ (const hessNumber &a, const double &b)
- hessNumber max (const hessNumber &a, const hessNumber &b)
- hessNumber min (const hessNumber &a, const hessNumber &b)
- ihessNumber operator∗ (const ihessNumber &a, const ihessNumber &b)
- ihessNumber operator/ (const ihessNumber &a, const ihessNumber &b)
- ihessNumber operator/ (const double &a, const ihessNumber &b)
- ihessNumber operator/ (const interval &a, const ihessNumber &b)
- ihessNumber exp (const ihessNumber &a)
- ihessNumber sin (const ihessNumber &a)
- ihessNumber cos (const ihessNumber &a)
- ihessNumber sqrt (const ihessNumber &a)
- ihessNumber abs (const ihessNumber &a)
- ihessNumber power (const ihessNumber &a, int n)
- ihessNumber pow (const ihessNumber &a, const ihessNumber &b)
- ihessNumber max (const ihessNumber &a, const ihessNumber &b)
- ihessNumber min (const ihessNumber &a, const ihessNumber &b)
- ihessNumber log (const ihessNumber &a)
- ihessNumber sinh (const ihessNumber &a)
- ihessNumber cosh (const ihessNumber &a)
- ihessNumber atan2 (const ihessNumber &a, const ihessNumber &b)
- ihessNumber atan (const ihessNumber &a)
- ihessNumber sqr (const ihessNumber &a)
- std::ostream & operator<< (std::ostream &s, const ihessNumber &a)
- ihessNumber operator+ (const ihessNumber &a, const ihessNumber &b)
- ihessNumber operator+ (const double &a, const ihessNumber &b)
- ihessNumber operator+ (const interval &a, const ihessNumber &b)
- ihessNumber operator+ (const ihessNumber &a, const double &b)
- ihessNumber operator+ (const ihessNumber &a, const interval &b)
- ihessNumber operator- (const ihessNumber &a, const ihessNumber &b)
- ihessNumber operator- (const double &a, const ihessNumber &b)
- ihessNumber operator- (const interval &a, const ihessNumber &b)
- ihessNumber operator- (const ihessNumber &a, const double &b)
- ihessNumber operator- (const ihessNumber &a, const interval &b)

- ihessNumber operator- (const ihessNumber &a)
- ihessNumber operator∗ (const double &a, const ihessNumber &b)
- ihessNumber operator∗ (const interval &a, const ihessNumber &b)
- ihessNumber operator∗ (const ihessNumber &a, const double &b)
- ihessNumber operator∗ (const ihessNumber &a, const interval &b)
- ihessNumber operator/ (const ihessNumber &a, const double &b)
- ihessNumber operator/ (const ihessNumber &a, const interval &b)
- double safeguarded_mid (const interval &__i)
- interval ipow (const interval &x, int n)
- interval gauss (const interval &x)
- interval atan2 (const interval &y, const interval &x)
- double absmin (const interval &__i)
- double gainfactor (const interval &_old, const interval &_new)
- bool operator== (const interval &a, const interval &b)
- bool operator== (const interval &a, double b)
- bool operator!= (const interval &a, const interval &b)
- bool operator!= (const interval &a, double b)
- bool operator< (const interval &a, const interval &b)
- interval operator+ (const interval &a, const interval &b)
- interval operator- (const interval &a, const interval &b)
- interval cancel (const interval &a, const interval &b)
- interval operator∗ (const interval &a, const interval &b)
- interval operator/ (const interval &a, const interval &b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- double mid (const interval &)
- double diam (const interval &)
- double width (const interval &)
- double relDiam (const interval &)
- double rad (const interval &)
- double mig (const interval &)
- double mag (const interval &)
- double dist (const interval &x, const interval &y)
- interval round_to_integer (const interval &x)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)

- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- double downward_plus (const double &a, const double &b)
- double upward_plus (const double &a, const double &b)
- double downward_minus (const double &a, const double &b)
- double upward_minus (const double &a, const double &b)
- double downward_multiplies (const double &a, const double &b)
- double upward_multiplies (const double &a, const double &b)
- double downward_divides (const double &a, const double &b)
- double upward_divides (const double &a, const double &b)
- interval intersect (const interval &__x, const interval &__y)
- interval hull (const interval &__x, const interval &__y)
- interval hulldiff (const interval &__x, const interval &__y)
- double width (const interval_set &a)
- double volume (const interval_set &a)
- interval_set divide (const interval &a, const interval &b)
- bool operator== (const interval_set &a, const interval_set &b)
- interval_set imin (const interval_set &a, const interval_set &b)
- interval_set imax (const interval_set &a, const interval_set &b)
- interval_set power (const interval_set &a, int n)
- interval_set sqr (const interval_set &a)
- interval_set sqrt (const interval_set &a)
- interval_set exp (const interval_set &a)
- interval_set exp10 (const interval_set &a)
- interval_set exp2 (const interval_set &a)
- interval_set expm1 (const interval_set &a)
- interval_set log (const interval_set &a)
- interval_set log10 (const interval_set &a)
- interval_set log1p (const interval_set &a)
- interval_set log2 (const interval_set &a)
- interval_set abs (const interval_set &a)
- interval_set sin (const interval_set &a)
- interval_set cos (const interval_set &a)
- interval_set cot (const interval_set &a)
- interval_set tan (const interval_set &a)
- interval_set asin (const interval_set &a)
- interval_set acos (const interval_set &a)

- interval_set acot (const interval_set &a)
- interval_set atan (const interval_set &a)
- interval_set sinh (const interval_set &a)
- interval_set cosh (const interval_set &a)
- interval_set coth (const interval_set &a)
- interval_set tanh (const interval_set &a)
- interval_set asinh (const interval_set &a)
- interval_set acosh (const interval_set &a)
- interval_set acoth (const interval_set &a)
- interval_set atanh (const interval_set &a)
- interval_set pow (const interval_set &a, const interval_set &b)
- interval_set atan2 (const interval_set &a, const interval_set &b)
- interval_set round_to_integer (const interval_set &a)
- interval_set operator+ (const interval_set &a, const interval_set &b)
- interval_set operator+ (const interval_set &a, const interval &b)
- interval_set operator+ (const interval &a, const interval_set &b)
- interval_set operator+ (const interval_set &a, double b)
- interval_set operator+ (double a, const interval_set &b)
- interval_set operator- (const interval_set &a, const interval_set &b)
- interval_set operator- (const interval_set &a, const interval &b)
- interval_set operator- (const interval &a, const interval_set &b)
- interval_set operator- (const interval_set &a, double b)
- interval_set operator- (double a, const interval_set &b)
- interval_set operator- (const interval_set &a)
- interval_set operator∗ (const interval_set &a, const interval_set &b)
- interval_set operator∗ (const interval_set &a, const interval &b)
- interval_set operator∗ (const interval &a, const interval_set &b)
- interval_set operator∗ (const interval_set &a, double b)
- interval_set operator∗ (double a, const interval_set &b)
- interval_set divide (const interval &a, double b)
- interval_set divide (double a, const interval &b)
- interval_set operator/ (const interval_set &a, const interval_set &b)
- interval_set operator/ (const interval_set &a, const interval &b)
- interval_set operator/ (const interval &a, const interval_set &b)
- interval_set operator/ (const interval_set &a, double b)
- interval_set operator/ (double a, const interval_set &b)
- bool operator!= (const interval_set &a, const interval_set &b)
- std::ostream & operator<< (std::ostream &o, const func_islp2_eval_ret_type &r)
- double get_locopt_weight_from_rval (const locopt_ret &ret_info, const std::string &iename, const int srval)
- termination_reason get_locopt_termr_from_rval (const locopt_ret &ret_info, const std::string &iename, const int srval)
- bool mat_inv (const std::vector< std::vector< double > > &A0, std::vector< std::vector< double > > &R)
- bool Newton_Step (const std::vector< std::vector< interval > > &J, const int N, std::vector< interval > x, const std::vector< interval > &c, const std::vector< interval > &fc, std::vector< interval > &xout, bool precond)
- bool Newton_Step (const std::vector< std::vector< interval > > &J, const int N, std::vector< interval > x, const std::vector< interval > &c, const std::vector< interval > &fc, std::vector< std::vector< interval > > &splits, bool precond)

- bool Newton_Step_1D (const interval &J, interval x, const interval &c, const interval &fc, interval &xout, bool precond)
- bool Newton_Step_1D (const interval &J, interval x, const interval &c, const interval &fc, std::vector< interval > &splits, bool precond)
- bool operator< (int n, const proj_rational &R)
- bool operator<= (int n, const proj_rational &R)
- bool operator> (int n, const proj_rational &R)
- bool operator>= (int n, const proj_rational &R)
- bool operator== (int n, const proj_rational &R)
- bool operator!= (int n, const proj_rational &R)
- proj_rational abs (const proj_rational &r)
- template<class I >
  I pow (const I &i, const proj_rational &r)
- template<class I >
  projective_interval< I > operator+ (const projective_interval< I > &a, const projective_interval< I > &b)
- template<class I >
  projective_interval< I > operator+ (const projective_interval< I > &a, double b)
- template<class I >
  projective_interval< I > operator+ (double b, const projective_interval< I > &a)
- template<class I >
  projective_interval< I > operator- (const projective_interval< I > &a, const projective_interval< I > &b)
- template<class I >
  projective_interval< I > operator- (const projective_interval< I > &a, double b)
- template<class I >
  projective_interval< I > operator- (double b, const projective_interval< I > &a)
- template<class I >
  projective_interval< I > operator* (const projective_interval< I > &a, const projective_interval< I > &b)
- template<class I >
  projective_interval< I > operator* (const projective_interval< I > &a, double b)
- template<class I >
  projective_interval< I > operator* (double b, const projective_interval< I > &a)
- template<class I >
  projective_interval< I > operator/ (const projective_interval< I > &a, const projective_interval< I > &b)
- template<class I >
  projective_interval< I > operator/ (const projective_interval< I > &a, double b)
- template<class I >
  projective_interval< I > operator/ (double b, const projective_interval< I > &a)
- template<class I >
  projective_interval< I > acos (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > abs (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > acosh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > acot (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > acoth (const projective_interval< I > &x)

- template<class I >
  projective_interval< I > asin (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > asinh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > atan (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > atanh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > cos (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > cosh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > cot (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coth (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > exp (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > exp10 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > exp2 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > expm1 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > log (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > log10 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > log1p (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > log2 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > power (const projective_interval< I > &x, int n)
- template<class I >
  projective_interval< I > pow (const projective_interval< I > &x, const projective_interval< I > &y)
- template<class I >
  projective_interval< I > sin (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > sinh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > sqr (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > sqrt (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > tan (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > tanh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > imax (const projective_interval< I > &x, const projective_interval< I > &y)

- template<class I >
  projective_interval< I > imin (const projective_interval< I > &x, const projective_interval< I > &y)
- template<class I >
  projective_interval< I > atan2 (const projective_interval< I > &x, const projective_interval< I > &y)
- template<class I >
  projective_interval< I > division_part1 (const projective_interval< I > &x, const projective_interval< I > &y, bool &b)
- template<class I >
  projective_interval< I > division_part2 (const projective_interval< I > &x, const projective_interval< I > &y, bool b)
- work_node full_node_to_work_node (full_node &n_full, gptr< search_node > &ground)

  This function converts a full_node to a work_node.
- void get_ids (const std::vector< si_ri_pair > &riv, std::vector< search_node_id > &v)
- void get_leaves_vector (const search_graph &__sgraph, std::vector< si_ri_pair > &v, search_inspector *sfoc=NULL)

  Sparsity calculation (Hessian and Jacobian)
- bool get_children_vector (const search_graph &__sgraph, const search_node_id &pid, std::vector< si_ri_pair > &v, search_inspector *sfoc)
- work_node operator- (const work_node &_w, const delta_id &_d)

  Remove one delta from a work node.
- work_node & operator-= (work_node &_w, const delta_id &_d)

  Remove one delta from a work node.
- work_node operator+ (const work_node &_w, const delta_id &_i)

  Add one delta to a work node.
- work_node & operator+= (work_node &_w, const delta_id &_i)

  Add one delta to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node operator+ (const work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Add a number of deltas to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node & operator+= (work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Add a number of deltas to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node operator- (const work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Remove a number of deltas from a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node & operator-= (work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Remove a number of deltas from a work node.
- std::ostream & operator<< (std::ostream &o, const semantics &__s)

  C++ stream output operator for semantics.
- void sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

Sparsity calculation (Hessian)

- void sparsity (const model &DAG, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)
- void sparsity (const model &DAG, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void sparsity3 (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void sparsity3 (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void sparsity3 (const model &DAG, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void set_translation_map (const std::map< std::string, std::string > &tr_msg_map)

  A datamap function for setting the message translation map.
- const std::string & get_translated_message (const std::string &msg)

  A datamap function for message translation.
- std::ostream & operator<< (std::ostream &o, const termination_reason &__x)

  C++ stream output operator for the termination_reason.
- std::string i2a (int x)

  converter int -> string
- std::string datetime (const struct tm *timeptr)

  local time as C++ string
- std::string localdatetime ()

  converter date & time into std::string
- std::string e2D (double d)

  Fortran real as C++ string.

- template<class _TC >
  bool operator== (const interval &__i, const _TC &__d)

- template<class _TC >
  bool operator!= (const interval &__i, const _TC &__d)

- bool operator< (const interval &a, double b)
- bool operator< (double a, const interval &b)

- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)

- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)

- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)

- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)

- interval division_part2 (const interval &x, const interval &y, bool b=true)

- bool operator> (const interval &a, const interval &b)
- bool operator> (const interval &a, double b)
- bool operator> (double a, const interval &b)

- bool operator<= (const interval &a, const interval &b)
- bool operator<= (const interval &a, double b)
- bool operator<= (double a, const interval &b)

- bool operator>= (const interval &a, const interval &b)
- bool operator>= (const interval &a, double b)
- bool operator>= (double a, const interval &b)

- bool lexicographic_lt (const interval &a, const interval &b)

- bool lexicographic_le (const interval &a, const interval &b)

- void sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

- void sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)

- void sparsity (const model &DAG, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

- void sparsity3 (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void sparsity3 (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)

- void sparsity3 (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void sparsity3 (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)

- void sparsity3 (const model &DAG, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void sparsity3 (const model &DAG, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)

Variables

- const char * expr_names [ ]
- const int STANDARD_MAXDEG_DIFFINTERVAL = 4
- const int STANDARD_MAXDEG_DIFFNUMBER = 4
- std::vector< std::vector< void * > > standard_evaluators
- const char * correxpr_names [ ] = {"log10", "asin", "acos", "atan", "tan", "<=", "<", ">=", ">", "==", "!="}

type for structure info summaries

- enum func_type { func_const = 0, func_lin = 1, func_quad = 2, func_poly = 3, func_nonpoly = 4 }
- enum fset_type { fset_unbd = 0, fset_box = 1, fset_lin = 2, fset_iquad = 3, fset_equad = 4, fset_ipoly = 5, fset_epoly = 6, fset_inonpoly = 7, fset_enonpoly = 8 }
- typedef enum coco::func_type func_type_t
- typedef enum coco::fset_type fset_type_t

9.1.1 Detailed Description

This is the main namespace holding all classes and functions of the COCONUT-Environment API.

9.1.2 Typedef Documentation

9.1.2.1 typedef analyticd(* coco::analyticd_evaluator)(const std::vector< analyticd > *__x, const variable_indicator &__v)

Definition at line 46 of file ade_evaluator.h.

9.1.2.2 typedef b_interval(* coco::b_interval_evaluator)(const std::vector< b_interval > *__x, const variable_indicator &__v)

Definition at line 46 of file bint_evaluator.h.

9.1.2.3 typedef cinterval(* coco::cinterval_evaluator)(const std::vector< cinterval > *__x, const variable_indicator &__v)

Definition at line 46 of file cint_evaluator.h.

9.1.2.4 typedef interval(∗ coco::dd1f_interval_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)

Functions of this type need to be defined for all user defined nodes.

Definition at line 48 of file dagd1func_ie.h.

9.1.2.5 typedef defdom_ret_type(∗ coco::defdom_evaluator)(const std::vector< interval > ∗__x, defdom_map &__m, const variable_indicator &__v)

Functions of this type need to be defined for all user defined nodes.

Definition at line 78 of file defdom_evaluator.h.

9.1.2.6 typedef std::multimap<unsigned int,defdom_problem_point> coco::defdom_map

Definition at line 75 of file defdom_evaluator.h.

9.1.2.7 typedef std::vector<double>&(∗ coco::der_evaluator)(const std::vector< double > &__d_dat, const variable_indicator &__v)

Functions of these types need to be defined for all user defined nodes.

Definition at line 53 of file der_evaluator.h.

9.1.2.8 typedef std::vector<void∗> coco::evaluator_v

This type defines the type of the standard evaluators for user defined nodes.

Definition at line 462 of file expression.h.

9.1.2.9 typedef enum coco::fset_type coco::fset_type_t

9.1.2.10 typedef double(∗ coco::func_d_evaluator)(const std::vector< double > ∗__x, const variable_indicator &__v, std::vector< double > &__d_data)

Functions of these types need to be defined for all user defined nodes.

Definition at line 50 of file der_evaluator.h.

9.1.2.11 typedef double(∗ coco::coco::func_evaluator)(const std::vector< double > ∗__x, const variable_indicator &__v)

Functions of this type need to be defined for all user defined nodes.

Definition at line 45 of file dfunc_evaluator.h.

9.1.2.12 typedef interval(∗ coco::func_id_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v, std::vector< interval > &__id_data)

Functions of these types need to be defined for all user defined nodes.

Definition at line 51 of file ider_evaluator.h.

9.1.2.13 typedef func_islp_return_type(∗ coco::func_islp_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v, std::vector< interval > &__islp_data)

Functions of these types need to be defined for all user defined nodes.

Definition at line 63 of file islp_evaluator.h.

9.1.2.14 typedef enum coco::func_type coco::func_type_t

9.1.2.15 typedef std::vector<interval>&(∗ coco::ider_evaluator)(const std::vector< interval > &__d_dat, const variable_indicator &__v)

Functions of these types need to be defined for all user defined nodes.

Definition at line 54 of file ider_evaluator.h.

9.1.2.16 typedef iderf_ret_type(∗ coco::iderf_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)

Functions of this type need to be defined for all user defined nodes.

Definition at line 56 of file iderf_evaluator.h.

9.1.2.17 typedef infbound(∗ coco::infbound_evaluator)(const std::vector< infbound > ∗__x, const variable_indicator &__v)

Definition at line 46 of file infb_evaluator.h.

9.1.2.18 typedef interval(∗ coco::interval_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)

Functions of this type need to be defined for all user defined nodes.

Definition at line 48 of file int_evaluator.h.

9.1.2.19 typedef Islope(∗ coco::Islope_evaluator)(const std::vector< Islope > ∗__x, const variable_indicator &__v)

Functions of this type need to be defined for all user defined nodes.

Definition at line 46 of file Islope_evaluator.h.

9.1.2.20 typedef std::vector<interval>&(∗ coco::islp_evaluator)(const std::vector< interval > &__d_dat, const variable_indicator &__v)

Functions of these types need to be defined for all user defined nodes.

Definition at line 66 of file islp_evaluator.h.

9.1.2.21 typedef std::map<int,locopt_ret_record> coco::locopt_ret

Definition at line 63 of file locopt_ret.h.

9.1.2.22 typedef boost::numeric::interval_lib::policies< my_rounded_math<double>, checking_my<double>> coco::my_policies

Definition at line 100 of file interval_boost.h.

9.1.2.23 typedef polsppt_ret_type(* coco::polsppt_evaluator)(const std::vector< interval > *__x, const variable_indicator &__v)

Functions of this type need to be defined for all user defined nodes.

Definition at line 57 of file dagd1func_pv.hpp.

9.1.2.24 typedef std::triple<d1func::d1func_point_t,diffI,int> coco::polsppt_info

Definition at line 60 of file dagd1func_pv.hpp.

9.1.2.25 typedef std::multimap<unsigned int, polsppt_info> coco::polsppt_info_map

Definition at line 62 of file dagd1func_pv.hpp.

9.1.2.26 typedef std::map<interval,polsppt_info_map,compare_intervals> coco::polsppt_interval_map

Definition at line 64 of file dagd1func_pv.hpp.

9.1.2.27 typedef std::multimap<unsigned int, polsppt_info_map::const_iterator> coco::polsppt_map_map

Definition at line 67 of file dagd1func_pv.hpp.

9.1.2.28 typedef bool(* coco::prep_d_evaluator)()

Functions of these types need to be defined for all user defined nodes.

Definition at line 49 of file der_evaluator.h.

9.1.2.29 typedef bool(* coco::prep_id_evaluator)()

Functions of these types need to be defined for all user defined nodes.

Definition at line 50 of file ider_evaluator.h.

9.1.2.30 typedef bool(* coco::prep_islp_evaluator)()

Functions of these types need to be defined for all user defined nodes.

Definition at line 62 of file islp_evaluator.h.

9.1.2.31 typedef interval coco::rhs_t

This type is the allowed type for constraint restrictions.

Definition at line 459 of file expression.h.

9.1.2.32 typedef std::pair<vdbl::rowid, search_node_id> coco::si_ri_pair

Definition at line 35 of file search_graph_funcs.h.

9.1.2.33 typedef enum coco::coco::tristate_e coco::coco::tristate

The tristate type is a boolean with the addition of a maybe value

9.1.2.34 typedef std::vector<std::pair<vdbl::tableid,vdbl::rowid> > coco::trvec

Definition at line 33 of file annotation_delta.cc.

9.1.2.35 typedef xxxNumber_t(* coco::xxxNumber_evaluator)(const std::vector< xxxNumber_t > *__x, const variable_indicator &__v)

Functions of this type need to be defined for all user defined nodes.

Definition at line 62 of file numfeval.h.

9.1.3 Enumeration Type Documentation

9.1.3.1 enum coco::activity_descr

This enum describes the activity state known about a constraint and whether it is redundant or not. The meaning of the enum entries is:

- `a_redundant`: The constraint is known to be redundant. There can be two reasons for that. It can be inactive on both sides, or it was constructed, knowing that it would be redundant (e.g. cuts).
- `a_active_lo`: The `work_node` may contain points, for which this constraint is active at the lower bound but no points, for which the constraint is active at the upper bound.
- `a_active_hi`: The `work_node` may contain points, for which this constraint is active at the upper bound but no points, for which the constraint is active at the lower bound.
- `a_active`: The `work_node` may contain points, for which this constraint is active at the upper bound and points, for which the constraint is active at the lower bound.
- ..._red: A combination with `_red` means that although the inactivity of the constraint could not be proved, it is still known to be redundant.

Enumerator:

a_redundant
a_active_lo
a_active_lo_red
a_active_hi
a_active_hi_red
a_active
a_active_red

Definition at line 81 of file semantics.h.

9.1.3.2 enum coco::convex_info

This enum is used to store information about the convexity of a node or an expression.

Enumerator:

c_convex The expression is convex and not linear

c_linear The expression is linear

c_concave The expression is concave and not linear

c_maybe The convexity type of the expression is unknown

c_not The expression is neither convex nor concave

c_constant The expression is constant

Definition at line 45 of file convex_info.h.

9.1.3.3 enum coco::dbc_method

Enumerator:

dbc_cmp_eq

dbc_cmp_ne

dbc_cmp_gt

dbc_cmp_ge

dbc_cmp_lt

dbc_cmp_le

dbc_cmp_in

dbc_cmp_abseq

dbc_cmp_absne

dbc_cmp_absgt

dbc_cmp_absge

dbc_cmp_abslt

dbc_cmp_absle

dbc_cmp_absin

dbc_sort_min

dbc_sort_max

dbc_sort_absmin

dbc_sort_absmax

dbc_sort_true

dbc_sort_false

Definition at line 64 of file dbcompare.h.

9.1.3.4 enum coco::e_expression_type

This enum is used for determining the type of the expression in the `is` method and similar methods and operators

Enumerator:

ex_bound a variable
ex_linear linear (but not a single variable)
ex_quadratic quadratic non-linear
ex_polynomial non-quadratic polynomial
ex_other non-polynomial non-linear
ex_kj only additional vars, constr. from KJ-conditions
ex_org original vars, constr. (ex_kj, ex_org both set = none set)
ex_redundant this constraint is known to be redundant
ex_notredundant this constraint is known to be not redundant
ex_active_lo this constraint is possible active at the lower bound
ex_inactive_lo this constraint is inactive at the lower bound
ex_active_hi this constraint is possible active at the upper bound
ex_inactive_hi this constraint is inactive at the upper bound
ex_active this constraint is possible active
ex_inactive this constraint is inactive
ex_integer this expression is an integer expression
ex_exists this variable has an exist-context
ex_forall this variable has a for-all-context
ex_free this variable is free
ex_stochastic this variable is a stochastic variable
ex_convex this expression is known to be convex
ex_concave this expression is known to be concave
ex_inequality this is an inequality constraint
ex_equality this is an equality constraint
ex_leftbound this is a constraint with lower bound
ex_rightbound this is a constraint with upper bound
ex_atmlin at most linear
ex_atmquad at most quadratic
ex_atmpoly at most polynomial
ex_nonlin non linear
ex_nonbnd not bounds
ex_any any
ex_bothbound this is a two-sided constraint

Definition at line 402 of file expression.h.

9.1.3.5 enum coco::fset_type

Enumerator:

fset_unbd
fset_box
fset_lin
fset_iquad
fset_equad
fset_ipoly
fset_epoly
fset_inonpoly
fset_enonpoly

Definition at line 39 of file structure_defs.h.

9.1.3.6 enum coco::func_type

Enumerator:

func_const
func_lin
func_quad
func_poly
func_nonpoly

Definition at line 36 of file structure_defs.h.

9.1.3.7 enum coco::sp_indexing

Available indexing methods for storing the nonzero positions of sparse matrices full_indexing: the row and column index vectors have the same length and the vectors specify pairs of row and column indices;

row_wise_indexing: the column vector contains the column indices of the nonzeros row-by-row. The number of nonzeros in row i (i=0..r.size()-2) is given by r[i+1]-r[i]; if this number is nonzero, then r[i] contains the index of the column vector, where the non-zero column indices for this row start. Zero rows in the bottom of the matrix does not appear in r; that is, the index i=r.size()-1 is the last nonzero row of the matrix, and r[i] contains the index of the column vector, where the non-zero column indices for this row start.

column_wise_indexing: this is the same as row_wise_indexing, with the roles of the row and column index vectors interchanged.

Enumerator:

full_indexing
row_wise_indexing
column_wise_indexing
full_indexing
row_wise_indexing
column_wise_indexing

Definition at line 54 of file sparsity.h.

9.1.3.8 enum coco::sp_indexing

Available indexing methods for storing the nonzero positions of sparse 3-tensors full_indexing: the row, column, and depth index vectors have the same length and the vectors specify triples of row, column, and depth indices;

row_wise_indexing: the column vector contains the column indices of the nonzeros row-by-row. The number of nonzeros in row i (i=0..r.size()-2) is given by r[i+1]-r[i]; if this number is nonzero, then r[i] contains the index of the column vector, where the non-zero column indices for this row start. Zero rows in the bottom of the matrix does not appear in r; that is, the index i=r.size()-1 is the last nonzero row of the matrix, and r[i] contains the index of the column vector, where the non-zero column indices for this row start.

column_wise_indexing: this is the same as row_wise_indexing, with the roles of the row and column index vectors interchanged.

Enumerator:

full_indexing

row_wise_indexing

column_wise_indexing

full_indexing

row_wise_indexing

column_wise_indexing

Definition at line 55 of file sparsity3.h.

9.1.3.9 enum coco::tristate_e

The tristate type is a boolean with the addition of a maybe value

Enumerator:

t_true

t_false

t_maybe

Definition at line 36 of file coconut_types.h.

9.1.3.10 enum coco::type_annotation

This enum is used to store information about the type of a node or an expression.

Enumerator:

v_exists The expression/variable is interpreted in a exists context.

v_forall The expression/variable is interpreted in a forall context.

v_free This is valid for variables only, and it defines a free variable.

v_stochastic This expression is stochastic.

Definition at line 46 of file semantics.h.

9.1.4 Function Documentation

9.1.4.1 std::ostream& coco::__wr_interval (std::ostream & *o,* const interval & *__i*) [inline]

This is an internal routine for writing an interval in the .dag format ("-I" for −COCO_INF and "I" for COCO_INF).

Definition at line 344 of file expression.hpp.

9.1.4.2 void coco::_internal_graphorder (const model & *DAG,* std::vector< unsigned int > & *order,* std::vector< unsigned int > * *inv_order*) [inline]

Definition at line 95 of file graphorder.cc.

9.1.4.3 interval coco::_internal_imperfect_eval (const interval & *x,* unsigned int *k,* const expression_const_walker & *result,* const model * *mod,* const variable_indicator & *vind,* const std::map< unsigned int, d1func * > * *p_infos*) [inline]

Definition at line 204 of file dagd1func.cc.

9.1.4.4 hessNumber coco::abs (const hessNumber & *a*)

Definition at line 115 of file hessNumber.cc.

9.1.4.5 proj_rational coco::abs (const proj_rational & *r*) [inline]

Definition at line 120 of file prointerval.h.

9.1.4.6 ihessNumber coco::abs (const ihessNumber & *a*)

Definition at line 130 of file ihessNumber.cc.

9.1.4.7 diffNumber coco::abs (const diffNumber & *a*)

Definition at line 145 of file diffNumber.cc.

9.1.4.8 interval coco::abs (const interval & *x*)

Returns an interval enclosure of the absolute value of the interval x.

9.1.4.9 diffI coco::abs (const diffI & *a*)

Definition at line 684 of file diffI.cc.

9.1.4.10 interval_set coco::abs (const interval_set & *a*)

Definition at line 847 of file interval_set.cc.

9.1.4.11 template<class I > projective_interval<I> coco::abs (const projective_interval< I > & *x*)

Definition at line 879 of file prointerval.hpp.

9.1.4.12 double coco::absmin (const interval & __i) [inline]

This function returns the element of __i with smallest absolute value.

Definition at line 784 of file interval_boost.h.

9.1.4.13 interval coco::acos (const interval & x)

Returns an interval enclosure of the inverse cosine of the interval x.

9.1.4.14 template<class I > projective_interval<I> coco::acos (const projective_interval< I > & x)

Definition at line 870 of file prointerval.hpp.

9.1.4.15 interval_set coco::acos (const interval_set & a)

Definition at line 919 of file interval_set.cc.

9.1.4.16 interval coco::acosh (const interval & x)

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

9.1.4.17 template<class I > projective_interval<I> coco::acosh (const projective_interval< I > & x)

Definition at line 887 of file prointerval.hpp.

9.1.4.18 interval_set coco::acosh (const interval_set & a)

Definition at line 1015 of file interval_set.cc.

9.1.4.19 interval coco::acot (const interval & x)

Returns an interval enclosure of the inverse cotangent of the interval x.

9.1.4.20 template<class I > projective_interval<I> coco::acot (const projective_interval< I > & x)

Definition at line 896 of file prointerval.hpp.

9.1.4.21 interval_set coco::acot (const interval_set & a)

Definition at line 931 of file interval_set.cc.

9.1.4.22 interval coco::acoth (const interval & x)

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

9.1.4.23 template<class I > projective_interval<I> coco::acoth (const projective_interval< I > & x)

Definition at line 905 of file prointerval.hpp.

9.1.4.24 interval_set coco::acoth (const interval_set & a)

Definition at line 1027 of file interval_set.cc.

9.1.4.25 interval coco::asin (const interval & *x*)

Returns an interval enclosure of the inverse sine of the interval x.

9.1.4.26 interval_set coco::asin (const interval_set & *a*)

Definition at line 907 of file interval_set.cc.

9.1.4.27 template<class I > projective_interval<I> coco::asin (const projective_interval< I > & *x*)

Definition at line 914 of file prointerval.hpp.

9.1.4.28 interval coco::asinh (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

9.1.4.29 template<class I > projective_interval<I> coco::asinh (const projective_interval< I > & *x*)

Definition at line 923 of file prointerval.hpp.

9.1.4.30 interval_set coco::asinh (const interval_set & *a*)

Definition at line 1003 of file interval_set.cc.

9.1.4.31 interval coco::asqr_find_f1min (const interval & *c1,* const interval & *c3,* const interval & *x2*) [inline]

Here we find a guaranteed enclosure for the minimum of f1(p2) = (c3-p2+c1)^2 + 4∗c1∗p2 = p2^2 + 2∗p2∗(c1-c3) + c3^2 + c1^2 + 2∗c1∗c3, with p2 x2, where c1 and c3 are constant point intervals.

Definition at line 51 of file asqr.h.

9.1.4.32 hessNumber coco::atan (const hessNumber & *a*)

Definition at line 230 of file hessNumber.cc.

9.1.4.33 ihessNumber coco::atan (const ihessNumber & *a*)

Definition at line 283 of file ihessNumber.cc.

9.1.4.34 diffNumber coco::atan (const diffNumber & *a*)

Definition at line 398 of file diffNumber.cc.

9.1.4.35 interval coco::atan (const interval & *x*)

Returns an interval enclosure of the inverse tangent of the interval x.

9.1.4.36 diffI coco::atan (const diffI & *a*)

Definition at line 890 of file diffI.cc.

9.1.4.37 template<class I > projective_interval<I> coco::atan (const projective_interval< I > & *x*)

Definition at line 932 of file prointerval.hpp.

9.1.4.38 interval_set coco::atan (const interval_set & *a*)

Definition at line 943 of file interval_set.cc.

9.1.4.39 hessNumber coco::atan2 (const hessNumber & *a,* const hessNumber & *b*)

Definition at line 213 of file hessNumber.cc.

9.1.4.40 ihessNumber coco::atan2 (const ihessNumber & *a,* const ihessNumber & *b*)

Definition at line 266 of file ihessNumber.cc.

9.1.4.41 diffNumber coco::atan2 (const diffNumber & *a,* const diffNumber & *b*)

Definition at line 381 of file diffNumber.cc.

9.1.4.42 interval coco::atan2 (const interval & *y,* const interval & *x*) [inline]

This function computes the atan2 function $\mathrm{atan}(y/x)$ for intervals.

Definition at line 777 of file interval_boost.h.

9.1.4.43 diffI coco::atan2 (const diffI & *a,* const diffI & *b*)

Definition at line 867 of file diffI.cc.

9.1.4.44 interval_set coco::atan2 (const interval_set & *a,* const interval_set & *b*)

Definition at line 1066 of file interval_set.cc.

9.1.4.45 template<class I > projective_interval<I> coco::atan2 (const projective_interval< I > & *x,* const projective_interval< I > & *y*)

Definition at line 1151 of file prointerval.hpp.

9.1.4.46 interval coco::atanh (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

9.1.4.47 template<class I > projective_interval<I> coco::atanh (const projective_interval< I > & *x*)

Definition at line 941 of file prointerval.hpp.

9.1.4.48 interval_set coco::atanh (const interval_set & *a*)

Definition at line 1039 of file interval_set.cc.

9.1.4.49 const basic_alltype & coco::coco::basic_alltype_empty ( )

Definition at line 44 of file basic_alltype.cc.

9.1.4.50 interval coco::cancel (const interval & *a*, const interval & *b*)

9.1.4.51 hessNumber coco::cos (const hessNumber & *a*)

Definition at line 92 of file hessNumber.cc.

9.1.4.52 ihessNumber coco::cos (const ihessNumber & *a*)

Definition at line 107 of file ihessNumber.cc.

9.1.4.53 diffNumber coco::cos (const diffNumber & *a*)

Definition at line 111 of file diffNumber.cc.

9.1.4.54 diffI coco::cos (const diffI & *a*)

Definition at line 629 of file diffI.cc.

9.1.4.55 interval coco::cos (const interval & *x*)

Returns an interval enclosure of the cosine of the interval x.

9.1.4.56 interval_set coco::cos (const interval_set & *a*)

Definition at line 871 of file interval_set.cc.

9.1.4.57 template<class I > projective_interval<I> coco::cos (const projective_interval< I > & *x*)

Definition at line 950 of file prointerval.hpp.

9.1.4.58 hessNumber coco::cosh (const hessNumber & *a*)

Definition at line 201 of file hessNumber.cc.

9.1.4.59 ihessNumber coco::cosh (const ihessNumber & *a*)

Definition at line 254 of file ihessNumber.cc.

9.1.4.60 diffNumber coco::cosh (const diffNumber & *a*)

Definition at line 366 of file diffNumber.cc.

9.1.4.61 interval coco::cosh (const interval & *x*)

Returns an interval enclosure of the hyperbolic cosine of the interval x.

9.1.4.62 diffI coco::cosh (const diffI & *a*)

Definition at line 846 of file diffI.cc.

9.1.4.63 template<class I > projective_interval<I> coco::cosh (const projective_interval< I > & *x*)

Definition at line 959 of file prointerval.hpp.

9.1.4.64 interval_set coco::cosh (const interval_set & *a*)

Definition at line 967 of file interval_set.cc.

9.1.4.65 interval coco::cot (const interval & *x*)

Returns an interval enclosure of the cotangent of the interval x.

9.1.4.66 interval_set coco::cot (const interval_set & *a*)

Definition at line 883 of file interval_set.cc.

9.1.4.67 template<class I > projective_interval<I> coco::cot (const projective_interval< I > & *x*)

Definition at line 968 of file prointerval.hpp.

9.1.4.68 interval coco::coth (const interval & *x*)

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

9.1.4.69 template<class I > projective_interval<I> coco::coth (const projective_interval< I > & *x*)

Definition at line 977 of file prointerval.hpp.

9.1.4.70 interval_set coco::coth (const interval_set & *a*)

Definition at line 979 of file interval_set.cc.

9.1.4.71 std::string coco::datetime (const struct tm * *timeptr*)

This function returns the local time as C++ string.

Definition at line 46 of file writer_utils.cc.

9.1.4.72 template<typename _TR > _TR coco::dbccmps_abs (const _TR & *x*)

Definition at line 118 of file dbcompare.hpp.

9.1.4.73 double coco::diam (const interval &)

diam(a) == a.diam()

9.1.4.74 double coco::dist (const interval & *x*, const interval & *y*)

Same as x.dist(y)

9.1.4.75 interval_set coco::divide (const interval & *a*, double *b*) [inline]

Definition at line 449 of file interval_set.h.

9.1.4.76 interval_set coco::divide (double *a*, const interval & *b*) [inline]

Definition at line 453 of file interval_set.h.

9.1.4.77 interval_set coco::divide (const interval & *a,* const interval & *b*)

Definition at line 660 of file interval_set.cc.

9.1.4.78 interval coco::division_part1 (const interval & *x,* const interval & *y,* bool & *b*)

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

9.1.4.79 template<class I > projective_interval<I> coco::division_part1 (const projective_interval< I > & *x,* const projective_interval< I > & *y,* bool & *b*)

Definition at line 1161 of file prointerval.hpp.

9.1.4.80 interval coco::division_part2 (const interval & *x,* const interval & *y,* bool *b* = `true`)

9.1.4.81 template<class I > projective_interval<I> coco::division_part2 (const projective_interval< I > & *x,* const projective_interval< I > & *y,* bool *b*)

Definition at line 1171 of file prointerval.hpp.

9.1.4.82 double coco::downward_divides (const double & *a,* const double & *b*)

This function divides `a` by `b`, rounding downwards.

9.1.4.83 double coco::downward_minus (const double & *a,* const double & *b*)

This function subtracts `b` from `a`, rounding downwards.

9.1.4.84 double coco::downward_multiplies (const double & *a,* const double & *b*)

This function multiplies `a` by `b`, rounding downwards.

9.1.4.85 double coco::downward_plus (const double & *a,* const double & *b*)

This function adds `a` and `b`, rounding downwards.

9.1.4.86 std::string coco::e2D (double *d*)

This function converts a double number to a double precision Fortran representation in a C++ string.

Definition at line 78 of file writer_utils.cc.

9.1.4.87 bool coco::eval (const vdbl::rowid & *rid,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* const basic_alltype & *cmpval*)

Definition at line 312 of file dbcompare.cc.

9.1.4.88 interval coco::eval_asqr (const interval & *x1,* const interval & *x2,* const interval & *x3*) [inline]

Evaluate the optimal range for an ASQR node.

Definition at line 68 of file asqr.h.

9.1.4.89 template<typename _TR > bool coco::eval_in (const vdbl::rowid & *rid,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* _TR , const basic_alltype & *l*)

Definition at line 39 of file dbcompare.cc.

9.1.4.90 template<> bool coco::eval_in< interval > (const vdbl::rowid & *rid,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* interval , const basic_alltype & *l*)

Definition at line 86 of file dbcompare.cc.

9.1.4.91 template<> bool coco::eval_in< std::string > (const vdbl::rowid & *rid,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* std::string , const basic_alltype & *l*)

Definition at line 106 of file dbcompare.cc.

9.1.4.92 template<typename _TR > bool coco::eval_s (const vdbl::rowid & *rid,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* _TR *cmpval*)

Definition at line 127 of file dbcompare.cc.

9.1.4.93 template<> bool coco::eval_s< interval > (const vdbl::rowid & *rid,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* interval *cmpval*)

Definition at line 195 of file dbcompare.cc.

9.1.4.94 template<typename _TR > bool coco::eval_u (const vdbl::rowid & *rid,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* _TR *cmpval*)

Definition at line 263 of file dbcompare.cc.

9.1.4.95 hessNumber coco::exp (const hessNumber & *a*)

Definition at line 70 of file hessNumber.cc.

9.1.4.96 diffNumber coco::exp (const diffNumber & *a*)

Definition at line 84 of file diffNumber.cc.

9.1.4.97 ihessNumber coco::exp (const ihessNumber & *a*)

Definition at line 85 of file ihessNumber.cc.

9.1.4.98 diffI coco::exp (const diffI & *a*)

Definition at line 585 of file diffI.cc.

9.1.4.99 interval coco::exp (const interval & *x*)

Returns an interval enclosure of the exponential of the interval x.

9.1.4.100 interval_set coco::exp (const interval_set & *a*)

Definition at line 751 of file interval_set.cc.

9.1.4.101 template<class I > projective_interval<I> coco::exp (const projective_interval< I > & *x*)

Definition at line 986 of file prointerval.hpp.

9.1.4.102 interval coco::exp10 (const interval & *x*)

Returns an interval enclosure of the exponential (base 10) of the interval x.

9.1.4.103 interval_set coco::exp10 (const interval_set & *a*)

Definition at line 763 of file interval_set.cc.

9.1.4.104 template<class I > projective_interval<I> coco::exp10 (const projective_interval< I > & *x*)

Definition at line 995 of file prointerval.hpp.

9.1.4.105 interval coco::exp2 (const interval & *x*)

Returns an interval enclosure of the exponential (base 2) of the interval x.

9.1.4.106 interval_set coco::exp2 (const interval_set & *a*)

Definition at line 775 of file interval_set.cc.

9.1.4.107 template<class I > projective_interval<I> coco::exp2 (const projective_interval< I > & *x*)

Definition at line 1004 of file prointerval.hpp.

9.1.4.108 interval coco::expm1 (const interval & *x*)

Returns an interval enclosure of exp(x)-1

9.1.4.109 interval_set coco::expm1 (const interval_set & *a*)

Definition at line 787 of file interval_set.cc.

9.1.4.110 template<class I > projective_interval<I> coco::expm1 (const projective_interval< I > & *x*)

Definition at line 1013 of file prointerval.hpp.

9.1.4.111 template<typename _AT > std::vector<std::pair<vdbl::rowid,_AT> >::iterator coco::filter (typename std::vector< std::pair< vdbl::rowid, _AT > > & *rid,* typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator *rid_b,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* unsigned int *n,* std::pair< double, double > *rel_abs*)

Definition at line 403 of file dbcompare.hpp.

9.1.4.112 template<typename _AT > void coco::filter_from_view (typename std::vector< std::pair< vdbl::rowid, _AT > > & *riv,* vdbl::view * *iv,* const std::map< int, std::triple< std::string, dbc_method, basic_alltype > > & *cols,* const std::vector< std::vector< int > > & *expr*)

Definition at line 41 of file dbcompare.hpp.

9.1.4.113 template<typename _TR , typename _AT > std::vector<std::pair<vdbl::rowid,_AT> >::iterator coco::filter_s (typename std::vector< std::pair< vdbl::rowid, _AT > > & *rid,* typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator *rid_b,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* unsigned int *n,* _TR , double *rel,* double *abs*)

Definition at line 239 of file dbcompare.hpp.

9.1.4.114 template<typename _TR , typename _AT > std::vector<std::pair<vdbl::rowid,_AT> >::iterator coco::filter_u (typename std::vector< std::pair< vdbl::rowid, _AT > > & *rid,* typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator *rid_b,* const std::string & *colname,* dbc_method *tp,* vdbl::view * *iv,* unsigned int *n,* _TR)

Definition at line 326 of file dbcompare.hpp.

9.1.4.115 interval coco::flb (const interval & *x1,* const interval & *x2,* const interval & *x3*) [inline]

Evaluate f(x1,x2,x3) = (x3-x2+x1)^2 + 4*x1*x2 with interval arguments. This is actually used with point intervals to get proper lower and upper bounds at the extremal points.

Definition at line 40 of file asqr.h.

9.1.4.116 double coco::gainfactor (const interval & *_old,* const interval & *_new*) [inline]

The call `gainfactor(o,n)` computes a safeguarded quotient of the widths of @ o and @ n under the assumption that `n` is a subset of `o`.

Definition at line 799 of file interval_boost.h.

9.1.4.117 interval coco::gauss (const interval & *x*) [inline]

This function computes the gaussian function e^{-x^2} for intervals.

Definition at line 770 of file interval_boost.h.

9.1.4.118 bool coco::get_children_vector (const search_graph & *__sgraph,* const search_node_id & *pid,* std::vector< si_ri_pair > & *v,* search_inspector * *sfoc*)

Definition at line 59 of file search_graph_funcs.cc.

9.1.4.119 std::string coco::get_diff_class (int *i*)

Definition at line 37 of file diff_info.cc.

9.1.4.120 void coco::get_ids (const std::vector< si_ri_pair > & *riv,* std::vector< search_node_id > & *v*)

Definition at line 34 of file search_graph_funcs.cc.

9.1.4.121 termination_reason coco::get_locopt_termr_from_rval (const locopt_ret & *ret_info,* const std::string & *iename,* const int *srval*)

Definition at line 49 of file locopt_ret.cc.

9.1.4.122 double coco::get_locopt_weight_from_rval (const locopt_ret & *ret_info,* const std::string & *iename,* const int *srval*)

Definition at line 34 of file locopt_ret.cc.

9.1.4.123 tableid coco::get_point_table (database * *ptr,* standard_table *& *ptb,* userid *_uid*)

Definition at line 41 of file dbtables.cc.

9.1.4.124 void coco::hess_star_color1 (const unsigned int *n,* const vmtl::sparse_matrix< int > & *H,* std::vector< std::vector< unsigned int > > & *parts,* std::vector< vmtl::sparse_vector< unsigned int > > & *provides*)

Definition at line 35 of file hess_color.cc.

9.1.4.125 interval coco::hull (const interval & *__x,* const interval & *__y*) [inline]

Definition at line 1006 of file interval_filib.h.

9.1.4.126 interval coco::hulldiff (const interval & *__x,* const interval & *__y*) [inline]

Definition at line 1012 of file interval_filib.h.

9.1.4.127 std::string coco::i2a (int *x*)

This function converts integers to a C++ string.

Definition at line 37 of file writer_utils.cc.

9.1.4.128 interval_set coco::imax (const interval_set & *a,* const interval_set & *b*)

Definition at line 700 of file interval_set.cc.

9.1.4.129 interval coco::imax (const interval & *x,* const interval & *y*)

Returns an interval enclosure of the maximum of two intervals x and y.

9.1.4.130 template<class I > projective_interval<I> coco::imax (const projective_interval< I > & *x,* const projective_interval< I > & *y*)

Definition at line 1133 of file prointerval.hpp.

9.1.4.131 interval_set coco::imin (const interval_set & *a*, const interval_set & *b*)

Definition at line 685 of file interval_set.cc.

9.1.4.132 interval coco::imin (const interval & *x*, const interval & *y*)

Returns an interval enclosure of the minimum of two intervals x and y.

9.1.4.133 template<class I > projective_interval<I> coco::imin (const projective_interval< I > & *x*, const projective_interval< I > & *y*)

Definition at line 1142 of file prointerval.hpp.

9.1.4.134 interval coco::intersect (const interval & *__x*, const interval & *__y*) [inline]

Definition at line 1000 of file interval_filib.h.

9.1.4.135 interval_set coco::intersect_inv_hlp (d1func * *d1help*, const interval & *xx*, const interval_set & *r*, int *order*) [inline]

Definition at line 176 of file dagd1func_bp.cc.

9.1.4.136 interval coco::ipow (const interval & *x*, int *n*) [inline]

This function computes the integer power operation x^n for intervals.

Definition at line 763 of file interval_boost.h.

9.1.4.137 bool coco::less_equal (const basic_alltype & *a*, const basic_alltype & *b*)

Total order comparison operator

Definition at line 557 of file basic_alltype.cc.

9.1.4.138 bool coco::less_than (const basic_alltype & *a*, const basic_alltype & *b*)

Total order comparison operator

Definition at line 555 of file basic_alltype.cc.

9.1.4.139 bool coco::lexicographic_le (const interval & *a*, const interval & *b*) [inline]

This comparison operator returns the "lexicographically less than or equal to interval relation", i.e. it is true iff `lexicographic_le(T, X) == T.inf() < X.inf() || (T.inf() == X.inf() && T.sup() <= X.sup())` .

Definition at line 696 of file interval_filib.h.

9.1.4.140 bool coco::lexicographic_lt (const interval & *a*, const interval & *b*) [inline]

This comparison operator returns the "lexicographically less than interval relation", i.e. it is true iff `lexicographic_lt(T, X) == T.inf() < X.inf() || (T.inf() == X.inf() && T.sup() < X.sup())` .

Definition at line 688 of file interval_filib.h.

9.1.4.141 template<typename _AT > void coco::list_from_view (typename std::vector< std::pair< vdbl::rowid, _AT > > & *riv,* vdbl::view * *iv,* const std::map< int, std::triple< std::string, dbc_method, basic_alltype > > & *cols,* const std::vector< int > & *sort,* unsigned int *n* = 1)

Definition at line 441 of file dbcompare.hpp.

9.1.4.142 std::string coco::localdatetime ( )

This function converts a date/time structure to a C++ string.

Definition at line 68 of file writer_utils.cc.

9.1.4.143 hessNumber coco::log (const hessNumber & *a*)

Definition at line 179 of file hessNumber.cc.

9.1.4.144 ihessNumber coco::log (const ihessNumber & *a*)

Definition at line 232 of file ihessNumber.cc.

9.1.4.145 diffNumber coco::log (const diffNumber & *a*)

Definition at line 338 of file diffNumber.cc.

9.1.4.146 interval coco::log (const interval & *x*)

Returns an interval enclosure of the natural logarithm of the interval x.

9.1.4.147 interval_set coco::log (const interval_set & *a*)

Definition at line 799 of file interval_set.cc.

9.1.4.148 diffI coco::log (const diffI & *a*)

Definition at line 806 of file diffI.cc.

9.1.4.149 template<class I > projective_interval<I> coco::log (const projective_interval< I > & *x*)

Definition at line 1022 of file prointerval.hpp.

9.1.4.150 interval coco::log10 (const interval & *x*)

Returns an interval enclosure of the logarithm (base 10) of the interval x.

9.1.4.151 interval_set coco::log10 (const interval_set & *a*)

Definition at line 811 of file interval_set.cc.

9.1.4.152 template<class I > projective_interval<I> coco::log10 (const projective_interval< I > & *x*)

Definition at line 1031 of file prointerval.hpp.

9.1.4.153 interval coco::log1p (const interval & *x*)

Returns an interval enclosure of log(1+x).

9.1.4.154 interval_set coco::log1p (const interval_set & *a*)

Definition at line 823 of file interval_set.cc.

9.1.4.155 template<class I > projective_interval<I> coco::log1p (const projective_interval< I > & *x*)

Definition at line 1040 of file prointerval.hpp.

9.1.4.156 interval coco::log2 (const interval & *x*)

Returns an interval enclosure of the logarithm (base 2) of the interval x.

9.1.4.157 interval_set coco::log2 (const interval_set & *a*)

Definition at line 835 of file interval_set.cc.

9.1.4.158 template<class I > projective_interval<I> coco::log2 (const projective_interval< I > & *x*)

Definition at line 1049 of file prointerval.hpp.

9.1.4.159 double coco::mag (const interval &)

mag(a) == a.mag()

9.1.4.160 bool coco::mat_inv (const std::vector< std::vector< double > > & *A0*, std::vector< std::vector< double > > & *R*)

Definition at line 55 of file NGS_step.cc.

9.1.4.161 hessNumber coco::max (const hessNumber & *a*, const hessNumber & *b*)

9.1.4.162 diffNumber coco::max (const diffNumber & *a*, const diffNumber & *b*)

Definition at line 156 of file diffNumber.cc.

9.1.4.163 ihessNumber coco::max (const ihessNumber & *a*, const ihessNumber & *b*)

Definition at line 198 of file ihessNumber.cc.

9.1.4.164 diffI coco::max (const diffI & *a*, const diffI & *b*)

Definition at line 772 of file diffI.cc.

9.1.4.165 double coco::mid (const interval &)

mid(a) == a.mid()

9.1.4.166 double coco::mig (const interval &)

mig(a) == a.mig()

9.1.4.167 hessNumber coco::min (const hessNumber & *a*, const hessNumber & *b*)

9.1.4.168 diffNumber coco::min (const diffNumber & *a*, const diffNumber & *b*)

Definition at line 164 of file diffNumber.cc.

9.1.4.169 ihessNumber coco::min (const ihessNumber & *a*, const ihessNumber & *b*)

Definition at line 215 of file ihessNumber.cc.

9.1.4.170 diffI coco::min (const diffI & *a*, const diffI & *b*)

Definition at line 789 of file diffI.cc.

9.1.4.171 bool coco::Newton_Step (const std::vector< std::vector< interval > > & *J*, const int *N*, std::vector< interval > *x*, const std::vector< interval > & *c*, const std::vector< interval > & *fc*, std::vector< interval > & *xout*, bool *precond*)

Definition at line 803 of file NGS_step.cc.

9.1.4.172 bool coco::Newton_Step (const std::vector< std::vector< interval > > & *J*, const int *N*, std::vector< interval > *x*, const std::vector< interval > & *c*, const std::vector< interval > & *fc*, std::vector< std::vector< interval > > & *splits*, bool *precond*)

Definition at line 827 of file NGS_step.cc.

9.1.4.173 bool coco::Newton_Step_1D (const interval & *J*, interval *x*, const interval & *c*, const interval & *fc*, interval & *xout*, bool *precond*)

Definition at line 845 of file NGS_step.cc.

9.1.4.174 bool coco::Newton_Step_1D (const interval & *J*, interval *x*, const interval & *c*, const interval & *fc*, std::vector< interval > & *splits*, bool *precond*)

Definition at line 865 of file NGS_step.cc.

9.1.4.175 bool coco::operator!= (int *n*, const proj_rational & *R*) [inline]

Definition at line 119 of file prointerval.h.

9.1.4.176 bool coco::operator!= (const interval_set & *a*, const interval_set & *b*) [inline]

Definition at line 166 of file interval_set.h.

9.1.4.177 template<class _TC > bool coco::operator!= (const interval & *__i*, const _TC & *__d*) [inline]

Definition at line 822 of file interval_boost.h.

9.1.4.178 bool coco::operator!= (const interval & __i, const interval & __d) [inline]

Comparison operator (not equal)

Definition at line 518 of file interval_boost.h.

9.1.4.179 bool coco::operator!= (const interval & a, double b) [inline]

Definition at line 520 of file interval_boost.h.

9.1.4.180 hessNumber coco::operator* (const hessNumber & a, const hessNumber & b)

Definition at line 32 of file hessNumber.cc.

9.1.4.181 ihessNumber coco::operator* (const ihessNumber & a, const ihessNumber & b)

Definition at line 32 of file ihessNumber.cc.

9.1.4.182 diffNumber coco::operator* (const diffNumber & a, const diffNumber & b)

Definition at line 36 of file diffNumber.cc.

9.1.4.183 diffNumber coco::operator* (const double & a, const diffNumber & b) [inline]

Definition at line 254 of file diffNumber.h.

9.1.4.184 diffNumber coco::operator* (const diffNumber & a, const double & b) [inline]

Definition at line 263 of file diffNumber.h.

9.1.4.185 hessNumber coco::operator* (const double & a, const hessNumber & b) [inline]

Definition at line 272 of file hessNumber.h.

9.1.4.186 hessNumber coco::operator* (const hessNumber & a, const double & b) [inline]

Definition at line 281 of file hessNumber.h.

9.1.4.187 ihessNumber coco::operator* (const double & a, const ihessNumber & b) [inline]

Definition at line 330 of file ihessNumber.h.

9.1.4.188 ihessNumber coco::operator* (const interval & a, const ihessNumber & b) [inline]

Definition at line 321 of file ihessNumber.h.

9.1.4.189 ihessNumber coco::operator* (const ihessNumber & a, const double & b) [inline]

Definition at line 339 of file ihessNumber.h.

9.1.4.190 ihessNumber coco::operator* (const ihessNumber & a, const interval & b) [inline]

Definition at line 348 of file ihessNumber.h.

9.1.4.191 interval_set coco::operator∗ (const interval_set & *a,* const interval_set & *b*) [inline]

Definition at line 411 of file interval_set.h.

9.1.4.192 interval_set coco::operator∗ (const interval_set & *a,* const interval & *b*) [inline]

Definition at line 419 of file interval_set.h.

9.1.4.193 interval_set coco::operator∗ (const interval & *a,* const interval_set & *b*) [inline]

Definition at line 426 of file interval_set.h.

9.1.4.194 interval_set coco::operator∗ (const interval_set & *a,* double *b*) [inline]

Definition at line 433 of file interval_set.h.

9.1.4.195 interval_set coco::operator∗ (double *a,* const interval_set & *b*) [inline]

Definition at line 440 of file interval_set.h.

9.1.4.196 diffI coco::operator∗ (const double & *a,* const diffI & *b*)

Definition at line 327 of file diffI.cc.

9.1.4.197 diffI coco::operator∗ (const diffI & *a,* const double & *b*)

Definition at line 342 of file diffI.cc.

9.1.4.198 diffI coco::operator∗ (const interval & *a,* const diffI & *b*)

Definition at line 372 of file diffI.cc.

9.1.4.199 diffI coco::operator∗ (const diffI & *a,* const interval & *b*)

Definition at line 387 of file diffI.cc.

9.1.4.200 diffI coco::operator∗ (const diffI & *a,* const diffI & *b*)

Definition at line 493 of file diffI.cc.

9.1.4.201 interval coco::operator∗ (const interval & *a,* const interval & *b*)

This operator computes a∗b as interval.

9.1.4.202 interval coco::operator∗ (const interval & *a,* double *b*)

9.1.4.203 interval coco::operator∗ (double *b,* const interval & *a*)

9.1.4.204 template<class I > projective_interval<I> coco::operator∗ (const projective_interval< I > & *a,* const projective_interval< I > & *b*)

Definition at line 818 of file prointerval.hpp.

9.1.4.205 template<class I > projective_interval<I> coco::operator∗ (const projective_interval< I > & *a,* double *b*)

Definition at line 828 of file prointerval.hpp.

9.1.4.206 template<class I > projective_interval<I> coco::operator∗ (double *b,* const projective_interval< I > & *a*)

Definition at line 836 of file prointerval.hpp.

9.1.4.207 diffNumber coco::operator+ (const diffNumber & *a,* const diffNumber & *b*) [inline]

Definition at line 180 of file diffNumber.h.

9.1.4.208 diffNumber coco::operator+ (const double & *a,* const diffNumber & *b*) [inline]

Definition at line 193 of file diffNumber.h.

9.1.4.209 hessNumber coco::operator+ (const hessNumber & *a,* const hessNumber & *b*) [inline]

Definition at line 205 of file hessNumber.h.

9.1.4.210 diffNumber coco::operator+ (const diffNumber & *a,* const double & *b*) [inline]

Definition at line 203 of file diffNumber.h.

9.1.4.211 hessNumber coco::operator+ (const double & *a,* const hessNumber & *b*) [inline]

Definition at line 214 of file hessNumber.h.

9.1.4.212 hessNumber coco::operator+ (const hessNumber & *a,* const double & *b*) [inline]

Definition at line 224 of file hessNumber.h.

9.1.4.213 ihessNumber coco::operator+ (const ihessNumber & *a,* const ihessNumber & *b*) [inline]

Definition at line 214 of file ihessNumber.h.

9.1.4.214 ihessNumber coco::operator+ (const double & *a,* const ihessNumber & *b*) [inline]

Definition at line 223 of file ihessNumber.h.

9.1.4.215 ihessNumber coco::operator+ (const interval & *a,* const ihessNumber & *b*) [inline]

Definition at line 233 of file ihessNumber.h.

9.1.4.216 ihessNumber coco::operator+ (const ihessNumber & *a,* const double & *b*) [inline]

Definition at line 243 of file ihessNumber.h.

9.1.4.217 ihessNumber coco::operator+ (const ihessNumber & *a*, const interval & *b*) [inline]

Definition at line 253 of file ihessNumber.h.

9.1.4.218 interval_set coco::operator+ (const interval_set & *a*, const interval_set & *b*) [inline]

Definition at line 332 of file interval_set.h.

9.1.4.219 interval_set coco::operator+ (const interval_set & *a*, const interval & *b*) [inline]

Definition at line 340 of file interval_set.h.

9.1.4.220 interval_set coco::operator+ (const interval & *a*, const interval_set & *b*) [inline]

Definition at line 347 of file interval_set.h.

9.1.4.221 interval_set coco::operator+ (const interval_set & *a*, double *b*) [inline]

Definition at line 354 of file interval_set.h.

9.1.4.222 interval_set coco::operator+ (double *a*, const interval_set & *b*) [inline]

Definition at line 361 of file interval_set.h.

9.1.4.223 diffI coco::operator+ (const diffI & *a*, const diffI & *b*)

Definition at line 162 of file diffI.cc.

9.1.4.224 diffI coco::operator+ (const double & *a*, const diffI & *b*)

Definition at line 181 of file diffI.cc.

9.1.4.225 diffI coco::operator+ (const diffI & *a*, const double & *b*)

Definition at line 195 of file diffI.cc.

9.1.4.226 diffI coco::operator+ (const interval & *a*, const diffI & *b*)

Definition at line 209 of file diffI.cc.

9.1.4.227 diffI coco::operator+ (const diffI & *a*, const interval & *b*)

Definition at line 223 of file diffI.cc.

9.1.4.228 work_node coco::operator+ (const work_node & *_w*, const delta_id & *_i*) [inline]

This function adds the delta with delta_id `_i` to work_node `_w` and returns a new work_node having the delta applied.

Definition at line 298 of file search_node.hpp.

9.1.4.229 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node coco::operator+ (const work_node & _w, const _Ctr< delta_id, _Al > & _d) [inline]

This function adds all the deltas whose delta_ids are in the linear container _d to work_node _w and returns a new work_node having all these deltas applied.

Definition at line 325 of file search_node.hpp.

9.1.4.230 interval coco::operator+ (const interval & *a*, const interval & *b*)

This operator computes a+b as interval.

9.1.4.231 interval coco::operator+ (const interval & *a*, double *b*)

9.1.4.232 interval coco::operator+ (double *b*, const interval & *a*)

9.1.4.233 template<class I > projective_interval<I> coco::operator+ (const projective_interval< I > & *a*, const projective_interval< I > & *b*)

Definition at line 702 of file prointerval.hpp.

9.1.4.234 template<class I > projective_interval<I> coco::operator+ (const projective_interval< I > & *a*, double *b*)

Definition at line 722 of file prointerval.hpp.

9.1.4.235 template<class I > projective_interval<I> coco::operator+ (double *b*, const projective_interval< I > & *a*)

Definition at line 741 of file prointerval.hpp.

9.1.4.236 work_node& coco::operator+= (work_node & _w, const delta_id & _i) [inline]

This function adds the delta with delta_id _i to work_node _w changing it in that process. The function returns _w.

Definition at line 311 of file search_node.hpp.

9.1.4.237 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node& coco::operator+= (work_node & _w, const _Ctr< delta_id, _Al > & _d) [inline]

This function adds the deltas whose delta_ids are in the linear container _d to work_node _w changing it in that process. The function returns _w.

Definition at line 341 of file search_node.hpp.

9.1.4.238 work_node coco::operator- (const work_node & _w, const delta_id & _d)

This function removes the delta with delta_id _i from work_node _w and returns a new work_node having the delta unapplied.

Definition at line 39 of file search_node.cc.

9.1.4.239 diffNumber coco::operator- (const diffNumber & *a,* const diffNumber & *b*) [inline]

Definition at line 213 of file diffNumber.h.

9.1.4.240 diffNumber coco::operator- (const double & *a,* const diffNumber & *b*) [inline]

Definition at line 226 of file diffNumber.h.

9.1.4.241 hessNumber coco::operator- (const hessNumber & *a,* const hessNumber & *b*) [inline]

Definition at line 234 of file hessNumber.h.

9.1.4.242 diffNumber coco::operator- (const diffNumber & *a,* const double & *b*) [inline]

Definition at line 236 of file diffNumber.h.

9.1.4.243 hessNumber coco::operator- (const double & *a,* const hessNumber & *b*) [inline]

Definition at line 243 of file hessNumber.h.

9.1.4.244 diffNumber coco::operator- (const diffNumber & *a*) [inline]

Definition at line 246 of file diffNumber.h.

9.1.4.245 hessNumber coco::operator- (const hessNumber & *a,* const double & *b*) [inline]

Definition at line 253 of file hessNumber.h.

9.1.4.246 hessNumber coco::operator- (const hessNumber & *a*) [inline]

Definition at line 263 of file hessNumber.h.

9.1.4.247 ihessNumber coco::operator- (const ihessNumber & *a,* const ihessNumber & *b*) [inline]

Definition at line 263 of file ihessNumber.h.

9.1.4.248 ihessNumber coco::operator- (const double & *a,* const ihessNumber & *b*) [inline]

Definition at line 272 of file ihessNumber.h.

9.1.4.249 ihessNumber coco::operator- (const interval & *a,* const ihessNumber & *b*) [inline]

Definition at line 282 of file ihessNumber.h.

9.1.4.250 ihessNumber coco::operator- (const ihessNumber & *a,* const double & *b*) [inline]

Definition at line 292 of file ihessNumber.h.

9.1.4.251 ihessNumber coco::operator- (const ihessNumber & *a,* const interval & *b*) [inline]

Definition at line 302 of file ihessNumber.h.

9.1.4.252 ihessNumber coco::operator- (const ihessNumber & *a*) [inline]

Definition at line 312 of file ihessNumber.h.

9.1.4.253 interval_set coco::operator- (const interval_set & *a*, const interval_set & *b*) [inline]

Definition at line 369 of file interval_set.h.

9.1.4.254 interval_set coco::operator- (const interval_set & *a*, const interval & *b*) [inline]

Definition at line 377 of file interval_set.h.

9.1.4.255 interval_set coco::operator- (const interval & *a*, const interval_set & *b*) [inline]

Definition at line 384 of file interval_set.h.

9.1.4.256 interval_set coco::operator- (const interval_set & *a*, double *b*) [inline]

Definition at line 391 of file interval_set.h.

9.1.4.257 interval_set coco::operator- (double *a*, const interval_set & *b*) [inline]

Definition at line 398 of file interval_set.h.

9.1.4.258 interval_set coco::operator- (const interval_set & *a*) [inline]

Definition at line 405 of file interval_set.h.

9.1.4.259 diffI coco::operator- (const diffI & *a*, const diffI & *b*)

Definition at line 237 of file diffI.cc.

9.1.4.260 diffI coco::operator- (const double & *a*, const diffI & *b*)

Definition at line 256 of file diffI.cc.

9.1.4.261 diffI coco::operator- (const diffI & *a*, const double & *b*)

Definition at line 270 of file diffI.cc.

9.1.4.262 diffI coco::operator- (const interval & *a*, const diffI & *b*)

Definition at line 284 of file diffI.cc.

9.1.4.263 diffI coco::operator- (const diffI & *a*, const interval & *b*)

Definition at line 298 of file diffI.cc.

9.1.4.264 diffI coco::operator- (const diffI & *a*)

Definition at line 312 of file diffI.cc.

9.1.4.265 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node coco::operator- (const work_node & _w, const _Ctr< delta_id, _Al > & _d) [inline]

This function removes all the deltas whose delta_ids are in the linear container _d from work_node _w and returns a new work_node having all these deltas unapplied.

Definition at line 365 of file search_node.hpp.

9.1.4.266 interval coco::operator- (const interval & a, const interval & b)

This operator computes a-b as interval.

9.1.4.267 interval coco::operator- (const interval & a, double b)

9.1.4.268 interval coco::operator- (double b, const interval & a)

9.1.4.269 template<class I > projective_interval<I> coco::operator- (const projective_interval< I > & a, const projective_interval< I > & b)

Definition at line 760 of file prointerval.hpp.

9.1.4.270 template<class I > projective_interval<I> coco::operator- (const projective_interval< I > & a, double b)

Definition at line 780 of file prointerval.hpp.

9.1.4.271 template<class I > projective_interval<I> coco::operator- (double b, const projective_interval< I > & a)

Definition at line 799 of file prointerval.hpp.

9.1.4.272 work_node& coco::operator-= (work_node & _w, const delta_id & _d)

This function removes (unapplies) the delta with delta_id _i from work_node _w changing it in that process. The function returns _w.

Definition at line 80 of file search_node.cc.

9.1.4.273 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node& coco::operator-= (work_node & _w, const _Ctr< delta_id, _Al > & _d) [inline]

This function removes (unapplies) the deltas whose delta_ids are in the linear container _d from work_node _w changing it in that process. The function returns _w.

Definition at line 383 of file search_node.hpp.

9.1.4.274 hessNumber coco::operator/ (const hessNumber & a, const hessNumber & b)

Definition at line 42 of file hessNumber.cc.

9.1.4.275 ihessNumber coco::operator/ (const ihessNumber & a, const ihessNumber & b)

Definition at line 42 of file ihessNumber.cc.

9.1.4.276 diffNumber coco::operator/ (const diffNumber & *a,* const diffNumber & *b*)

Definition at line 52 of file diffNumber.cc.

9.1.4.277 hessNumber coco::operator/ (const double & *a,* const hessNumber & *b*)

Definition at line 55 of file hessNumber.cc.

9.1.4.278 ihessNumber coco::operator/ (const double & *a,* const ihessNumber & *b*)

Definition at line 55 of file ihessNumber.cc.

9.1.4.279 diffNumber coco::operator/ (const diffNumber & *a,* const double & *b*) [inline]

Definition at line 272 of file diffNumber.h.

9.1.4.280 hessNumber coco::operator/ (const hessNumber & *a,* const double & *b*) [inline]

Definition at line 290 of file hessNumber.h.

9.1.4.281 ihessNumber coco::operator/ (const ihessNumber & *a,* const double & *b*) [inline]

Definition at line 357 of file ihessNumber.h.

9.1.4.282 ihessNumber coco::operator/ (const ihessNumber & *a,* const interval & *b*) [inline]

Definition at line 366 of file ihessNumber.h.

9.1.4.283 ihessNumber coco::operator/ (const interval & *a,* const ihessNumber & *b*)

Definition at line 70 of file ihessNumber.cc.

9.1.4.284 diffNumber coco::operator/ (const double & *a,* const diffNumber & *b*)

Definition at line 70 of file diffNumber.cc.

9.1.4.285 interval_set coco::operator/ (const interval_set & *a,* const interval_set & *b*) [inline]

Definition at line 458 of file interval_set.h.

9.1.4.286 interval_set coco::operator/ (const interval_set & *a,* const interval & *b*) [inline]

Definition at line 466 of file interval_set.h.

9.1.4.287 interval_set coco::operator/ (const interval & *a,* const interval_set & *b*) [inline]

Definition at line 473 of file interval_set.h.

9.1.4.288 interval_set coco::operator/ (const interval_set & *a,* double *b*) [inline]

Definition at line 480 of file interval_set.h.

9.1.4.289 interval_set coco::operator/ (double *a*, const interval_set & *b*) [inline]

Definition at line 487 of file interval_set.h.

9.1.4.290 diffI coco::operator/ (const diffI & *a*, const double & *b*)

Definition at line 357 of file diffI.cc.

9.1.4.291 diffI coco::operator/ (const diffI & *a*, const interval & *b*)

Definition at line 402 of file diffI.cc.

9.1.4.292 diffI coco::operator/ (const diffI & *a*, const diffI & *b*)

Definition at line 520 of file diffI.cc.

9.1.4.293 interval coco::operator/ (const interval & *a*, const interval & *b*)

This operator computes a/b as interval.

9.1.4.294 interval coco::operator/ (const interval & *a*, double *b*)

9.1.4.295 interval coco::operator/ (double *b*, const interval & *a*)

9.1.4.296 diffI coco::operator/ (const double & *a*, const diffI & *b*)

Definition at line 544 of file diffI.cc.

9.1.4.297 diffI coco::operator/ (const interval & *a*, const diffI & *b*)

Definition at line 564 of file diffI.cc.

9.1.4.298 template<class I > projective_interval<I> coco::operator/ (const projective_interval< I > & *a*, const projective_interval< I > & *b*)

Definition at line 844 of file prointerval.hpp.

9.1.4.299 template<class I > projective_interval<I> coco::operator/ (const projective_interval< I > & *a*, double *b*)

Definition at line 854 of file prointerval.hpp.

9.1.4.300 template<class I > projective_interval<I> coco::operator/ (double *b*, const projective_interval< I > & *a*)

Definition at line 862 of file prointerval.hpp.

9.1.4.301 bool coco::operator< (int *n*, const proj_rational & *R*) [inline]

Definition at line 114 of file prointerval.h.

9.1.4.302 bool coco::operator< (const interval & *a,* const interval & *b*) [inline]

This comparison operator returns the "surely less than interval relation", i.e. it is true iff `T < X == T.sup() < X.inf()`.

This comparison operator returns the "sure less than interval relation", i.e. it is true iff `T.sgt(X) == T.inf() > X.inf() && T.sup() > X.sup()`.

Definition at line 522 of file interval_boost.h.

9.1.4.303 bool coco::operator< (const interval & *a,* double *b*) [inline]

Definition at line 523 of file interval_boost.h.

9.1.4.304 bool coco::operator< (double *a,* const interval & *b*) [inline]

Definition at line 524 of file interval_boost.h.

9.1.4.305 std::ostream & coco::operator<< (std::ostream & *os,* const basic_alltype & *b*)

Definition at line 50 of file basic_alltype.cc.

9.1.4.306 std::ostream& coco::operator<< (std::ostream & *o,* const func_islp2_eval_ret_type & *r*) [inline]

Definition at line 158 of file islp2_evaluator.h.

9.1.4.307 std::ostream& coco::operator<< (std::ostream & *o,* const expression_node & *__x*)

Definition at line 208 of file expression.cc.

9.1.4.308 std::ostream & coco::operator<< (std::ostream & *s,* const hessNumber & *a*)

Definition at line 278 of file hessNumber.cc.

9.1.4.309 std::ostream & coco::operator<< (std::ostream & *s,* const ihessNumber & *a*)

Definition at line 310 of file ihessNumber.cc.

9.1.4.310 std::ostream & coco::operator<< (std::ostream & *s,* const diffNumber & *a*)

Definition at line 482 of file diffNumber.cc.

9.1.4.311 std::ostream & coco::operator<< (std::ostream & *s,* const interval & *a*) [inline]

Definition at line 1070 of file interval_filib.h.

9.1.4.312 std::ostream & coco::operator<< (std::ostream & *s,* const diffI & *a*)

Definition at line 1020 of file diffI.cc.

9.1.4.313 bool coco::operator<= (int *n,* const proj_rational & *R*) [inline]

Definition at line 115 of file prointerval.h.

9.1.4.314 bool coco::operator<= (const interval & *a,* const interval & *b*) [inline]

This comparison operator returns the "surely less than or equal to interval relation", i.e. it is true iff `T <= X == T.sup() <= X.inf()`.

Definition at line 670 of file interval_filib.h.

9.1.4.315 bool coco::operator<= (const interval & *a,* double *b*) [inline]

This comparison operator returns the "surely less than or equal to interval relation", i.e. it is true iff `T <= X == T.sup() <= X.inf()`.

Definition at line 671 of file interval_filib.h.

9.1.4.316 bool coco::operator<= (double *a,* const interval & *b*) [inline]

This comparison operator returns the "surely less than or equal to interval relation", i.e. it is true iff `T <= X == T.sup() <= X.inf()`.

Definition at line 672 of file interval_filib.h.

9.1.4.317 bool coco::operator== (int *n,* const proj_rational & *R*) [inline]

Definition at line 118 of file prointerval.h.

9.1.4.318 template<class _TC > bool coco::operator== (const interval & *__i,* const _TC & *__d*) [inline]

Definition at line 814 of file interval_boost.h.

9.1.4.319 bool coco::operator== (const interval & *__i,* const interval & *__d*) [inline]

Comparison operator (equal)

Definition at line 514 of file interval_boost.h.

9.1.4.320 bool coco::operator== (const interval & *a,* double *b*) [inline]

Definition at line 516 of file interval_boost.h.

9.1.4.321 bool coco::operator== (const interval_set & *a,* const interval_set & *b*)

Definition at line 671 of file interval_set.cc.

9.1.4.322 bool coco::operator> (int *n,* const proj_rational & *R*) [inline]

Definition at line 116 of file prointerval.h.

9.1.4.323 bool coco::operator> (const interval & *a,* const interval & *b*) [inline]

This comparison operator returns the "surely greater than interval relation", i.e. it is true iff `T > X == T.inf() > X.sup()`.

Definition at line 661 of file interval_filib.h.

9.1.4.324 bool coco::operator> (const interval & *a,* double *b*) [inline]

This comparison operator returns the "surely greater than interval relation", i.e. it is true iff `T > X == T.inf() > X.sup()`.

Definition at line 662 of file interval_filib.h.

9.1.4.325 bool coco::operator> (double *a,* const interval & *b*) [inline]

This comparison operator returns the "surely greater than interval relation", i.e. it is true iff `T > X == T.inf() > X.sup()`.

Definition at line 663 of file interval_filib.h.

9.1.4.326 bool coco::operator>= (int *n,* const proj_rational & *R*) [inline]

Definition at line 117 of file prointerval.h.

9.1.4.327 bool coco::operator>= (const interval & *a,* const interval & *b*) [inline]

This comparison operator returns the "surely greater than or equal to interval relation", i.e. it is true iff `T >= X == T.inf() >= X.sup()`.

Definition at line 679 of file interval_filib.h.

9.1.4.328 bool coco::operator>= (const interval & *a,* double *b*) [inline]

This comparison operator returns the "surely greater than or equal to interval relation", i.e. it is true iff `T >= X == T.inf() >= X.sup()`.

Definition at line 680 of file interval_filib.h.

9.1.4.329 bool coco::operator>= (double *a,* const interval & *b*) [inline]

This comparison operator returns the "surely greater than or equal to interval relation", i.e. it is true iff `T >= X == T.inf() >= X.sup()`.

Definition at line 681 of file interval_filib.h.

9.1.4.330 void coco::parse_dbcompare_expression (const std::string & *s,* std::map< int, std::triple< std::string, dbc_method, basic_alltype > > & *cols,* std::vector< std::vector< int > > & *expr*)

Definition at line 386 of file dbcompare.cc.

9.1.4.331 void coco::parse_dbcompare_sortorder (const std::string & *s,* std::map< int, std::triple< std::string, dbc_method, basic_alltype > > & *cols,* std::vector< int > & *order*)

Definition at line 399 of file dbcompare.cc.

9.1.4.332 hessNumber coco::pow (const hessNumber & *a,* const hessNumber & *b*)

Definition at line 163 of file hessNumber.cc.

9.1.4.333 ihessNumber coco::pow (const ihessNumber & *a*, const ihessNumber & *b*)

Definition at line 182 of file ihessNumber.cc.

9.1.4.334 diffNumber coco::pow (const diffNumber & *a*, const diffNumber & *b*)

Definition at line 192 of file diffNumber.cc.

9.1.4.335 template<class I > I coco::pow (const I & *i*, const proj_rational & *r*) [inline]

Definition at line 540 of file prointerval.h.

9.1.4.336 interval coco::pow (const interval & *x*, const interval & *y*)

Returns an interval enclosure of x^y = exp(y∗log(x)).

9.1.4.337 diffI coco::pow (const diffI & *a*, const diffI & *b*)

Definition at line 750 of file diffI.cc.

9.1.4.338 interval_set coco::pow (const interval_set & *a*, const interval_set & *b*)

Definition at line 1051 of file interval_set.cc.

9.1.4.339 template<class I > projective_interval<I> coco::pow (const projective_interval< I > & *x*, const projective_interval< I > & *y*)

Definition at line 1067 of file prointerval.hpp.

9.1.4.340 hessNumber coco::power (const hessNumber & *a*, int *n*)

Definition at line 126 of file hessNumber.cc.

9.1.4.341 ihessNumber coco::power (const ihessNumber & *a*, int *n*)

Definition at line 145 of file ihessNumber.cc.

9.1.4.342 diffNumber coco::power (const diffNumber & *a*, int *n*)

Definition at line 172 of file diffNumber.cc.

9.1.4.343 interval coco::power (const interval & *x*, int *n*)

Returns an interval enclosure of x^n.

9.1.4.344 diffI coco::power (const diffI & *a*, int *n*)

Definition at line 702 of file diffI.cc.

9.1.4.345 interval_set coco::power (const interval_set & *a*, int *n*)

Definition at line 715 of file interval_set.cc.

9.1.4.346 template<class I > projective_interval<I> coco::power (const projective_interval< I > & *x*, int *n*)

Definition at line 1058 of file prointerval.hpp.

9.1.4.347 double coco::r_random_h_eval (double *t*, double *b*, double *a*) [inline]

Definition at line 123 of file coconut_random.h.

9.1.4.348 double coco::r_random_hinv_eval (double *x*, double *b*, double *a*) [inline]

Definition at line 136 of file coconut_random.h.

9.1.4.349 double coco::rad (const interval &)

rad(a) == a.rad()

9.1.4.350 double coco::relDiam (const interval &)

relDiam(a) == a.relDiam()

9.1.4.351 interval coco::round_to_integer (const interval & *x*)

This function rounds the interval inward to integer borders.

9.1.4.352 interval_set coco::round_to_integer (const interval_set & *a*)

This function rounds the interval inward to integer borders.

Definition at line 1081 of file interval_set.cc.

9.1.4.353 double coco::safeguarded_mid (const interval & *__i*) [inline]

This function computes the safeguarded midpoint of `__i`.

Definition at line 492 of file interval_boost.h.

9.1.4.354 hessNumber coco::sin (const hessNumber & *a*)

Definition at line 80 of file hessNumber.cc.

9.1.4.355 ihessNumber coco::sin (const ihessNumber & *a*)

Definition at line 95 of file ihessNumber.cc.

9.1.4.356 diffNumber coco::sin (const diffNumber & *a*)

Definition at line 96 of file diffNumber.cc.

9.1.4.357 diffI coco::sin (const diffI & *a*)

Definition at line 603 of file diffI.cc.

9.1.4.358 interval coco::sin (const interval & *x*)

Returns an interval enclosure of the sine of the interval x.

9.1.4.359 interval_set coco::sin (const interval_set & *a*)

Definition at line 859 of file interval_set.cc.

9.1.4.360 template<class I > projective_interval<I> coco::sin (const projective_interval< I > & *x*)

Definition at line 1079 of file prointerval.hpp.

9.1.4.361 hessNumber coco::sinh (const hessNumber & *a*)

Definition at line 189 of file hessNumber.cc.

9.1.4.362 ihessNumber coco::sinh (const ihessNumber & *a*)

Definition at line 242 of file ihessNumber.cc.

9.1.4.363 diffNumber coco::sinh (const diffNumber & *a*)

Definition at line 351 of file diffNumber.cc.

9.1.4.364 interval coco::sinh (const interval & *x*)

Returns an interval enclosure of the hyperbolic sine of the interval x.

9.1.4.365 diffI coco::sinh (const diffI & *a*)

Definition at line 825 of file diffI.cc.

9.1.4.366 interval_set coco::sinh (const interval_set & *a*)

Definition at line 955 of file interval_set.cc.

9.1.4.367 template<class I > projective_interval<I> coco::sinh (const projective_interval< I > & *x*)

Definition at line 1088 of file prointerval.hpp.

9.1.4.368 void coco::sparsity (const model & *DAG*, int & *nnz_H*, int *& *H_ridx*, int *& *H_cidx*, int & *nnz_J*, int *& *J_funidx*, int *& *J_varidx*, bool *J_with_obj*, bool *J_obj_is_first*, bool *org_only*, bool *H_upper_only*, sp_indexing *indexing*, bool *fun_one_based*, bool *var_one_based*)

Definition at line 712 of file sparsity.cc.

9.1.4.369 void coco::sparsity (const model & *DAG*, unsigned int & *nnz_H*, unsigned int *& *H_ridx*, unsigned int *& *H_cidx*, bool *org_only*, bool *H_upper_only*, sp_indexing *indexing*, bool *fun_one_based*, bool *var_one_based*)

Definition at line 725 of file sparsity.cc.

9.1.4.370 void coco::sparsity (const model & *DAG*, int & *nnz_J*, int *& *J_funidx*, int *& *J_varidx*, bool *J_with_obj*, bool *J_obj_is_first*, bool *org_only*, sp_indexing *indexing*, bool *fun_one_based*, bool *var_one_based*)

Definition at line 780 of file sparsity.cc.

9.1.4.371 ihessNumber coco::sqr (const ihessNumber & *a*)

Definition at line 294 of file ihessNumber.cc.

9.1.4.372 interval coco::sqr (const interval & *x*)

Returns an interval enclosure of x^2.

9.1.4.373 interval_set coco::sqr (const interval_set & *a*)

Definition at line 727 of file interval_set.cc.

9.1.4.374 diffI coco::sqr (const diffI & *a*)

Definition at line 911 of file diffI.cc.

9.1.4.375 template<class I > projective_interval<I> coco::sqr (const projective_interval< I > & *x*)

Definition at line 1097 of file prointerval.hpp.

9.1.4.376 hessNumber coco::sqrt (const hessNumber & *a*)

Definition at line 104 of file hessNumber.cc.

9.1.4.377 ihessNumber coco::sqrt (const ihessNumber & *a*)

Definition at line 119 of file ihessNumber.cc.

9.1.4.378 diffNumber coco::sqrt (const diffNumber & *a*)

Definition at line 126 of file diffNumber.cc.

9.1.4.379 diffI coco::sqrt (const diffI & *a*)

Definition at line 656 of file diffI.cc.

9.1.4.380 interval coco::sqrt (const interval & *x*)

Returns an interval enclosure of the square root of the interval x.

9.1.4.381 interval_set coco::sqrt (const interval_set & *a*)

Definition at line 739 of file interval_set.cc.

9.1.4.382 template<class I > projective_interval<I> coco::sqrt (const projective_interval< I > & *x*)

Definition at line 1106 of file prointerval.hpp.

9.1.4.383 hessNumber coco::square (const hessNumber & *a*)

Definition at line 242 of file hessNumber.cc.

9.1.4.384 diffNumber coco::square (const diffNumber & *a*)

Definition at line 414 of file diffNumber.cc.

9.1.4.385 std::vector<std::vector<void *> > coco::standard_evaluators (NUM_EVALUATORS)

This variables contains pointers to all standard evaluation routines for user defined nodes.

9.1.4.386 interval coco::tan (const interval & *x*)

Returns an interval enclosure of the tangent of the interval x.

9.1.4.387 interval_set coco::tan (const interval_set & *a*)

Definition at line 895 of file interval_set.cc.

9.1.4.388 template<class I > projective_interval<I> coco::tan (const projective_interval< I > & *x*)

Definition at line 1115 of file prointerval.hpp.

9.1.4.389 interval coco::tanh (const interval & *x*)

Returns an interval enclosure of the hyperbolic tangent of the interval x.

9.1.4.390 interval_set coco::tanh (const interval_set & *a*)

Definition at line 991 of file interval_set.cc.

9.1.4.391 template<class I > projective_interval<I> coco::tanh (const projective_interval< I > & *x*)

Definition at line 1124 of file prointerval.hpp.

9.1.4.392 double coco::upward_divides (const double & *a*, const double & *b*)

This function divides `a` by `b`, rounding upwards.

9.1.4.393 double coco::upward_minus (const double & *a*, const double & *b*)

This function subtracts `b` from `a`, rounding upwards.

9.1.4.394 double coco::upward_multiplies (const double & *a*, const double & *b*)

This function multiplies `a` by `b`, rounding upwards.

9.1.4.395 double coco::upward_plus (const double & *a*, const double & *b*)

This function adds `a` and `b`, rounding upwards.

9.1.4.396 double coco::volume (const interval_set & *a*)

Definition at line 657 of file interval_set.cc.

9.1.4.397 double coco::width (const interval &)

9.1.4.398 double coco::width (const interval_set & *a*)

Definition at line 656 of file interval_set.cc.

9.1.5 Variable Documentation

9.1.5.1 const char* coco::correxpr_names[ ] = {"log10", "asin", "acos", "atan", "tan", "<=", "<", ">=", ">", "==", "!="}

Definition at line 662 of file model.cc.

9.1.5.2 std::vector<std::vector<void *> > coco::standard_evaluators

9.1.5.3 const int coco::STANDARD_MAXDEG_DIFFINTERVAL = 4

Definition at line 12 of file diffI.h.

9.1.5.4 const int coco::STANDARD_MAXDEG_DIFFNUMBER = 4

Definition at line 12 of file diffNumber.h.

9.2 coco::coco Namespace Reference

Namespaces

- namespace coco
- namespace num

Classes

- struct checking_my
- struct my_rounded_math
- struct interval_st
- class interval
- class api_exception

  API exception class.
- class nyi_exception

  Not Yet Implemented exception class.
- class basic_alltype

  The basic alltype which can hold any of a number of basic types.
- class certificate

  The certificate class (certifies deltas for rigorous mode operation)

- class certificate_base

  Base class for the certificates.
- class delta

  The delta class (updates to work nodes)
- class delta_base

  Base class for the deltas.
- class undelta

  The undelta class (undo of updates to work nodes)
- class undelta_base

  Base class for the undeltas.
- class convex_e

  Convexity information.
- class semantics

  Expression Semantics.
- class variable_indicator

  Bitmap class used to indicate variable occurrence.
- class _evaluator_base

  Base class of all evaluators.
- class evaluator_base

  Base class of all (non-caching) evaluators.
- class cached_evaluator_base

  Base class of all caching evaluators.
- class forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class backward_evaluator_base

  Base class of all (non-caching) backward evaluators.
- class cached_forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class cached_backward_evaluator_base

  Base class of all caching backward evaluators.
- class expression_node

  The base class for a node in the expression DAGs.
- class annotation

  Annotations for Models.
- class model

  The model class (an attributed DAG of expression nodes, lowest class in the model hierarchy)
- class model_iddata

  The model id-data class (the topmost in the model class hierarchy)
- class model_gid

  Model Group Data Class (middle class in the model hierarchy)
- class datamap

  The base class for communicating with COCONUT modules.
- class inference_module_cache_autogen

  Inference module cache auto-generate method base class.
- class inference_module_cache

  Inference module cache class.

- class search_node

  Base type of the nodes in the search graph.
- class delta_node

  Class holding the delta nodes in the search graph.
- class full_node

  Class holding the full nodes in the search graph.
- class work_node

  Work node, which is passed to the inference engines.
- class dbt_row

  This type is used to hold one row in some table of the search database.
- class work_node_context

  The evaluation context when retrieving from the search database.
- class point_check_feasibility

  Stored procedure checking the feasibility of a point.
- class box_check_intersection

  Stored procedure checking whether a box intersects the work_node's box.
- class delta_get_action

  Stored procedure class for computing the delta action specifier.

Typedefs

- typedef boost::numeric::interval_lib::policies < my_rounded_math< double > , checking_my< double >> my_policies
- typedef vdbl::rowid delta_id

  The class for delta ids.
- typedef enum coco::coco::tristate_e tristate
- typedef uint32_t search_node_id

  Type of the unique search node identifier.
- typedef interval rhs_t
- typedef std::vector< void ∗ > evaluator_v
- typedef model::walker expression_walker

  Walker to the expression DAG.
- typedef model::const_walker expression_const_walker

  Const walker to the expression DAG.

Enumerations

- enum api_exception_type { apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph =

5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14 }

Enum for classifying api_exceptions.

- enum tristate_e { t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0 }
- enum tristate_e { t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0 }
- enum convex_info { c_convex = 1, c_linear = 0, c_concave = -1, c_maybe = 2, c_not = -2, c_constant = 3 }

  Convexity information enum.
- enum type_annotation { v_exists = 0, v_forall = 1, v_free = 2, v_stochastic = 3 }

  Node type information enum.
- enum activity_descr { a_redundant = 1, a_active_lo = 2, a_active_lo_red = a_active_lo|a_redundant, a_active_hi = 4, a_active_hi_red = a_active_hi|a_redundant, a_active = a_active_lo|a_active_hi, a_active_red = a_active|a_redundant }

  Constraint activity information enum.
- enum api_exception_type { apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14 }

  Enum for classifying api_exceptions.
- enum e_expression_type { ex_bound = 1, ex_linear = 1<<1, ex_quadratic = 1<<2, ex_polynomial = 1<<3, ex_other = 1<<4, ex_kj = 1<<7, ex_org = 1<<8, ex_redundant = 1<<9, ex_notredundant = 1<<10, ex_active_lo = 1<<11, ex_inactive_lo = 1<<12, ex_active_hi = 1<<13, ex_inactive_hi = 1<<14, ex_active = ex_active_lo|ex_active_hi, ex_inactive = ex_inactive_lo|ex_inactive_hi, ex_integer = 1<<15, ex_exists = 1<<16, ex_forall = 1<<17, ex_free = 1<<18, ex_stochastic = 1<<19, ex_convex = 1<<20, ex_concave = 1<<21, ex_inequality = 1<<28, ex_equality = 1<<29, ex_leftbound = 1<<30, ex_rightbound = 1<<31, ex_atmlin = ex_bound|ex_linear, ex_atmquad = ex_atmlin|ex_quadratic, ex_atmpoly = ex_atmquad|ex_polynomial, ex_nonlin = ex_quadratic|ex_polynomial|ex_other, ex_nonbnd = ex_linear|ex_nonlin, ex_any = ex_atmlin|ex_nonlin, ex_bothbound = ex_leftbound|ex_rightbound }
- enum tristate_e { t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0 }

- enum api_exception_type { apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14 }

  Enum for classifying api_exceptions.
- enum tristate_e { t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0, t_true = 1, t_false = -1, t_maybe = 0 }
- enum api_exception_type { apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14 }

  Enum for classifying api_exceptions.
- enum api_exception_type { apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14, apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_-

graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14 }

Enum for classifying api_exceptions.

- enum search_node_relation { snr_root, snr_reduction, snr_relaxation, snr_split, snr_glue, snr_worknode, snr_virtual }

  Enum specifying node relations in the search graph.

Functions

- double safeguarded_mid (const interval &__i)
- interval ipow (const interval &x, int n)
- interval gauss (const interval &x)
- interval atan2 (const interval &y, const interval &x)
- double absmin (const interval &__i)
- double gainfactor (const interval &_old, const interval &_new)
- template<class _TC >
  bool operator== (const interval &__i, const _TC &__d)
- template<class _TC >
  bool operator!= (const interval &__i, const _TC &__d)
- bool operator== (const interval &a, double b)
- bool operator!= (const interval &a, double b)
- bool operator< (const interval &a, double b)
- bool operator< (double a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- double mid (const interval &)
- double diam (const interval &)
- double width (const interval &)
- double relDiam (const interval &)
- double rad (const interval &)
- double mig (const interval &)
- double mag (const interval &)
- double dist (const interval &x, const interval &y)
- interval round_to_integer (const interval &x)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)

- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part2 (const interval &x, const interval &y, bool b=true)
- template<class _TH >
  std::string convert_to_str (const _TH &_h)

  Convert an object of any printable class to a string.
- std::string i2a (int x)

  converter int -> string
- std::string localdatetime ()

  converter date & time into std::string
- std::string datetime (const struct tm *timeptr)

  local time as C++ string
- std::string e2D (double d)

  Fortran real as C++ string.
- const basic_alltype & basic_alltype_empty ()
- std::ostream & operator<< (std::ostream &os, const basic_alltype &b)
- bool less_than (const basic_alltype &a, const basic_alltype &b)
- bool less_equal (const basic_alltype &a, const basic_alltype &b)
- std::ostream & operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.
- std::ostream & operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.
- bool operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.
- bool operator== (const convex_e &__c, const convex_info &__d)
- bool operator== (const convex_info &__c, const convex_e &__d)
- bool operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.

- bool operator!= (const convex_e &__c, const convex_info &__d)
- bool operator!= (const convex_info &__c, const convex_e &__d)
- std::ostream & operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.
- std::ostream & operator<< (std::ostream &o, const semantics &__s)

  C++ stream output operator for semantics.

- bool operator== (const interval &a, const interval &b)

- bool operator!= (const interval &a, const interval &b)

- bool operator< (const interval &a, const interval &b)

- interval operator+ (const interval &a, const interval &b)

- interval operator- (const interval &a, const interval &b)

- interval operator* (const interval &a, const interval &b)

- interval operator/ (const interval &a, const interval &b)

- interval division_part1 (const interval &x, const interval &y, bool &b)

9.2.1 Typedef Documentation

9.2.1.1 typedef std::vector<void*> coco::coco::evaluator_v

This type defines the type of the standard evaluators for user defined nodes.

Definition at line 462 of file search_graph.cc.

9.2.1.2 typedef boost::numeric::interval_lib::policies< my_rounded_math<double>, checking_my<double>> coco::coco::my_policies

Definition at line 101 of file expression.h.

9.2.1.3 typedef interval coco::coco::rhs_t

This type is the allowed type for constraint restrictions.

Definition at line 459 of file search_graph.cc.

9.2.1.4 typedef enum coco::coco::tristate_e coco::coco::tristate

The tristate type is a boolean with the addition of a maybe value

9.2.2 Enumeration Type Documentation

9.2.2.1 enum coco::coco::activity_descr

This enum describes the activity state known about a constraint and whether it is redundant or not. The meaning of the enum entries is:

- `a_redundant`: The constraint is known to be redundant. There can be two reasons for that. It can be inactive on both sides, or it was constructed, knowing that it would be redundant (e.g. cuts).
- `a_active_lo`: The `work_node` may contain points, for which this constraint is active at the lower bound but no points, for which the constraint is active at the upper bound.
- `a_active_hi`: The `work_node` may contain points, for which this constraint is active at the upper bound but no points, for which the constraint is active at the lower bound.
- `a_active`: The `work_node` may contain points, for which this constraint is active at the upper bound and points, for which the constraint is active at the lower bound.
- ..._red: A combination with `_red` means that although the inactivity of the constraint could not be proved, it is still known to be redundant.

Enumerator:

a_redundant

a_active_lo

a_active_lo_red

a_active_hi

a_active_hi_red

a_active

a_active_red

Definition at line 81 of file search_graph.cc.

9.2.2.2 enum coco::coco::convex_info

This enum is used to store information about the convexity of a node or an expression.

Enumerator:

c_convex The expression is convex and not linear

c_linear The expression is linear

c_concave The expression is concave and not linear

c_maybe The convexity type of the expression is unknown

c_not The expression is neither convex nor concave

c_constant The expression is constant

Definition at line 45 of file search_graph.cc.

9.2.2.3 enum coco::coco::e_expression_type

This enum is used for determining the type of the expression in the `is` method and similar methods and operators

Enumerator:

ex_bound a variable

ex_linear linear (but not a single variable)

ex_quadratic quadratic non-linear

ex_polynomial non-quadratic polynomial

ex_other non-polynomial non-linear

ex_kj only additional vars, constr. from KJ-conditions

ex_org original vars, constr. (ex_kj, ex_org both set = none set)

ex_redundant this constraint is known to be redundant

ex_notredundant this constraint is known to be not redundant

ex_active_lo this constraint is possible active at the lower bound

ex_inactive_lo this constraint is inactive at the lower bound

ex_active_hi this constraint is possible active at the upper bound

ex_inactive_hi this constraint is inactive at the upper bound

ex_active this constraint is possible active

ex_inactive this constraint is inactive

ex_integer this expression is an integer expression

ex_exists this variable has an exist-context

ex_forall this variable has a for-all-context

ex_free this variable is free

ex_stochastic this variable is a stochastic variable

ex_convex this expression is known to be convex

ex_concave this expression is known to be concave

ex_inequality this is an inequality constraint

ex_equality this is an equality constraint

ex_leftbound this is a constraint with lower bound

ex_rightbound this is a constraint with upper bound

ex_atmlin at most linear

ex_atmquad at most quadratic

ex_atmpoly at most polynomial

ex_nonlin non linear
ex_nonbnd not bounds
ex_any any
ex_bothbound this is a two-sided constraint

Definition at line 402 of file search_graph.cc.

9.2.2.4 enum coco::coco::tristate_e

The tristate type is a boolean with the addition of a maybe value

Enumerator:

t_true
t_false
t_maybe
t_true
t_false
t_maybe
t_true
t_false
t_maybe
t_true
t_false
t_maybe

Definition at line 37 of file search_graph.cc.

9.2.2.5 enum coco::coco::tristate_e

The tristate type is a boolean with the addition of a maybe value

Enumerator:

t_true
t_false
t_maybe
t_true
t_false
t_maybe
t_true
t_false
t_maybe
t_true
t_false
t_maybe

Definition at line 37 of file search_graph.cc.

9.2.2.6 enum coco::coco::tristate_e

The tristate type is a boolean with the addition of a maybe value

Enumerator:

t_true
t_false
t_maybe
t_true
t_false
t_maybe
t_true
t_false
t_maybe
t_true
t_false
t_maybe

Definition at line 37 of file search_graph.cc.

9.2.2.7 enum coco::coco::tristate_e

The tristate type is a boolean with the addition of a maybe value

Enumerator:

t_true
t_false
t_maybe
t_true
t_false
t_maybe
t_true
t_false
t_maybe
t_true
t_false
t_maybe

Definition at line 37 of file search_graph.cc.

9.2.2.8 enum coco::coco::type_annotation

This enum is used to store information about the type of a node or an expression.

Enumerator:

v_exists The expression/variable is interpreted in a exists context.
v_forall The expression/variable is interpreted in a forall context.
v_free This is valid for variables only, and it defines a free variable.
v_stochastic This expression is stochastic.

Definition at line 46 of file search_graph.cc.

9.2.3 Function Documentation

9.2.3.1 interval coco::coco::abs (const interval & *x*)

Returns an interval enclosure of the absolute value of the interval x.

9.2.3.2 double coco::coco::absmin (const interval & *__i*) [inline]

This function returns the element of `__i` with smallest absolute value.

Definition at line 785 of file expression.h.

9.2.3.3 interval coco::coco::acos (const interval & *x*)

Returns an interval enclosure of the inverse cosine of the interval x.

9.2.3.4 interval coco::coco::acosh (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

9.2.3.5 interval coco::coco::acot (const interval & *x*)

Returns an interval enclosure of the inverse cotangent of the interval x.

9.2.3.6 interval coco::coco::acoth (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

9.2.3.7 interval coco::coco::asin (const interval & *x*)

Returns an interval enclosure of the inverse sine of the interval x.

9.2.3.8 interval coco::coco::asinh (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

9.2.3.9 interval coco::coco::atan (const interval & *x*)

Returns an interval enclosure of the inverse tangent of the interval x.

9.2.3.10 interval coco::coco::atan2 (const interval & *y*, const interval & *x*) [inline]

This function computes the atan2 function $\mathrm{atan}(y/x)$ for intervals.

Definition at line 778 of file expression.h.

9.2.3.11 interval coco::coco::atanh (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

9.2.3.12 const basic_alltype& coco::coco::basic_alltype_empty ( )

Definition at line 44 of file basic_alltype.cc.

9.2.3.13 interval coco::coco::cancel (const interval & *a*, const interval & *b*)

9.2.3.14 interval coco::coco::cos (const interval & *x*)

Returns an interval enclosure of the cosine of the interval x.

9.2.3.15 interval coco::coco::cosh (const interval & *x*)

Returns an interval enclosure of the hyperbolic cosine of the interval x.

9.2.3.16 interval coco::coco::cot (const interval & *x*)

Returns an interval enclosure of the cotangent of the interval x.

9.2.3.17 interval coco::coco::coth (const interval & *x*)

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

9.2.3.18 std::string coco::coco::datetime (const struct tm * *timeptr*)

This function returns the local time as C++ string.

Definition at line 46 of file writer_utils.cc.

9.2.3.19 double coco::coco::diam (const interval &)

diam(a) == a.diam()

9.2.3.20 double coco::coco::dist (const interval & *x*, const interval & *y*)

Same as x.dist(y)

9.2.3.21 interval coco::coco::division_part1 (const interval & *x*, const interval & *y*, bool & *b*)

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`;

a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

9.2.3.22 interval coco::coco::division_part2 (const interval & *x*, const interval & *y*, bool *b* = `true`)

9.2.3.23 std::string coco::coco::e2D (double *d*)

This function converts a double number to a double precision Fortran representation in a C++ string.

Definition at line 78 of file writer_utils.cc.

9.2.3.24 interval coco::coco::exp (const interval & *x*)

Returns an interval enclosure of the exponential of the interval x.

9.2.3.25 interval coco::coco::exp10 (const interval & *x*)

Returns an interval enclosure of the exponential (base 10) of the interval x.

9.2.3.26 interval coco::coco::exp2 (const interval & *x*)

Returns an interval enclosure of the exponential (base 2) of the interval x.

9.2.3.27 interval coco::coco::expm1 (const interval & *x*)

Returns an interval enclosure of exp(x)-1

9.2.3.28 double coco::coco::gainfactor (const interval & *_old*, const interval & *_new*) `[inline]`

The call `gainfactor(o,n)` computes a safeguarded quotient of the widths of @ o and @ n under the assumption that `n` is a subset of `o`.

Definition at line 800 of file expression.h.

9.2.3.29 interval coco::coco::gauss (const interval & *x*) `[inline]`

This function computes the gaussian function e^{-x^2} for intervals.

Definition at line 771 of file expression.h.

9.2.3.30 std::string coco::coco::i2a (int *x*)

This function converts integers to a C++ string.

Definition at line 37 of file writer_utils.cc.

9.2.3.31 interval coco::coco::imax (const interval & *x*, const interval & *y*)

Returns an interval enclosure of the maximum of two intervals x and y.

9.2.3.32 interval coco::coco::imin (const interval & *x*, const interval & *y*)

Returns an interval enclosure of the minimum of two intervals x and y.

9.2.3.33 interval coco::coco::ipow (const interval & *x*, int *n*) [inline]

This function computes the integer power operation x^n for intervals.

Definition at line 764 of file expression.h.

9.2.3.34 bool coco::coco::less_equal (const basic_alltype & *a*, const basic_alltype & *b*)

Total order comparison operator

9.2.3.35 bool coco::coco::less_than (const basic_alltype & *a*, const basic_alltype & *b*)

Total order comparison operator

9.2.3.36 std::string coco::coco::localdatetime ( )

This function converts a date/time structure to a C++ string.

Definition at line 68 of file writer_utils.cc.

9.2.3.37 interval coco::coco::log (const interval & *x*)

Returns an interval enclosure of the natural logarithm of the interval x.

9.2.3.38 interval coco::coco::log10 (const interval & *x*)

Returns an interval enclosure of the logarithm (base 10) of the interval x.

9.2.3.39 interval coco::coco::log1p (const interval & *x*)

Returns an interval enclosure of log(1+x).

9.2.3.40 interval coco::coco::log2 (const interval & *x*)

Returns an interval enclosure of the logarithm (base 2) of the interval x.

9.2.3.41 double coco::coco::mag (const interval &)

mag(a) == a.mag()

9.2.3.42 double coco::coco::mid (const interval &)

mid(a) == a.mid()

9.2.3.43 double coco::coco::mig (const interval &)

mig(a) == a.mig()

9.2.3.44 template<class _TC > bool coco::coco::operator!= (const interval & *__i*, const _TC & *__d*) [inline]

Definition at line 823 of file expression.h.

9.2.3.45 bool coco::coco::operator!= (const interval & __i, const interval & __d) [inline]

Comparison operator (not equal)

Definition at line 519 of file expression.h.

9.2.3.46 bool coco::coco::operator!= (const interval & a, double b) [inline]

Definition at line 521 of file expression.h.

9.2.3.47 interval coco::coco::operator∗ (const interval & a, const interval & b)

This operator computes a∗b as interval.

9.2.3.48 interval coco::coco::operator∗ (const interval & a, double b)

9.2.3.49 interval coco::coco::operator∗ (double b, const interval & a)

9.2.3.50 interval coco::coco::operator+ (const interval & a, const interval & b)

This operator computes a+b as interval.

9.2.3.51 interval coco::coco::operator+ (const interval & a, double b)

9.2.3.52 interval coco::coco::operator+ (double b, const interval & a)

9.2.3.53 interval coco::coco::operator- (const interval & a, const interval & b)

This operator computes a-b as interval.

9.2.3.54 interval coco::coco::operator- (const interval & a, double b)

9.2.3.55 interval coco::coco::operator- (double b, const interval & a)

9.2.3.56 interval coco::coco::operator/ (const interval & a, const interval & b)

This operator computes a/b as interval.

9.2.3.57 interval coco::coco::operator/ (const interval & a, double b)

9.2.3.58 interval coco::coco::operator/ (double b, const interval & a)

9.2.3.59 bool coco::coco::operator< (const interval & a, const interval & b) [inline]

This comparison operator returns the "surely less than interval relation", i.e. it is true iff `T < X == T.sup() < X.inf()`.

This comparison operator returns the "sure less than interval relation", i.e. it is true iff `T.sgt(X) == T.inf() > X.inf() && T.sup() > X.sup()`.

Definition at line 523 of file expression.h.

9.2.3.60 bool coco::coco::operator< (const interval & *a*, double *b*) [inline]

Definition at line 524 of file expression.h.

9.2.3.61 bool coco::coco::operator< (double *a*, const interval & *b*) [inline]

Definition at line 525 of file expression.h.

9.2.3.62 std::ostream & coco::coco::operator<< (std::ostream & *s*, const interval & *a*) [inline]

Definition at line 1070 of file interval_filib.h.

9.2.3.63 std::ostream & coco::coco::operator<< (std::ostream & *os*, const basic_alltype & *b*)

9.2.3.64 template<class _TC > bool coco::coco::operator== (const interval & *__i*, const _TC & *__d*) [inline]

Definition at line 815 of file expression.h.

9.2.3.65 bool coco::coco::operator== (const interval & *__i*, const interval & *__d*) [inline]

friends

Comparison operator (equal)

Definition at line 515 of file expression.h.

9.2.3.66 bool coco::coco::operator== (const interval & *a*, double *b*) [inline]

Definition at line 517 of file expression.h.

9.2.3.67 interval coco::coco::pow (const interval & *x*, const interval & *y*)

Returns an interval enclosure of x^y = exp(y∗log(x)).

9.2.3.68 interval coco::coco::power (const interval & *x*, int *n*)

Returns an interval enclosure of x^n.

9.2.3.69 double coco::coco::rad (const interval &)

rad(a) == a.rad()

9.2.3.70 double coco::coco::relDiam (const interval &)

relDiam(a) == a.relDiam()

9.2.3.71 interval coco::coco::round_to_integer (const interval & *x*)

This function rounds the interval inward to integer borders.

9.2.3.72 double coco::coco::safeguarded_mid (const interval & __i) [inline]

This function computes the safeguarded midpoint of __i.

Definition at line 493 of file expression.h.

9.2.3.73 interval coco::coco::sin (const interval & x)

Returns an interval enclosure of the sine of the interval x.

9.2.3.74 interval coco::coco::sinh (const interval & x)

Returns an interval enclosure of the hyperbolic sine of the interval x.

9.2.3.75 interval coco::coco::sqr (const interval & x)

Returns an interval enclosure of x^2.

9.2.3.76 interval coco::coco::sqrt (const interval & x)

Returns an interval enclosure of the square root of the interval x.

9.2.3.77 interval coco::coco::tan (const interval & x)

Returns an interval enclosure of the tangent of the interval x.

9.2.3.78 interval coco::coco::tanh (const interval & x)

Returns an interval enclosure of the hyperbolic tangent of the interval x.

9.2.3.79 double coco::coco::width (const interval &)

9.3 coco::coco::coco Namespace Reference

Classes

- struct checking_my
- struct my_rounded_math
- struct interval_st
- class interval
- class api_exception

  API exception class.
- class nyi_exception

  Not Yet Implemented exception class.
- class basic_alltype

  The basic alltype which can hold any of a number of basic types.

Typedefs

- typedef boost::numeric::interval_lib::policies < my_rounded_math< double > , checking_my< double >> my_policies

Enumerations

- enum api_exception_type { apiee_internal = 1, apiee_evaluator = 2, apiee_io = 3, apiee_delta = 4, apiee_search_graph = 5, apiee_communication_data = 6, apiee_inference_engine = 7, apiee_graph_analyzer = 8, apiee_management_module = 9, apiee_initializer = 10, apiee_report_module = 11, apiee_oom = 12, apiee_nyi = 13, apiee_other = 14 }

  Enum for classifying api_exceptions.

Functions

- double safeguarded_mid (const interval &__i)
- interval ipow (const interval &x, int n)
- interval gauss (const interval &x)
- interval atan2 (const interval &y, const interval &x)
- double absmin (const interval &__i)
- double gainfactor (const interval &_old, const interval &_new)
- template<class _TC >
  bool operator== (const interval &__i, const _TC &__d)
- template<class _TC >
  bool operator!= (const interval &__i, const _TC &__d)
- bool operator== (const interval &a, const interval &b)
- bool operator== (const interval &a, double b)
- bool operator!= (const interval &a, const interval &b)
- bool operator!= (const interval &a, double b)
- bool operator< (const interval &a, const interval &b)
- bool operator< (const interval &a, double b)
- bool operator< (double a, const interval &b)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- double mid (const interval &)
- double diam (const interval &)
- double width (const interval &)
- double relDiam (const interval &)
- double rad (const interval &)
- double mig (const interval &)
- double mag (const interval &)
- double dist (const interval &x, const interval &y)
- interval round_to_integer (const interval &x)

- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b=true)
- std::string i2a (int x)

  converter int -> string
- std::string localdatetime ()

  converter date & time into std::string
- std::string datetime (const struct tm *timeptr)

  local time as C++ string
- std::string e2D (double d)

  Fortran real as C++ string.
- const basic_alltype & basic_alltype_empty ()
- std::ostream & operator<< (std::ostream &os, const basic_alltype &b)
- bool less_than (const basic_alltype &a, const basic_alltype &b)
- bool less_equal (const basic_alltype &a, const basic_alltype &b)

9.3.1 Typedef Documentation

9.3.1.1 typedef boost::numeric::interval_lib::policies< my_rounded_math<double>, checking_my<double>> coco::coco::coco::my_policies

Definition at line 101 of file search_graph.cc.

9.3.2 Function Documentation

9.3.2.1 interval coco::coco::coco::abs (const interval & *x*)

Returns an interval enclosure of the absolute value of the interval x.

9.3.2.2 double coco::coco::coco::absmin (const interval & *__i*) [inline]

Definition at line 785 of file search_graph.cc.

9.3.2.3 interval coco::coco::coco::acos (const interval & *x*)

Returns an interval enclosure of the inverse cosine of the interval x.

9.3.2.4 interval coco::coco::coco::acosh (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

9.3.2.5 interval coco::coco::coco::acot (const interval & *x*)

Returns an interval enclosure of the inverse cotangent of the interval x.

9.3.2.6 interval coco::coco::coco::acoth (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

9.3.2.7 interval coco::coco::coco::asin (const interval & *x*)

Returns an interval enclosure of the inverse sine of the interval x.

9.3.2.8 interval coco::coco::coco::asinh (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

9.3.2.9 interval coco::coco::coco::atan (const interval & *x*)

Returns an interval enclosure of the inverse tangent of the interval x.

9.3.2.10 interval coco::coco::coco::atan2 (const interval & *y*, const interval & *x*) [inline]

Definition at line 778 of file search_graph.cc.

9.3.2.11 interval coco::coco::coco::atanh (const interval & *x*)

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

9.3.2.12 const basic_alltype& coco::coco::coco::basic_alltype_empty ( )

Definition at line 44 of file basic_alltype.cc.

9.3.2.13 interval coco::coco::coco::cancel (const interval & *a*, const interval & *b*)

9.3.2.14 interval coco::coco::coco::cos (const interval & *x*)

Returns an interval enclosure of the cosine of the interval x.

9.3.2.15 interval coco::coco::coco::cosh (const interval & *x*)

Returns an interval enclosure of the hyperbolic cosine of the interval x.

9.3.2.16 interval coco::coco::coco::cot (const interval & *x*)

Returns an interval enclosure of the cotangent of the interval x.

9.3.2.17 interval coco::coco::coco::coth (const interval & *x*)

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

9.3.2.18 std::string coco::coco::coco::datetime (const struct tm * *timeptr*)

This function returns the local time as C++ string.

Definition at line 46 of file writer_utils.cc.

9.3.2.19 double coco::coco::coco::diam (const interval &)

diam(a) == a.diam()

9.3.2.20 double coco::coco::coco::dist (const interval & *x*, const interval & *y*)

Same as x.dist(y)

9.3.2.21 interval coco::coco::coco::division_part1 (const interval & *x*, const interval & *y*, bool & *b*)

9.3.2.22 interval coco::coco::coco::division_part2 (const interval & *x*, const interval & *y*, bool *b* = `true`)

9.3.2.23 std::string coco::coco::coco::e2D (double *d*)

This function converts a double number to a double precision Fortran representation in a C++ string.

Definition at line 78 of file writer_utils.cc.

9.3.2.24 interval coco::coco::coco::exp (const interval & *x*)

Returns an interval enclosure of the exponential of the interval x.

9.3.2.25 interval coco::coco::coco::exp10 (const interval & *x*)

Returns an interval enclosure of the exponential (base 10) of the interval x.

9.3.2.26 interval coco::coco::coco::exp2 (const interval & *x*)

Returns an interval enclosure of the exponential (base 2) of the interval x.

9.3.2.27 interval coco::coco::coco::expm1 (const interval & *x*)

Returns an interval enclosure of exp(x)-1

9.3.2.28 double coco::coco::coco::gainfactor (const interval & *_old*, const interval & *_new*) [inline]

Definition at line 800 of file search_graph.cc.

9.3.2.29 interval coco::coco::coco::gauss (const interval & *x*) [inline]

Definition at line 771 of file search_graph.cc.

9.3.2.30 std::string coco::coco::coco::i2a (int *x*)

This function converts integers to a C++ string.

Definition at line 37 of file writer_utils.cc.

9.3.2.31 interval coco::coco::coco::imax (const interval & *x*, const interval & *y*)

Returns an interval enclosure of the maximum of two intervals x and y.

9.3.2.32 interval coco::coco::coco::imin (const interval & *x*, const interval & *y*)

Returns an interval enclosure of the minimum of two intervals x and y.

9.3.2.33 interval coco::coco::coco::ipow (const interval & *x*, int *n*) [inline]

Definition at line 764 of file search_graph.cc.

9.3.2.34 bool coco::coco::coco::less_equal (const basic_alltype & *a*, const basic_alltype & *b*)

Total order comparison operator

9.3.2.35 bool coco::coco::coco::less_than (const basic_alltype & *a*, const basic_alltype & *b*)

Total order comparison operator

9.3.2.36 std::string coco::coco::coco::localdatetime ( )

This function converts a date/time structure to a C++ string.

Definition at line 68 of file writer_utils.cc.

9.3.2.37 interval coco::coco::coco::log (const interval & *x*)

Returns an interval enclosure of the natural logarithm of the interval x.

9.3.2.38 interval coco::coco::coco::log10 (const interval & *x*)

Returns an interval enclosure of the logarithm (base 10) of the interval x.

9.3.2.39 interval coco::coco::coco::log1p (const interval & *x*)

Returns an interval enclosure of log(1+x).

9.3.2.40 interval coco::coco::coco::log2 (const interval & *x*)

Returns an interval enclosure of the logarithm (base 2) of the interval x.

9.3.2.41 double coco::coco::coco::mag (const interval &)

mag(a) == a.mag()

9.3.2.42 double coco::coco::coco::mid (const interval &)

mid(a) == a.mid()

9.3.2.43 double coco::coco::coco::mig (const interval &)

mig(a) == a.mig()

9.3.2.44 template<class _TC > bool coco::coco::coco::operator!= (const interval & *__i*, const _TC & *__d*) [inline]

Definition at line 823 of file search_graph.cc.

9.3.2.45 bool coco::coco::coco::operator!= (const interval & *a*, const interval & *b*) [inline]

Definition at line 519 of file search_graph.cc.

9.3.2.46 bool coco::coco::coco::operator!= (const interval & *a*, double *b*) [inline]

Definition at line 521 of file search_graph.cc.

9.3.2.47 interval coco::coco::coco::operator* (const interval & *a*, const interval & *b*)

9.3.2.48 interval coco::coco::coco::operator* (const interval & *a*, double *b*)

9.3.2.49 interval coco::coco::coco::operator* (double *b*, const interval & *a*)

9.3.2.50 interval coco::coco::coco::operator+ (const interval & *a*, const interval & *b*)

9.3.2.51 interval coco::coco::coco::operator+ (const interval & *a*, double *b*)

9.3.2.52 interval coco::coco::coco::operator+ (double *b*, const interval & *a*)

9.3.2.53 interval coco::coco::coco::operator- (const interval & *a*, const interval & *b*)

9.3.2.54 interval coco::coco::coco::operator- (const interval & *a*, double *b*)

9.3.2.55 interval coco::coco::coco::operator- (double *b*, const interval & *a*)

9.3.2.56 interval coco::coco::coco::operator/ (const interval & *a*, const interval & *b*)

9.3.2.57 interval coco::coco::coco::operator/ (const interval & *a*, double *b*)

9.3.2.58 interval coco::coco::coco::operator/ (double *b*, const interval & *a*)

9.3.2.59 bool coco::coco::coco::operator< (const interval & *a*, const interval & *b*) [inline]

Definition at line 523 of file search_graph.cc.

9.3.2.60 bool coco::coco::coco::operator< (const interval & *a*, double *b*) [inline]

Definition at line 524 of file search_graph.cc.

9.3.2.61 bool coco::coco::coco::operator< (double *a*, const interval & *b*) [inline]

Definition at line 525 of file search_graph.cc.

9.3.2.62 std::ostream & coco::coco::coco::operator<< (std::ostream & *s*, const interval & *a*)

9.3.2.63 std::ostream& coco::coco::coco::operator<< (std::ostream & *os*, const basic_alltype & *b*)

9.3.2.64 template<class _TC > bool coco::coco::coco::operator== (const interval & *__i*, const _TC & *__d*) [inline]

Definition at line 815 of file search_graph.cc.

9.3.2.65 bool coco::coco::coco::operator== (const interval & *a*, const interval & *b*) [inline]

Definition at line 515 of file search_graph.cc.

9.3.2.66 bool coco::coco::coco::operator== (const interval & *a*, double *b*) [inline]

Definition at line 517 of file search_graph.cc.

9.3.2.67 interval coco::coco::coco::pow (const interval & *x*, const interval & *y*)

Returns an interval enclosure of x^y = exp(y∗log(x)).

9.3.2.68 interval coco::coco::coco::power (const interval & *x*, int *n*)

Returns an interval enclosure of x^n.

9.3.2.69 double coco::coco::coco::rad (const interval &)

rad(a) == a.rad()

9.3.2.70 double coco::coco::coco::relDiam (const interval &)

relDiam(a) == a.relDiam()

9.3.2.71 interval coco::coco::coco::round_to_integer (const interval & *x*)

This function rounds the interval inward to integer borders.

9.3.2.72 double coco::coco::coco::safeguarded_mid (const interval & *__i*) [inline]

Definition at line 493 of file search_graph.cc.

9.3.2.73 interval coco::coco::coco::sin (const interval & *x*)

Returns an interval enclosure of the sine of the interval x.

9.3.2.74 interval coco::coco::coco::sinh (const interval & *x*)

Returns an interval enclosure of the hyperbolic sine of the interval x.

9.3.2.75 interval coco::coco::coco::sqr (const interval & *x*)

Returns an interval enclosure of x^2.

9.3.2.76 interval coco::coco::coco::sqrt (const interval & *x*)

Returns an interval enclosure of the square root of the interval x.

9.3.2.77 interval coco::coco::coco::tan (const interval & *x*)

Returns an interval enclosure of the tangent of the interval x.

9.3.2.78 interval coco::coco::coco::tanh (const interval & *x*)

Returns an interval enclosure of the hyperbolic tangent of the interval x.

9.3.2.79 double coco::coco::coco::width (const interval &)

9.4 coco::coco::num Namespace Reference

Classes

- class Number
- class number_exception

Typedefs

- typedef mpq_t rat
- typedef MP_RAT * prat
- typedef const MP_RAT * cprat
- typedef coco::interval intv
- typedef std::string str

Enumerations

- enum numtypes { NT_RAT, NT_STR, NT_INTV, NT_DBL }
- enum number_exception_type { NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR }

9.4.1 Typedef Documentation

9.4.1.1 typedef const MP_RAT* coco::coco::num::cprat

Definition at line 44 of file search_graph.cc.

9.4.1.2 typedef coco::interval coco::coco::num::intv

Definition at line 45 of file search_graph.cc.

9.4.1.3 typedef MP_RAT* coco::coco::num::prat

Definition at line 43 of file search_graph.cc.

9.4.1.4 typedef mpq_t coco::coco::num::rat

Definition at line 42 of file search_graph.cc.

9.4.1.5 typedef std::string coco::coco::num::str

Definition at line 46 of file search_graph.cc.

9.4.2 Enumeration Type Documentation

9.4.2.1 enum coco::coco::num::number_exception_type

Enumerator:

NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR

Definition at line 251 of file search_graph.cc.

9.4.2.2 enum coco::coco::num::numtypes

Enumerator:

NT_RAT
NT_STR
NT_INTV
NT_DBL

Definition at line 47 of file search_graph.cc.

9.5 coco::num Namespace Reference

Classes

- class Number
- class number_exception

Typedefs

- typedef mpq_t rat
- typedef MP_RAT ∗ prat
- typedef const MP_RAT ∗ cprat
- typedef coco::interval intv
- typedef std::string str

Enumerations

- enum numtypes { NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL }
- enum number_exception_type { NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR }
- enum numtypes { NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL }
- enum number_exception_type { NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR }
- enum numtypes { NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL }
- enum number_exception_type { NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR }
- enum numtypes { NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL, NT_RAT, NT_STR, NT_INTV, NT_DBL }
- enum number_exception_type { NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR, NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR }

9.5.1 Typedef Documentation

9.5.1.1 typedef const MP_RAT * coco::num::cprat

Definition at line 44 of file expression.h.

9.5.1.2 typedef coco::interval coco::num::intv

Definition at line 45 of file expression.h.

9.5.1.3 typedef MP_RAT * coco::num::prat

Definition at line 43 of file expression.h.

9.5.1.4 typedef mpq_t coco::num::rat

Definition at line 42 of file expression.h.

9.5.1.5 typedef std::string coco::num::str

Definition at line 46 of file expression.h.

9.5.2 Enumeration Type Documentation

9.5.2.1 enum coco::num::number_exception_type

Enumerator:

NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR

Definition at line 251 of file expression.h.

9.5.2.2 enum coco::num::number_exception_type

Enumerator:

NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR

Definition at line 251 of file search_graph.cc.

9.5.2.3 enum coco::num::number_exception_type

Enumerator:

NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT

NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR

Definition at line 251 of file search_graph.cc.

9.5.2.4 enum coco::num::number_exception_type

Enumerator:

NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR
NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR

Definition at line 251 of file search_graph.cc.

9.5.2.5 enum coco::num::numtypes

Enumerator:

NT_RAT
NT_STR
NT_INTV
NT_DBL

NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL

Definition at line 47 of file search_graph.cc.

9.5.2.6 enum coco::num::numtypes

Enumerator:

NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL

Definition at line 47 of file expression.h.

9.5.2.7 enum coco::num::numtypes

Enumerator:

NT_RAT
NT_STR
NT_INTV

NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL

Definition at line 47 of file search_graph.cc.

9.5.2.8 enum coco::num::numtypes

Enumerator:

NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL
NT_RAT
NT_STR
NT_INTV
NT_DBL

Definition at line 47 of file search_graph.cc.

9.6 coco_api_internal Namespace Reference

9.7 num Namespace Reference

Classes

- class Number
- class number_exception

Typedefs

- typedef mpq_t rat
- typedef MP_RAT * prat
- typedef const MP_RAT * cprat
- typedef coco::interval intv
- typedef std::string str

Enumerations

- enum numtypes { NT_RAT, NT_STR, NT_INTV, NT_DBL }
- enum number_exception_type { NE_IRRATIONAL, NE_NOSTRING, NE_INVALIDINPUT, NE_SIMPLIFY, NE_INTERNAL_ERROR }

Functions

- std::ostream & operator<< (std::ostream &os, const Number &n)
- std::istream & operator>> (std::istream &is, Number &n)
- Number acos (const Number &x)
- Number abs (const Number &x)
- Number acosh (const Number &x)
- Number acot (const Number &x)
- Number acoth (const Number &x)
- Number asin (const Number &x)
- Number asinh (const Number &x)
- Number atan (const Number &x)
- Number atanh (const Number &x)
- Number cos (const Number &x)
- Number cosh (const Number &x)
- Number cot (const Number &x)
- Number coth (const Number &x)
- Number exp (const Number &x)
- Number exp10 (const Number &x)
- Number exp2 (const Number &x)
- Number expm1 (const Number &x)
- Number log (const Number &x)
- Number log10 (const Number &x)
- Number log1p (const Number &x)
- Number log2 (const Number &x)
- Number power (const Number &x, int n)
- Number pow (const Number &x, const Number &y)
- Number sin (const Number &x)
- Number sinh (const Number &x)
- Number sqr (const Number &x)
- Number sqrt (const Number &x)
- Number tan (const Number &x)
- Number tanh (const Number &x)

9.7.1 Typedef Documentation

9.7.1.1 typedef const MP_RAT* num::cprat

Definition at line 44 of file Number.h.

9.7.1.2 typedef coco::interval num::intv

Definition at line 45 of file Number.h.

9.7.1.3 typedef MP_RAT* num::prat

Definition at line 43 of file Number.h.

9.7.1.4 typedef mpq_t num::rat

Definition at line 42 of file Number.h.

9.7.1.5 typedef std::string num::str

Definition at line 46 of file Number.h.

9.7.2 Enumeration Type Documentation

9.7.2.1 enum num::number_exception_type

Enumerator:

NE_IRRATIONAL
NE_NOSTRING
NE_INVALIDINPUT
NE_SIMPLIFY
NE_INTERNAL_ERROR

Definition at line 251 of file Number.h.

9.7.2.2 enum num::numtypes

Enumerator:

NT_RAT
NT_STR
NT_INTV
NT_DBL

Definition at line 47 of file Number.h.

9.7.3 Function Documentation

9.7.3.1 Number num::abs (const Number & *x*)

Definition at line 526 of file Number.cc.

9.7.3.2 Number num::acos (const Number & *x*)

Definition at line 515 of file Number.cc.

9.7.3.3 Number num::acosh (const Number & *x*)

Definition at line 546 of file Number.cc.

9.7.3.4 Number num::acot (const Number & *x*)

Definition at line 557 of file Number.cc.

9.7.3.5 Number num::acoth (const Number & *x*)

Definition at line 568 of file Number.cc.

9.7.3.6 Number num::asin (const Number & *x*)

Definition at line 579 of file Number.cc.

9.7.3.7 Number num::asinh (const Number & *x*)

Definition at line 590 of file Number.cc.

9.7.3.8 Number num::atan (const Number & *x*)

Definition at line 601 of file Number.cc.

9.7.3.9 Number num::atanh (const Number & *x*)

Definition at line 612 of file Number.cc.

9.7.3.10 Number num::cos (const Number & *x*)

Definition at line 623 of file Number.cc.

9.7.3.11 Number num::cosh (const Number & *x*)

Definition at line 634 of file Number.cc.

9.7.3.12 Number num::cot (const Number & *x*)

Definition at line 645 of file Number.cc.

9.7.3.13 Number num::coth (const Number & *x*)

Definition at line 656 of file Number.cc.

9.7.3.14 Number num::exp (const Number & *x*)

Definition at line 667 of file Number.cc.

9.7.3.15 Number num::exp10 (const Number & *x*)

Definition at line 678 of file Number.cc.

9.7.3.16 Number num::exp2 (const Number & *x*)

Definition at line 724 of file Number.cc.

9.7.3.17 Number num::expm1 (const Number & *x*)

Definition at line 770 of file Number.cc.

9.7.3.18 Number num::log (const Number & *x*)

Definition at line 782 of file Number.cc.

9.7.3.19 Number num::log10 (const Number & *x*)

Definition at line 793 of file Number.cc.

9.7.3.20 Number num::log1p (const Number & *x*)

Definition at line 805 of file Number.cc.

9.7.3.21 Number num::log2 (const Number & *x*)

Definition at line 817 of file Number.cc.

9.7.3.22 std::ostream& num::operator<< (std::ostream & *os,* const Number & *n*)

Definition at line 313 of file Number.cc.

9.7.3.23 std::istream& num::operator>> (std::istream & *is,* Number & *n*)

Definition at line 403 of file Number.cc.

9.7.3.24 Number num::pow (const Number & *x,* const Number & *y*)

Definition at line 862 of file Number.cc.

9.7.3.25 Number num::power (const Number & *x,* int *n*)

Definition at line 829 of file Number.cc.

9.7.3.26 Number num::sin (const Number & *x*)

Definition at line 878 of file Number.cc.

9.7.3.27 Number num::sinh (const Number & *x*)

Definition at line 889 of file Number.cc.

9.7.3.28 Number num::sqr (const Number & *x*)

Definition at line 900 of file Number.cc.

9.7.3.29 Number num::sqrt (const Number & *x*)

Definition at line 912 of file Number.cc.

9.7.3.30 Number num::tan (const Number & *x*)

Definition at line 934 of file Number.cc.

9.7.3.31 Number num::tanh (const Number & *x*)

Definition at line 945 of file Number.cc.

9.8 std Namespace Reference

The standard namespace.

Classes

- struct triple

  triple holds three objects of arbitrary type.

Enumerations

- enum { _M_p_chunk_size = 7 }
- enum { _M_p_threshold = 16 }

Functions

- ostream & operator<< (ostream &s, const coco::interval_set &a)
- ostream & operator<< (ostream &o, const coco::proj_rational &x)
- template<class I >
  ostream & operator<< (ostream &o, const coco::projective_interval< I > &a)
- template<typename _TK , typename _Ve , typename _TC , typename _TA >
  ostream & operator<< (ostream &o, const map< _TK, _Ve, _TC, _TA > &a)
- template<typename _TK , typename _Ve , typename _TC , typename _TA >
  ostream & operator<< (ostream &o, const multimap< _TK, _Ve, _TC, _TA > &a)
- template<typename _Ve , typename _TA >
  ostream & operator<< (ostream &o, const vector< _Ve, _TA > &a)
- template<typename _Ve , typename _TA >
  ostream & operator<< (ostream &o, const list< _Ve, _TA > &a)

- template<typename _Ve , typename _TA >
  ostream & operator<< (ostream &o, const deque< _Ve, _TA > &a)
- template<typename _Ve , typename _TC , typename _TA >
  ostream & operator<< (ostream &o, const set< _Ve, _TC, _TA > &a)
- template<typename _Ve , typename _TC , typename _TA >
  ostream & operator<< (ostream &o, const multiset< _Ve, _TC, _TA > &a)
- template<typename _TA , typename _TB >
  ostream & operator<< (ostream &o, const pair< _TA, _TB > &a)
- template<typename _TA , typename _TB , typename _TC >
  ostream & operator<< (ostream &o, const triple< _TA, _TB, _TC > &a)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp >
  pair< _RandomAccessIter, _RandomAccessIterp > __unguarded_p_partition (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Tp __pivot)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp , typename _Compare >
  pair< _RandomAccessIter, _RandomAccessIterp > __unguarded_p_partition (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Tp __pivot, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp , typename _Tpp >
  void __unguarded_linear_p_insert (_RandomAccessIter __last1, _Tp __val1, _RandomAccessIterp __last2, _Tpp __val2)
- template<typename _RandomAccessIter , typename _Tp , typename _RandomAccessIterp , typename _Tpp , typename _Compare >
  void __unguarded_linear_p_insert (_RandomAccessIter __last1, _Tp __val1, _RandomAccessIterp __last2, _Tpp __val2, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void __insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void __insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void __unguarded_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void __unguarded_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void __final_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void __final_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size >
  void __intro_p_sort_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size , typename _Compare >
  void __intro_p_sort_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size >
  void __introsort_p_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit)

- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size , typename _Compare >
  void __introsort_p_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  bool pair_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

  Sort the elements of two sequences in common.
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  bool pair_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

  Sort the elements of a sequence using a predicate for comparison.
- template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance >
  void __merge_p_without_buffer (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2)
- template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Compare >
  void __merge_p_without_buffer (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2, _Compare __comp)
- template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Pointer , typename _Pointerp >
  void __p_merge_adaptive (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2, _Pointer __buffer, _Pointerp __bufferp, _Distance __buffer_size)
- template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Pointer , typename _Pointerp , typename _Compare >
  void __p_merge_adaptive (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2, _Pointer __buffer, _Pointerp __bufferp, _Distance __buffer_size, _Compare __comp)
- template<typename _InputIter1 , typename _InputIter2 , typename _OutputIter , typename _InputIter1p , typename _InputIter2p , typename _OutputIterp >
  pair< _OutputIter, _OutputIterp > pair_merge (_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _InputIter1p __first1p, _InputIter1p __last1p, _InputIter2p __first2p, _InputIter2p __last2p, _OutputIterp __resultp)

  Merges two sorted ranges.
- template<typename _InputIter1 , typename _InputIter2 , typename _OutputIter , typename _InputIter1p , typename _InputIter2p , typename _OutputIterp , typename _Compare >
  pair< _OutputIter, _OutputIterp > pair_merge (_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _InputIter1p __first1p, _InputIter1p __last1p, _InputIter2p __first2p, _InputIter2p __last2p, _OutputIterp __resultp, _Compare __comp)

  Merges two sorted ranges.
- template<typename _BidirectionalIter1 , typename _BidirectionalIter2 , typename _BidirectionalIter3 , typename _BidirectionalIter1p , typename _BidirectionalIter2p , typename _BidirectionalIter3p >
  pair< _BidirectionalIter3, _BidirectionalIter3p > __p_merge_backward (_BidirectionalIter1 __first1, _BidirectionalIter1 __last1, _BidirectionalIter2 __first2, _BidirectionalIter2 __last2, _BidirectionalIter3 __result, _BidirectionalIter1p __first1p, _BidirectionalIter1p __last1p, _BidirectionalIter2p __first2p, _BidirectionalIter2p __last2p, _BidirectionalIter3p __resultp)
- template<typename _BidirectionalIter1 , typename _BidirectionalIter2 , typename _BidirectionalIter3 , typename _BidirectionalIter1p , typename _BidirectionalIter2p , typename _BidirectionalIter3p , typename _Compare >

pair< _BidirectionalIter3, _BidirectionalIter3p > __p_merge_backward (_BidirectionalIter1 __first1, _BidirectionalIter1 __last1, _BidirectionalIter2 __first2, _BidirectionalIter2 __last2, _BidirectionalIter3 __result, _BidirectionalIter1p __first1p, _BidirectionalIter1p __last1p, _BidirectionalIter2p __first2p, _BidirectionalIter2p __last2p, _BidirectionalIter3p __resultp, _Compare __comp)

- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void __inplace_stable_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void __inplace_stable_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)
- template<typename _RandomAccessIter1 , typename _RandomAccessIter2 , typename _RandomAccessIter1p , typename _RandomAccessIter2p , typename _Distance >
  void __merge_p_sort_loop (_RandomAccessIter1 __first, _RandomAccessIter1 __last, _RandomAccessIter2 __result, _RandomAccessIter1p __firstp, _RandomAccessIter1p __lastp, _RandomAccessIter2p __resultp, _Distance __step_size)
- template<typename _RandomAccessIter1 , typename _RandomAccessIter2 , typename _RandomAccessIter1p , typename _RandomAccessIter2p , typename _Distance , typename _Compare >
  void __merge_p_sort_loop (_RandomAccessIter1 __first, _RandomAccessIter1 __last, _RandomAccessIter2 __result, _RandomAccessIter1p __firstp, _RandomAccessIter1p __lastp, _RandomAccessIter2p __resultp, _Distance __step_size, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Distance >
  void __chunk_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Distance __chunk_size)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Distance , typename _Compare >
  void __chunk_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Distance __chunk_size, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Pointer , typename _Pointerp >
  void __merge_p_sort_with_buffer (_RandomAccessIter __first, _RandomAccessIter __last, _Pointer __buffer, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Pointerp __bufferp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Pointer , typename _Pointerp , typename _Compare >
  void __merge_p_sort_with_buffer (_RandomAccessIter __first, _RandomAccessIter __last, _Pointer __buffer, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Pointerp __bufferp, _Compare __comp)
- template<typename _RandomAccessIter , typename _Pointer , typename _RandomAccessIterp , typename _Pointerp , typename _Distance >
  void __stable_p_sort_adaptive (_RandomAccessIter __first, _RandomAccessIter __last, _Pointer __buffer, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Pointerp __bufferp, _Distance __buffer_size)
- template<typename _RandomAccessIter , typename _Pointer , typename _RandomAccessIterp , typename _Pointerp , typename _Distance , typename _Compare >
  void __stable_p_sort_adaptive (_RandomAccessIter __first, _RandomAccessIter __last, _Pointer __buffer, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Pointerp __bufferp, _Distance __buffer_size, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  bool stable_pair_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

  Sort the elements of a pair of sequences, preserving the relative order of equivalent elements.
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  bool stable_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

  Sort the elements of a pair of sequences using a predicate for comparison, preserving the relative order of equivalent elements.

- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  bool partial_pair_sort (_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

  Sort the smallest elements of a pair of sequences.
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  bool partial_pair_sort (_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

  Sort the smallest elements of a pair of sequences.
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  bool partial_sort (_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last, _RandomAccessIter __firstp, _RandomAccessIter __lastp, _Compare __comp)

  Sort the smallest elements of a pair of sequences using a predicate for comparison.
- template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp >
  void __push_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __topIndex, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __topIndexp, _Tpp __valuep)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp >
  void push_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)
- template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp , typename _Compare >
  void __push_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __topIndex, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __topIndexp, _Tpp __valuep, _Compare __comp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare >
  void push_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp)
- template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp >
  void __adjust_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __len, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __lenp, _Tpp __valuep)
- template<typename _RandomAccessIterator , typename _Tp , typename _RandomAccessIteratorp , typename _Tpp >
  void __pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIterator __result, _Tp __value, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _RandomAccessIteratorp __resultp, _Tpp __valuep)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp >
  void pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)
- template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp , typename _Compare >
  void __adjust_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __len, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __lenp, _Tpp __valuep, _Compare __comp)
- template<typename _RandomAccessIterator , typename _Tp , typename _RandomAccessIteratorp , typename _Tpp , typename _Compare >
  void __pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIterator __result, _Tp __value, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _RandomAccessIteratorp __resultp, _Tpp __valuep, _Compare __comp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare >
  void pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp)

- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp >
  void make_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare >
  void make_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp >
  void sort_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare >
  void sort_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp)
- template<class _T1 , class _T2 , class _T3 >
  bool operator== (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Two triples of the same type are equal iff their members are equal.
- template<class _T1 , class _T2 , class _T3 >
  bool operator< (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  This is lexicographic ordering of triples.
- template<class _T1 , class _T2 , class _T3 >
  bool operator!= (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Uses `operator==` to find the result.
- template<class _T1 , class _T2 , class _T3 >
  bool operator> (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Uses `operator<` to find the result.
- template<class _T1 , class _T2 , class _T3 >
  bool operator<= (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Uses `operator<` to find the result.
- template<class _T1 , class _T2 , class _T3 >
  bool operator>= (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Uses `operator<` to find the result.
- template<class _T1 , class _T2 , class _T3 >
  triple< _T1, _T2, _T3 > make_triple (const _T1 &__x, const _T2 &__y, const _T3 &__z)

  A convenience wrapper for creating a triple from three objects.

9.8.1 Detailed Description

This is the standard namespace of the C++ library.

9.8.2 Enumeration Type Documentation

9.8.2.1 anonymous enum

Enumerator:

_M_p_chunk_size

Definition at line 62 of file stlp_addalgo.h.

9.8.2.2 anonymous enum

Enumerator:

_M_p_threshold

Definition at line 63 of file stlp_addalgo.h.

9.8.3 Function Documentation

9.8.3.1 template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp > void std::__adjust_pair_heap (_RandomAccessIterator *__first*, _Distance *__holeIndex*, _Distance *__len*, _Tp *__value*, _RandomAccessIteratorp *__firstp*, _Distancep *__holeIndexp*, _Distancep *__lenp*, _Tpp *__valuep*)

Definition at line 155 of file stlp_pairheap.h.

9.8.3.2 template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp , typename _Compare > void std::__adjust_pair_heap (_RandomAccessIterator *__first*, _Distance *__holeIndex*, _Distance *__len*, _Tp *__value*, _RandomAccessIteratorp *__firstp*, _Distancep *__holeIndexp*, _Distancep *__lenp*, _Tpp *__valuep*, _Compare *__comp*)

Definition at line 230 of file stlp_pairheap.h.

9.8.3.3 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Distance > void std::__chunk_insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Distance *__chunk_size*)

Definition at line 1178 of file stlp_addalgo.h.

9.8.3.4 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Distance , typename _Compare > void std::__chunk_insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Distance *__chunk_size*, _Compare *__comp*)

Definition at line 1194 of file stlp_addalgo.h.

9.8.3.5 template<typename _RandomAccessIter , typename _RandomAccessIterp > void std::__final_insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*)

Definition at line 296 of file stlp_addalgo.h.

9.8.3.6 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > void std::__final_insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Compare *__comp*)

Definition at line 318 of file stlp_addalgo.h.

9.8.3.7 template<typename _RandomAccessIter , typename _RandomAccessIterp > void std::__inplace_stable_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

Definition at line 1069 of file stlp_addalgo.h.

9.8.3.8 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > void std::__inplace_stable_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

Definition at line 1094 of file stlp_addalgo.h.

9.8.3.9 template<typename _RandomAccessIter , typename _RandomAccessIterp > void std::__insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

Definition at line 193 of file stlp_addalgo.h.

9.8.3.10 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > void std::__insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

Definition at line 223 of file stlp_addalgo.h.

9.8.3.11 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size > void std::__intro_p_sort_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit)

Definition at line 340 of file stlp_addalgo.h.

9.8.3.12 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size , typename _Compare > void std::__intro_p_sort_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit, _Compare __comp)

Definition at line 372 of file stlp_addalgo.h.

9.8.3.13 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size > void std::__introsort_p_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit)

This is a helper function for the pair_sort routine.

Definition at line 403 of file stlp_addalgo.h.

9.8.3.14 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size , typename _Compare > void std::__introsort_p_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit, _Compare __comp)

Definition at line 435 of file stlp_addalgo.h.

9.8.3.15 template<typename _RandomAccessIter1 , typename _RandomAccessIter2 , typename _RandomAccessIter1p , typename _RandomAccessIter2p , typename _Distance > void std::__merge_p_sort_loop (_RandomAccessIter1 *__first,* _RandomAccessIter1 *__last,* _RandomAccessIter2 *__result,* _RandomAccessIter1p *__firstp,* _RandomAccessIter1p *__lastp,* _RandomAccessIter2p *__resultp,* _Distance *__step_size*)

Definition at line 1117 of file stlp_addalgo.h.

9.8.3.16 template<typename _RandomAccessIter1 , typename _RandomAccessIter2 , typename _RandomAccessIter1p , typename _RandomAccessIter2p , typename _Distance , typename _Compare > void std::__merge_p_sort_loop (_RandomAccessIter1 *__first,* _RandomAccessIter1 *__last,* _RandomAccessIter2 *__result,* _RandomAccessIter1p *__firstp,* _RandomAccessIter1p *__lastp,* _RandomAccessIter2p *__resultp,* _Distance *__step_size,* _Compare *__comp*)

Definition at line 1147 of file stlp_addalgo.h.

9.8.3.17 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Pointer , typename _Pointerp > void std::__merge_p_sort_with_buffer (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _Pointer *__buffer,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp,* _Pointerp *__bufferp*)

Definition at line 1210 of file stlp_addalgo.h.

9.8.3.18 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Pointer , typename _Pointerp , typename _Compare > void std::__merge_p_sort_with_buffer (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _Pointer *__buffer,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp,* _Pointerp *__bufferp,* _Compare *__comp*)

Definition at line 1239 of file stlp_addalgo.h.

9.8.3.19 template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance > void std::__merge_p_without_buffer (_BidirectionalIter *__first,* _BidirectionalIter *__middle,* _BidirectionalIter *__last,* _BidirectionalIterp *__firstp,* _BidirectionalIterp *__middlep,* _BidirectionalIterp *__lastp,* _Distance *__len1,* _Distance *__len2*)

Definition at line 562 of file stlp_addalgo.h.

9.8.3.20 template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Compare > void std::__merge_p_without_buffer (_BidirectionalIter *__first,* _BidirectionalIter *__middle,* _BidirectionalIter *__last,* _BidirectionalIterp *__firstp,* _BidirectionalIterp *__middlep,* _BidirectionalIterp *__lastp,* _Distance *__len1,* _Distance *__len2,* _Compare *__comp*)

Definition at line 624 of file stlp_addalgo.h.

9.8.3.21 template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Pointer , typename _Pointerp > void std::__p_merge_adaptive (_BidirectionalIter *__first,* _BidirectionalIter *__middle,* _BidirectionalIter *__last,* _BidirectionalIterp *__firstp,* _BidirectionalIterp *__middlep,* _BidirectionalIterp *__lastp,* _Distance *__len1,* _Distance *__len2,* _Pointer *__buffer,* _Pointerp *__bufferp,* _Distance *__buffer_size*)

Definition at line 686 of file stlp_addalgo.h.

9.8.3.22 template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Pointer , typename _Pointerp , typename _Compare > void std::__p_merge_adaptive (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2, _Pointer __buffer, _Pointerp __bufferp, _Distance __buffer_size, _Compare __comp)

Definition at line 758 of file stlp_addalgo.h.

9.8.3.23 template<typename _BidirectionalIter1 , typename _BidirectionalIter2 , typename _BidirectionalIter3 , typename _BidirectionalIter1p , typename _BidirectionalIter2p , typename _BidirectionalIter3p > pair<_BidirectionalIter3,_BidirectionalIter3p> std::__p_merge_backward (_BidirectionalIter1 __first1, _BidirectionalIter1 __last1, _BidirectionalIter2 __first2, _BidirectionalIter2 __last2, _BidirectionalIter3 __result, _BidirectionalIter1p __first1p, _BidirectionalIter1p __last1p, _BidirectionalIter2p __first2p, _BidirectionalIter2p __last2p, _BidirectionalIter3p __resultp)

Definition at line 973 of file stlp_addalgo.h.

9.8.3.24 template<typename _BidirectionalIter1 , typename _BidirectionalIter2 , typename _BidirectionalIter3 , typename _BidirectionalIter1p , typename _BidirectionalIter2p , typename _BidirectionalIter3p , typename _Compare > pair<_BidirectionalIter3,_BidirectionalIter3p> std::__p_merge_backward (_BidirectionalIter1 __first1, _BidirectionalIter1 __last1, _BidirectionalIter2 __first2, _BidirectionalIter2 __last2, _BidirectionalIter3 __result, _BidirectionalIter1p __first1p, _BidirectionalIter1p __last1p, _BidirectionalIter2p __first2p, _BidirectionalIter2p __last2p, _BidirectionalIter3p __resultp, _Compare __comp)

Definition at line 1022 of file stlp_addalgo.h.

9.8.3.25 template<typename _RandomAccessIterator , typename _Tp , typename _RandomAccessIteratorp , typename _Tpp > void std::__pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIterator __result, _Tp __value, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _RandomAccessIteratorp __resultp, _Tpp __valuep) [inline]

Definition at line 190 of file stlp_pairheap.h.

9.8.3.26 template<typename _RandomAccessIterator , typename _Tp , typename _RandomAccessIteratorp , typename _Tpp , typename _Compare > void std::__pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIterator __result, _Tp __value, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _RandomAccessIteratorp __resultp, _Tpp __valuep, _Compare __comp) [inline]

Definition at line 265 of file stlp_pairheap.h.

9.8.3.27 template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp > void std::__push_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __topIndex, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __topIndexp, _Tpp __valuep)

Definition at line 53 of file stlp_pairheap.h.

9.8.3.28 template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp , typename _Compare > void std::__push_pair_heap (_RandomAccessIterator *__first,* _Distance *__holeIndex,* _Distance *__topIndex,* _Tp *__value,* _RandomAccessIteratorp *__firstp,* _Distancep *__holeIndexp,* _Distancep *__topIndexp,* _Tpp *__valuep,* _Compare *__comp*)

Definition at line 105 of file stlp_pairheap.h.

9.8.3.29 template<typename _RandomAccessIter , typename _Pointer , typename _RandomAccessIterp , typename _Pointerp , typename _Distance > void std::__stable_p_sort_adaptive (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _Pointer *__buffer,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp,* _Pointerp *__bufferp,* _Distance *__buffer_size*)

Definition at line 1269 of file stlp_addalgo.h.

9.8.3.30 template<typename _RandomAccessIter , typename _Pointer , typename _RandomAccessIterp , typename _Pointerp , typename _Distance , typename _Compare > void std::__stable_p_sort_adaptive (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _Pointer *__buffer,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp,* _Pointerp *__bufferp,* _Distance *__buffer_size,* _Compare *__comp*)

Definition at line 1298 of file stlp_addalgo.h.

9.8.3.31 template<typename _RandomAccessIter , typename _RandomAccessIterp > void std::__unguarded_insertion_p_sort (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp*) [inline]

Definition at line 253 of file stlp_addalgo.h.

9.8.3.32 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > void std::__unguarded_insertion_p_sort (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp,* _Compare *__comp*) [inline]

Definition at line 275 of file stlp_addalgo.h.

9.8.3.33 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp , typename _Tpp > void std::__unguarded_linear_p_insert (_RandomAccessIter *__last1,* _Tp *__val1,* _RandomAccessIterp *__last2,* _Tpp *__val2*)

Definition at line 140 of file stlp_addalgo.h.

9.8.3.34 template<typename _RandomAccessIter , typename _Tp , typename _RandomAccessIterp , typename _Tpp , typename _Compare > void std::__unguarded_linear_p_insert (_RandomAccessIter *__last1,* _Tp *__val1,* _RandomAccessIterp *__last2,* _Tpp *__val2,* _Compare *__comp*)

Definition at line 167 of file stlp_addalgo.h.

9.8.3.35 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp > pair<_RandomAccessIter, _RandomAccessIterp> std::__unguarded_p_partition (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Tp __pivot)

Definition at line 72 of file stlp_addalgo.h.

9.8.3.36 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp , typename _Compare > pair<_RandomAccessIter, _RandomAccessIterp> std::__unguarded_p_partition (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Tp __pivot, _Compare __comp)

Definition at line 106 of file stlp_addalgo.h.

9.8.3.37 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp > void std::make_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)

Definition at line 303 of file stlp_pairheap.h.

9.8.3.38 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare > void std::make_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp) [inline]

Definition at line 343 of file stlp_pairheap.h.

9.8.3.39 template<class _T1 , class _T2 , class _T3 > triple<_T1, _T2, _T3> std::make_triple (const _T1 & __x, const _T2 & __y, const _T3 & __z) [inline]

Parameters

x	The first object.
y	The second object.
z	The third object.

Returns

A newly-constructed triple<> object of the appropriate type.

Definition at line 141 of file stlp_triple.h.

9.8.3.40 template<class _T1 , class _T2 , class _T3 > bool std::operator!= (const triple< _T1, _T2, _T3 > & __x, const triple< _T1, _T2, _T3 > & __y) [inline]

Definition at line 103 of file stlp_triple.h.

9.8.3.41 template<class _T1 , class _T2 , class _T3 > bool std::operator< (const triple< _T1, _T2, _T3 > & __x, const triple< _T1, _T2, _T3 > & __y) [inline]

Definition at line 92 of file stlp_triple.h.

9.8.3.42 ostream& std::operator<< (ostream & *o,* const coco::proj_rational & *x*) [inline]

Definition at line 550 of file prointerval.h.

9.8.3.43 template<class I > ostream& std::operator<< (ostream & *o,* const coco::projective_interval< I > & *a*) [inline]

Definition at line 560 of file prointerval.h.

9.8.3.44 ostream & std::operator<< (ostream & *s,* const coco::interval_set & *a*)

Definition at line 1091 of file interval_set.cc.

9.8.3.45 template<class _T1 , class _T2 , class _T3 > bool std::operator<= (const triple< _T1, _T2, _T3 > & *__x,* const triple< _T1, _T2, _T3 > & *__y*) [inline]

Definition at line 117 of file stlp_triple.h.

9.8.3.46 template<class _T1 , class _T2 , class _T3 > bool std::operator== (const triple< _T1, _T2, _T3 > & *__x,* const triple< _T1, _T2, _T3 > & *__y*) [inline]

Definition at line 83 of file stlp_triple.h.

9.8.3.47 template<class _T1 , class _T2 , class _T3 > bool std::operator> (const triple< _T1, _T2, _T3 > & *__x,* const triple< _T1, _T2, _T3 > & *__y*) [inline]

Definition at line 110 of file stlp_triple.h.

9.8.3.48 template<class _T1 , class _T2 , class _T3 > bool std::operator>= (const triple< _T1, _T2, _T3 > & *__x,* const triple< _T1, _T2, _T3 > & *__y*) [inline]

Definition at line 124 of file stlp_triple.h.

9.8.3.49 template<typename _InputIter1 , typename _InputIter2 , typename _OutputIter , typename _InputIter1p , typename _InputIter2p , typename _OutputIterp > pair<_OutputIter,_OutputIterp> std::pair_merge (_InputIter1 *__first1,* _InputIter1 *__last1,* _InputIter2 *__first2,* _InputIter2 *__last2,* _OutputIter *__result,* _InputIter1p *__first1p,* _InputIter1p *__last1p,* _InputIter2p *__first2p,* _InputIter2p *__last2p,* _OutputIterp *__resultp*)

Parameters

first1	An iterator.
first2	Another iterator.
last1	Another iterator.
last2	Another iterator.
result	An iterator pointing to the end of the merged range.

Returns

An iterator pointing to the first element "not less than" *val*.

Merges the ranges [first1,last1) and [first2,last2) into the sorted range [result, result + (last1-first1) + (last2-first2)). Both input ranges must be sorted, and the output range must not overlap with either of the input ranges. The sort is *stable*, that is, for equivalent elements in the two ranges, elements from the first range will always come before elements from the second.

Definition at line 841 of file stlp_addalgo.h.

9.8.3.50 template<typename _InputIter1 , typename _InputIter2 , typename _OutputIter , typename _InputIter1p , typename _InputIter2p , typename _OutputIterp , typename _Compare > pair<_OutputIter,_OutputIterp> std::pair_merge (_InputIter1 *__first1,* _InputIter1 *__last1,* _InputIter2 *__first2,* _InputIter2 *__last2,* _OutputIter *__result,* _InputIter1p *__first1p,* _InputIter1p *__last1p,* _InputIter2p *__first2p,* _InputIter2p *__last2p,* _OutputIterp *__resultp,* _Compare *__comp*)

Parameters

first1	An iterator.
first2	Another iterator.
last1	Another iterator.
last2	Another iterator.
result	An iterator pointing to the end of the merged range.
comp	A functor to use for comparisons.

Returns

An iterator pointing to the first element "not less than" *val*.

Merges the ranges [first1,last1) and [first2,last2) into the sorted range [result, result + (last1-first1) + (last2-first2)). Both input ranges must be sorted, and the output range must not overlap with either of the input ranges. The sort is *stable*, that is, for equivalent elements in the two ranges, elements from the first range will always come before elements from the second.

The comparison function should have the same effects on ordering as the function used for the initial sort.

Definition at line 914 of file stlp_addalgo.h.

9.8.3.51 template<typename _RandomAccessIter , typename _RandomAccessIterp > bool std::pair_sort (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp*) [inline]

Parameters

first	An iterator.
last	Another iterator.
firstp	Yet Another iterator.
lastp	Yet Another Another iterator.

Returns

true if the two ranges had the same length (i.e. sorting worked)

Sorts the elements in the range [first,last) in ascending order, such that ∗(i+1)<∗i is false for each iterator `i` in the range [first,last-1).

The relative ordering of equivalent elements is not preserved, use `stable_pair_sort()` if this is needed.

The order of the elements in the range [firstp, lastp) is changed in parallel to the changes made in the range [first, last). So elements in the two sequences which had the same index before the pair_sort will have the same index after the pair_sort.

Definition at line 482 of file stlp_addalgo.h.

9.8.3.52 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > bool std::pair_sort (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp,* _Compare *__comp*) [inline]

Parameters

first	An iterator.
last	Another iterator.
firstp	Yet Another iterator.
lastp	Yet Another Another iterator.
comp	A comparison functor.

Returns

true if the two ranges had the same length (i.e. sorting worked)

Sorts the elements in the range [first,last) in ascending order, such that `comp`(∗(i+1),∗i) is false for every iterator `i` in the range [first,last-1).

The relative ordering of equivalent elements is not preserved, use `stable_pair_sort()` if this is needed.

The order of the elements in the range [firstp, lastp) is changed in parallel to the changes made in the range [first, last). So elements in the two sequences which had the same index before the pair_sort will have the same index after the pair_sort.

Definition at line 530 of file stlp_addalgo.h.

9.8.3.53 template<typename _RandomAccessIter , typename _RandomAccessIterp > bool std::partial_pair_sort (_RandomAccessIter *__first,* _RandomAccessIter *__middle,* _RandomAccessIter *__last,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp*)

Parameters

first	An iterator.
middle	Another iterator.
last	Another iterator.
firstp	Yet Another iterator.
lastp	Yet Another Another iterator.

Returns

true if the two ranges had the same length (i.e. sorting worked)

Sorts the smallest (middle-first) elements in the range [first,last) and moves them to the range [first,middle). The order of the remaining elements in the range [middle,last) is undefined. After the sort if `i` and are iterators in the range [first,middle) such that precedes and is an iterator in the range [middle,last) then `*j<*i` and `*k<*i` are both false.

The order of the elements in the range [firstp, lastp) is changed in parallel to the changes made in the range [first, last). So elements in the two sequences which had the same index before the sort will have the same index after the sort.

Definition at line 1456 of file stlp_addalgo.h.

9.8.3.54 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > bool std::partial_pair_sort (_RandomAccessIter *__first*, _RandomAccessIter *__middle*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Compare *__comp*)

Parameters

first	An iterator.
middle	Another iterator.
last	Another iterator.
firstp	Yet Another iterator.
lastp	Yet Another Another iterator.

Returns

true if the two ranges had the same length (i.e. sorting worked)

Sorts the smallest (middle-first) elements in the range [first,last) and moves them to the range [first,middle). The order of the remaining elements in the range [middle,last) is undefined. After the sort if `i` and are iterators in the range [first,middle) such that precedes and is an iterator in the range [middle,last) then `*j<*i` and `*k<*i` are both false.

The order of the elements in the range [firstp, lastp) is changed in parallel to the changes made in the range [first, last). So elements in the two sequences which had the same index before the sort will have the same index after the sort.

Definition at line 1514 of file stlp_addalgo.h.

9.8.3.55 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > bool std::partial_sort (_RandomAccessIter *__first*, _RandomAccessIter *__middle*, _RandomAccessIter *__last*, _RandomAccessIter *__firstp*, _RandomAccessIter *__lastp*, _Compare *__comp*)

Parameters

first	An iterator.
middle	Another iterator.
last	Another iterator.

firstp	Yet Another iterator.
lastp	Yet Another Another iterator.
comp	A comparison functor.

Returns

true if the two ranges had the same length (i.e. sorting worked)

Sorts the smallest (middle-first) elements in the range [first,last) and moves them to the range [first,middle). The order of the remaining elements in the range [middle,last) is undefined. After the sort if `i` and are iterators in the range [first,middle) such that precedes and is an iterator in the range [middle,last) then `*comp`(j,*i) and `comp(*k,*i)` are both false.

The order of the elements in the range [firstp, lastp) is changed in parallel to the changes made in the range [first, last). So elements in the two sequences which had the same index before the sort will have the same index after the sort.

Definition at line 1577 of file stlp_addalgo.h.

9.8.3.56 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp > void std::pop_pair_heap (_RandomAccessIterator *__first,* _RandomAccessIterator *__last,* _RandomAccessIteratorp *__firstp,* _RandomAccessIteratorp *__lastp*) [inline]

Definition at line 206 of file stlp_pairheap.h.

9.8.3.57 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare > void std::pop_pair_heap (_RandomAccessIterator *__first,* _RandomAccessIterator *__last,* _RandomAccessIteratorp *__firstp,* _RandomAccessIteratorp *__lastp,* _Compare *__comp*) [inline]

Definition at line 282 of file stlp_pairheap.h.

9.8.3.58 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp > void std::push_pair_heap (_RandomAccessIterator *__first,* _RandomAccessIterator *__last,* _RandomAccessIteratorp *__firstp,* _RandomAccessIteratorp *__lastp*) [inline]

Definition at line 74 of file stlp_pairheap.h.

9.8.3.59 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare > void std::push_pair_heap (_RandomAccessIterator *__first,* _RandomAccessIterator *__last,* _RandomAccessIteratorp *__firstp,* _RandomAccessIteratorp *__lastp,* _Compare *__comp*) [inline]

Definition at line 127 of file stlp_pairheap.h.

9.8.3.60 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp > void std::sort_pair_heap (_RandomAccessIterator *__first,* _RandomAccessIterator *__last,* _RandomAccessIteratorp *__firstp,* _RandomAccessIteratorp *__lastp*)

Definition at line 383 of file stlp_pairheap.h.

9.8.3.61 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare > void std::sort_pair_heap (_RandomAccessIterator *__first,* _RandomAccessIterator *__last,* _RandomAccessIteratorp *__firstp,* _RandomAccessIteratorp *__lastp,* _Compare *__comp*)

Definition at line 403 of file stlp_pairheap.h.

9.8.3.62 template<typename _RandomAccessIter , typename _RandomAccessIterp > bool std::stable_pair_sort (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp*) [inline]

Parameters

first	An iterator.
last	Another iterator.
firstp	Yet Another iterator.
lastp	Yet Another Another iterator.

Returns

true if the two ranges had the same length (i.e. sorting worked)

Sorts the elements in the range [first,last) in ascending order, such that ∗(i+1)<∗i is false for each iterator `i` in the range [first,last-1).

The relative ordering of equivalent elements is preserved, so any two elements `x` and `y` in the range [first,last) such that `x<y` is false and `y<x` is false will have the same relative ordering after calling `stable_sort()`.

The order of the elements in the range [firstp, lastp) is changed in parallel to the changes made in the range [first, last). So elements in the two sequences which had the same index before the sort will have the same index after the sort.

Definition at line 1349 of file stlp_addalgo.h.

9.8.3.63 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > bool std::stable_sort (_RandomAccessIter *__first,* _RandomAccessIter *__last,* _RandomAccessIterp *__firstp,* _RandomAccessIterp *__lastp,* _Compare *__comp*) [inline]

Parameters

first	An iterator.
last	Another iterator.
firstp	Yet Another iterator.
lastp	Yet Another Another iterator.
comp	A comparison functor.

Returns

true if the two ranges had the same length (i.e. sorting worked)

Sorts the elements in the range [first,last) in ascending order, such that `comp`(∗(i+1),∗i) is false for each iterator `i` in the range [first,last-1).

The relative ordering of equivalent elements is preserved, so any two elements `x` and `y` in the range [first,last) such that `comp(x,y)` is false and `comp(y,x)` is false will have the same relative ordering after calling `stable_sort()`.

The order of the elements in the range [firstp, lastp) is changed in parallel to the changes made in the range [first, last). So elements in the two sequences which had the same index before the sort will have the same index after the sort.

Definition at line 1404 of file stlp_addalgo.h.

10 Class Documentation

10.1 coco::dag_delta::__check_nodes Class Reference

Public Member Functions

- __check_nodes (const std::vector< walker > &__nr)
- bool operator() (const walker &_w) const

10.1.1 Detailed Description

This function class is used as a predicate for partitioning the vector of constraints in added constraints and original constraints using `stable_partition`.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 coco::dag_delta::__check_nodes::__check_nodes (const std::vector< walker > & *__nr*) [inline]

Definition at line 287 of file dag_delta.h.

10.1.3 Member Function Documentation

10.1.3.1 bool coco::dag_delta::__check_nodes::operator() (const walker & *_w*) const [inline]

Definition at line 289 of file dag_delta.h.

The documentation for this class was generated from the following file:

- dag_delta.h

10.2 coco::dag_undelta::__check_walkers Class Reference

Public Member Functions

- __check_walkers (const std::vector< walker > &__nr)
- bool operator() (const walker &_w) const

10.2.1 Detailed Description

This function class is used as a predicate for partitioning the vector of constraints in added constraints and original constraints using `stable_partition`.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 coco::dag_undelta::__check_walkers::__check_walkers (const std::vector< walker > & *__nr*) [inline]

Definition at line 89 of file dag_delta.h.

10.2.3 Member Function Documentation

10.2.3.1 bool coco::dag_undelta::__check_walkers::operator() (const walker & *_w*) const [inline]

Definition at line 91 of file dag_delta.h.

The documentation for this class was generated from the following file:

- dag_delta.h

10.3 coco::dag_delta::__docompare_nodes Struct Reference

Public Member Functions

- bool operator() (const expression_walker &_a, unsigned int _b) const
- bool operator() (unsigned int _a, const expression_walker &_b) const
- bool operator() (const expression_walker &_a, const expression_walker &_b) const

10.3.1 Detailed Description

Compare a node with an `unsigned int` (node number) for sorting certain node vectors.

10.3.2 Member Function Documentation

10.3.2.1 bool coco::dag_delta::__docompare_nodes::operator() (const expression_walker & *_a*, unsigned int *_b*) const [inline]

Definition at line 270 of file dag_delta.h.

10.3.2.2 bool coco::dag_delta::__docompare_nodes::operator() (unsigned int *_a*, const expression_walker & *_b*) const [inline]

Definition at line 272 of file dag_delta.h.

10.3.2.3 bool coco::dag_delta::__docompare_nodes::operator() (const expression_walker & _a, const expression_walker & _b) const [inline]

Definition at line 274 of file dag_delta.h.

The documentation for this struct was generated from the following file:

- dag_delta.h

10.4 coco::model::__docompare_nodes Struct Reference

`#include <model.hpp>`

Public Member Functions

- bool operator() (const model::walker &_a, unsigned int _b) const
- bool operator() (unsigned int _a, const model::walker &_b) const
- bool operator() (const model::walker &_a, const model::walker &_b) const

10.4.1 Detailed Description

This function class is used to compare nodes for sorting and binary search.

10.4.2 Member Function Documentation

10.4.2.1 bool coco::model::__docompare_nodes::operator() (const model::walker & _a, unsigned int _b) const [inline]

Definition at line 1836 of file model.hpp.

10.4.2.2 bool coco::model::__docompare_nodes::operator() (unsigned int _a, const model::walker & _b) const [inline]

Definition at line 1838 of file model.hpp.

10.4.2.3 bool coco::model::__docompare_nodes::operator() (const model::walker & _a, const model::walker & _b) const [inline]

Definition at line 1840 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

10.5 coco::model::__docompare_variables Struct Reference

`#include <model.hpp>`

Public Member Functions

- bool operator() (const model::walker &_a, unsigned int _b) const
- bool operator() (unsigned int _a, const model::walker &_b) const
- bool operator() (const model::walker &_a, const model::walker &_b) const

10.5.1 Detailed Description

This function class is used to compare variables for sorting and binary search.

10.5.2 Member Function Documentation

10.5.2.1 bool coco::model::__docompare_variables::operator() (const model::walker & _a, unsigned int _b) const [inline]

Definition at line 1848 of file model.hpp.

10.5.2.2 bool coco::model::__docompare_variables::operator() (unsigned int _a, const model::walker & _b) const [inline]

Definition at line 1850 of file model.hpp.

10.5.2.3 bool coco::model::__docompare_variables::operator() (const model::walker & _a, const model::walker & _b) const [inline]

Definition at line 1852 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

10.6 coco::__sg_anc_visitor Class Reference

Public Member Functions

- __sg_anc_visitor (std::set< search_node_id > &anc)
- ~__sg_anc_visitor ()

- void vinit ()
- void vcollect (return_value const &)
- return_value vvalue ()
- bool preorder (search_node *const &r)
- bool postorder (search_node *const &r)
- void collect (search_node *const &, return_value const &ancestor)
- return_value value ()

10.6.1 Constructor & Destructor Documentation

10.6.1.1 coco::__sg_anc_visitor::__sg_anc_visitor (std::set< search_node_id > & *anc*) [inline]

Constructor

Definition at line 56 of file search_graph.cc.

10.6.1.2 coco::__sg_anc_visitor::~__sg_anc_visitor ( ) [inline]

Destructor

Definition at line 59 of file search_graph.cc.

10.6.2 Member Function Documentation

10.6.2.1 void coco::__sg_anc_visitor::collect (search_node *const & , return_value const & *ancestor*) [inline]

This method is required by a prepost_visitor.

Definition at line 94 of file search_graph.cc.

10.6.2.2 bool coco::__sg_anc_visitor::postorder (search_node *const & *r*) [inline]

This method is required by a prepost_visitor.

Definition at line 88 of file search_graph.cc.

10.6.2.3 bool coco::__sg_anc_visitor::preorder (search_node *const & *r*) [inline]

This method is required by a prepost_visitor.

Definition at line 81 of file search_graph.cc.

10.6.2.4 return_value coco::__sg_anc_visitor::value ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 99 of file search_graph.cc.

10.6.2.5 void coco::__sg_anc_visitor::vcollect (return_value const &) [inline]

This method is required by a prepost_visitor.

Definition at line 68 of file search_graph.cc.

10.6.2.6 void coco::__sg_anc_visitor::vinit ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 63 of file search_graph.cc.

10.6.2.7 return_value coco::__sg_anc_visitor::vvalue ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 75 of file search_graph.cc.

The documentation for this class was generated from the following file:

- search_graph.cc

10.7 coco::_evaluator_base Class Reference

Base class of all evaluators.

`#include <evaluator.h>`

Public Types

- typedef _Tp data_type
- typedef _NData node_data_type
- typedef _Result return_value
- typedef _Walker const_walker

Public Member Functions

- _evaluator_base ()
- _evaluator_base (const _Tp &__x)
- _evaluator_base (const _Self &__x)
- virtual ~_evaluator_base ()
- virtual return_value vvalue ()
- virtual return_value value ()
- virtual int vcollect (const return_value &__cresult)
- virtual int collect (const node_data_type &__data, const return_value &__cresult)
- virtual void postorder (const node_data_type &__data)

Protected Attributes

- _Tp eval_data

10.7.1 Detailed Description

This class is the base class of all evaluators. Basically, it is a visitor to expression_node nodes in a DAG.

10.7.2 Member Typedef Documentation

10.7.2.1 typedef _Walker coco::_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Definition at line 312 of file evaluator.h.

10.7.2.2 typedef _Tp coco::_evaluator_base::data_type

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.7.2.3 typedef _NData coco::_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Definition at line 307 of file evaluator.h.

10.7.2.4 typedef _Result coco::_evaluator_base::return_value

This type is the result type of the evaluator.

Definition at line 310 of file evaluator.h.

10.7.3 Constructor & Destructor Documentation

10.7.3.1 coco::_evaluator_base::_evaluator_base ( ) [inline]

Standard Constructor

Definition at line 320 of file evaluator.h.

10.7.3.2 coco::_evaluator_base::_evaluator_base (const _Tp & __x) [inline]

Constructor, which initializes the internal data

Definition at line 322 of file evaluator.h.

10.7.3.3 coco::_evaluator_base::_evaluator_base (const _Self & __x) [inline]

Standard Copy Constructor

Definition at line 324 of file evaluator.h.

10.7.3.4 virtual coco::_evaluator_base::~_evaluator_base ( ) [inline, virtual]

Standard Destructor

Definition at line 327 of file evaluator.h.

10.7.4 Member Function Documentation

10.7.4.1 virtual int coco::_evaluator_base::collect (const node_data_type & __data, const return_value & __cresult) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. The return value has the following effect-
:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 359 of file evaluator.h.

10.7.4.2 virtual void coco::_evaluator_base::postorder (const node_data_type & *__data*) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited.

Definition at line 364 of file evaluator.h.

10.7.4.3 virtual return_value coco::_evaluator_base::value ( ) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value.

Definition at line 336 of file evaluator.h.

10.7.4.4 virtual int coco::_evaluator_base::vcollect (const return_value & *__cresult*) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 348 of file evaluator.h.

10.7.4.5 virtual return_value coco::_evaluator_base::vvalue ( ) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value.

Definition at line 332 of file evaluator.h.

10.7.5 Member Data Documentation

10.7.5.1 _Tp coco::_evaluator_base::eval_data [protected]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

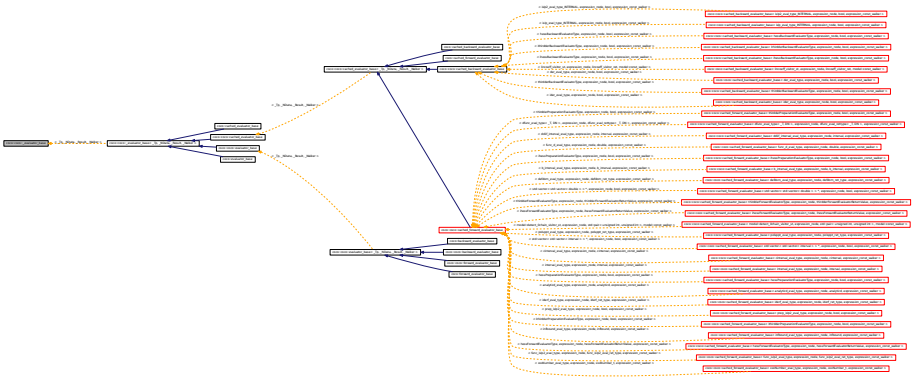
The documentation for this class was generated from the following file:

- evaluator.h

10.8 coco::coco::_evaluator_base Class Reference

Base class of all evaluators.

Inheritance diagram for coco::coco::_evaluator_base:



Public Types

- typedef _Tp data_type
- typedef _NData node_data_type
- typedef _Result return_value
- typedef _Walker const_walker

Public Member Functions

- _evaluator_base ()
- _evaluator_base (const _Tp &__x)
- _evaluator_base (const _Self &__x)
- virtual ~_evaluator_base ()
- virtual return_value vvalue ()
- virtual return_value value ()
- virtual int vcollect (const return_value &__cresult)
- virtual int collect (const node_data_type &__data, const return_value &__cresult)
- virtual void postorder (const node_data_type &__data)

Protected Attributes

- _Tp eval_data

10.8.1 Detailed Description

This class is the base class of all evaluators. Basically, it is a visitor to expression_node nodes in a DAG.

10.8.2 Member Typedef Documentation

10.8.2.1 typedef _Walker coco::coco::_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented in coco::coco::cached_evaluator_base, coco::cached_evaluator_base, coco::coco::evaluator_base, and coco::evaluator_base.

Definition at line 313 of file search_graph.cc.

10.8.2.2 typedef _Tp coco::coco::_evaluator_base::data_type

The data_type specifies the type of the internal data of the evaluator.

Definition at line 306 of file search_graph.cc.

10.8.2.3 typedef _NData coco::coco::_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented in coco::coco::cached_evaluator_base, coco::cached_evaluator_base, coco::coco::evaluator_base, and coco::evaluator_base.

Definition at line 308 of file search_graph.cc.

10.8.2.4 typedef _Result coco::coco::_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented in coco::coco::cached_evaluator_base, coco::cached_evaluator_base, coco::coco::evaluator_base, and coco::evaluator_base.

Definition at line 311 of file search_graph.cc.

10.8.3 Constructor & Destructor Documentation

10.8.3.1 coco::coco::_evaluator_base::_evaluator_base ( ) [inline]

Standard Constructor

Definition at line 321 of file search_graph.cc.

10.8.3.2 coco::coco::_evaluator_base::_evaluator_base (const _Tp & __x) [inline]

Constructor, which initializes the internal data

Definition at line 323 of file search_graph.cc.

10.8.3.3 coco::coco::_evaluator_base::_evaluator_base (const _Self & __x) [inline]

Standard Copy Constructor

Definition at line 325 of file search_graph.cc.

10.8.3.4 virtual coco::coco::_evaluator_base::~_evaluator_base ( ) [inline, virtual]

Standard Destructor

Definition at line 328 of file search_graph.cc.

10.8.4 Member Function Documentation

10.8.4.1 virtual int coco::coco::_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__cresult*) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 360 of file search_graph.cc.

10.8.4.2 virtual void coco::coco::_evaluator_base::postorder (const node_data_type & *__data*) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited.

Definition at line 365 of file search_graph.cc.

10.8.4.3 virtual return_value coco::coco::_evaluator_base::value ( ) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value.

Definition at line 337 of file search_graph.cc.

10.8.4.4 virtual int coco::coco::_evaluator_base::vcollect (const return_value & *__cresult*) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 349 of file search_graph.cc.

10.8.4.5 virtual return_value coco::coco::_evaluator_base::vvalue ( ) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value.

Definition at line 333 of file search_graph.cc.

10.8.5 Member Data Documentation

10.8.5.1 _Tp coco::coco::_evaluator_base::eval_data [protected]

The internal data of the evaluator

Definition at line 317 of file search_graph.cc.

The documentation for this class was generated from the following file:

- evaluator.h

10.9 coco::analyticd_eval Class Reference

`#include <ade_evaluator.h>`

Inheritance diagram for coco::analyticd_eval:

Collaboration diagram for coco::analyticd_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- analyticd_eval (const std::vector< analyticd > &__x, const variable_indicator &__v, const model &__m, std::vector< analyticd > *__c)
- analyticd_eval (const analyticd_eval &__v)
- ~analyticd_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const analyticd &__rval)
- int update (const expression_node &__data, const analyticd &__rval)
- analyticd calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.9.1 Member Typedef Documentation

10.9.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.9.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.9.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 coco::analyticd_eval::analyticd_eval (const std::vector< analyticd > & __x, const variable_indicator & __v, const model & __m, std::vector< analyticd > * __c) [inline]

Definition at line 116 of file ade_evaluator.h.

10.9.2.2 coco::analyticd_eval::analyticd_eval (const analyticd_eval & __v) [inline]

Definition at line 128 of file ade_evaluator.h.

10.9.2.3 coco::analyticd_eval::~analyticd_eval ( ) [inline]

Definition at line 130 of file ade_evaluator.h.

10.9.3 Member Function Documentation

10.9.3.1 void coco::analyticd_eval::calculate (const expression_node & __data) [inline]

Definition at line 193 of file ade_evaluator.h.

10.9.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.9.3.3 analyticd coco::analyticd_eval::calculate_value (bool eval_all) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 707 of file ade_evaluator.h.

10.9.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.9.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.9.3.6 void coco::analyticd_eval::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 135 of file ade_evaluator.h.

10.9.3.7 int coco::analyticd_eval::initialize (const expression_node & *__data*) [inline]

Definition at line 137 of file ade_evaluator.h.

10.9.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.9.3.9 bool coco::analyticd_eval::is_cached (const node_data_type & *__data*) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 70 of file ade_evaluator.h.

10.9.3.10 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.9.3.11 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.9.3.12 void coco::analyticd_eval::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 205 of file ade_evaluator.h.

10.9.3.13 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.9.3.14 expression_const_walker coco::analyticd_eval::short_cut_to (const expression_node & __data) [inline]

Definition at line 132 of file ade_evaluator.h.

10.9.3.15 int coco::analyticd_eval::update (const analyticd & __rval) [inline]

Definition at line 221 of file ade_evaluator.h.

10.9.3.16 int coco::analyticd_eval::update (const expression_node & __data, const analyticd & __rval) [inline]

Definition at line 227 of file ade_evaluator.h.

10.9.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.9.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.9.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.9.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.9.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.9.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

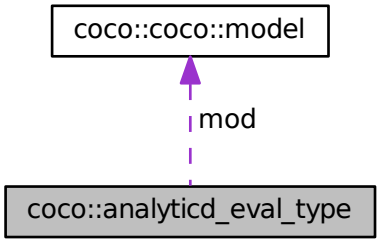
The documentation for this class was generated from the following file:

- ade_evaluator.h

10.10 coco::analyticd_eval_type Struct Reference

`#include <ade_evaluator.h>`

Collaboration diagram for coco::analyticd_eval_type:



Public Attributes

- const std::vector< analyticd > * x
- std::vector< analyticd > * cache
- const model * mod
- void * p
- analyticd d
- int info
- analyticd r
- unsigned int n

10.10.1 Member Data Documentation

10.10.1.1 std::vector<analyticd>* coco::analyticd_eval_type::cache

Definition at line 52 of file ade_evaluator.h.

10.10.1.2 analyticd coco::analyticd_eval_type::d

Definition at line 55 of file ade_evaluator.h.

10.10.1.3 int coco::analyticd_eval_type::info

Definition at line 56 of file ade_evaluator.h.

10.10.1.4 const model* coco::analyticd_eval_type::mod

Definition at line 53 of file ade_evaluator.h.

10.10.1.5 unsigned int coco::analyticd_eval_type::n

Definition at line 58 of file ade_evaluator.h.

10.10.1.6 void* coco::analyticd_eval_type::p

Definition at line 54 of file ade_evaluator.h.

10.10.1.7 analyticd coco::analyticd_eval_type::r

Definition at line 57 of file ade_evaluator.h.

10.10.1.8 const std::vector<analyticd>* coco::analyticd_eval_type::x

Definition at line 51 of file ade_evaluator.h.

The documentation for this struct was generated from the following file:

- ade_evaluator.h

10.11 coco::coco::annotation Class Reference

Annotations for Models.

Public Member Functions

- annotation (const vdbl::tableid &_ti, const vdbl::rowid &_ri)
- virtual ~annotation ()
- vdbl::tableid get_table () const
- vdbl::rowid get_entry () const
- annotation (const vdbl::tableid &_ti, const vdbl::rowid &_ri)
- virtual ~annotation ()
- vdbl::tableid get_table () const
- vdbl::rowid get_entry () const

10.11.1 Detailed Description

The annotation class serves as a means to connect a model to its relevant database information. Annotations are stored within a work_node (resp. full_node).

Definition at line 47 of file search_graph.cc.

10.11.2 Constructor & Destructor Documentation

10.11.2.1 coco::coco::annotation::annotation (const vdbl::tableid & *_ti*, const vdbl::rowid & *_ri*) [inline]

standard constructor

Definition at line 55 of file search_graph.cc.

10.11.2.2 virtual coco::coco::annotation::~annotation ( ) [inline, virtual]

standard destructor

Definition at line 58 of file search_graph.cc.

10.11.2.3 coco::coco::annotation::annotation (const vdbl::tableid & *_ti*, const vdbl::rowid & *_ri*) [inline]

standard constructor

Definition at line 55 of file search_graph.cc.

10.11.2.4 virtual coco::coco::annotation::~annotation ( ) [inline, virtual]

standard destructor

Definition at line 58 of file search_graph.cc.

10.11.3 Member Function Documentation

10.11.3.1 vdbl::rowid coco::coco::annotation::get_entry ( ) const [inline]

retrieve the row_id part from the annotation

Definition at line 63 of file search_graph.cc.

10.11.3.2 vdbl::rowid coco::coco::annotation::get_entry ( ) const [inline]

retrieve the row_id part from the annotation

Definition at line 63 of file search_graph.cc.

10.11.3.3 vdbl::tableid coco::coco::annotation::get_table ( ) const [inline]

retrieve the table_id part from the annotation

Definition at line 61 of file search_graph.cc.

10.11.3.4 vdbl::tableid coco::coco::annotation::get_table ( ) const [inline]

retrieve the table_id part from the annotation

Definition at line 61 of file search_graph.cc.

The documentation for this class was generated from the following file:

- annotation.h

10.12 coco::annotation Class Reference

Annotations for Models.

`#include <annotation.h>`

Public Member Functions

- annotation (const vdbl::tableid &_ti, const vdbl::rowid &_ri)
- virtual ~annotation ()
- vdbl::tableid get_table () const
- vdbl::rowid get_entry () const

10.12.1 Detailed Description

The annotation class serves as a means to connect a model to its relevant database information. Annotations are stored within a work_node (resp. full_node).

10.12.2 Constructor & Destructor Documentation

10.12.2.1 coco::annotation::annotation (const vdbl::tableid & *_ti,* const vdbl::rowid & *_ri*) [inline]

standard constructor

Definition at line 55 of file annotation.h.

10.12.2.2 virtual coco::annotation::~annotation ( ) [inline, virtual]

standard destructor

Definition at line 58 of file annotation.h.

10.12.3 Member Function Documentation

10.12.3.1 vdbl::rowid coco::annotation::get_entry ( ) const [inline]

retrieve the row_id part from the annotation

Definition at line 63 of file annotation.h.

10.12.3.2 vdbl::tableid coco::annotation::get_table ( ) const [inline]

retrieve the table_id part from the annotation

Definition at line 61 of file annotation.h.

The documentation for this class was generated from the following file:

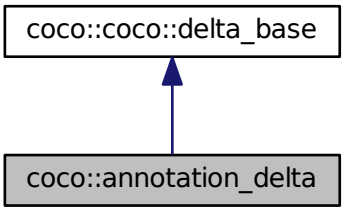
- annotation.h

10.13 coco::annotation_delta Class Reference

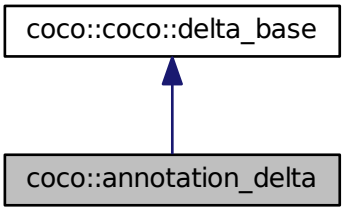
the delta class for annotation changes

`#include <annotation_delta.h>`

Inheritance diagram for coco::annotation_delta:



Collaboration diagram for coco::annotation_delta:



Public Member Functions

- annotation_delta (const std::string &_act)
- annotation_delta (const std::string &_act, const std::vector< annotation > &__a, const std::vector< annotation > &__r)
- annotation_delta (const std::string &_act, const annotation &_ad)
- annotation_delta (const std::string &_act, bool _dummy, const annotation &_rm)
- annotation_delta (const annotation_delta &__d)
- annotation_delta (const char *_act, const std::vector< annotation > &__a, const std::vector< annotation > &__r)
- annotation_delta (const char *_act, const annotation &_ad)
- annotation_delta (const char *_act, bool _dummy, const annotation &_rm)
- virtual annotation_delta * new_copy () const
- virtual void destroy_copy (delta_base *__d) const

- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const annotation_delta &_c) const
- bool operator!= (const annotation_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- const std::string & get_action () const
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual void unkeep ()
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Public Attributes

- std::vector< annotation > add
- std::vector< annotation > rm

Protected Attributes

- std::string _action

10.13.1 Detailed Description

The annotation_delta class describes the apply information for changes to the annotation structure in the work_node.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 coco::annotation_delta::annotation_delta (const std::string & *_act*) [inline]

standard constructor: `_act` describes the action specifier of the delta

Definition at line 107 of file annotation_delta.h.

10.13.2.2 coco::annotation_delta::annotation_delta (const std::string & _act, const std::vector< annotation > & __a, const std::vector< annotation > & __r) [inline]

constructor: _act describes the action specifier of the delta, the annotations ___a are added, the annotations ___r removed.

Definition at line 112 of file annotation_delta.h.

10.13.2.3 coco::annotation_delta::annotation_delta (const std::string & _act, const annotation & _ad) [inline]

constructor: _act describes the action specifier of the delta, the annotation _ad is added, nothing is removed.

Definition at line 120 of file annotation_delta.h.

10.13.2.4 coco::annotation_delta::annotation_delta (const std::string & _act, bool _dummy, const annotation & _rm) [inline]

constructor: _act describes the action specifier of the delta, the annotation _rm is removed, nothing is added. The parameter _dummy is ignored.

Definition at line 127 of file annotation_delta.h.

10.13.2.5 coco::annotation_delta::annotation_delta (const annotation_delta & __d) [inline]

standard copy constructor

Definition at line 131 of file annotation_delta.h.

10.13.2.6 coco::annotation_delta::annotation_delta (const char * _act, const std::vector< annotation > & __a, const std::vector< annotation > & __r) [inline]

constructor: _act describes the action specifier of the delta, the annotations ___a are added, the annotations ___r removed.

Definition at line 142 of file annotation_delta.h.

10.13.2.7 coco::annotation_delta::annotation_delta (const char * _act, const annotation & _ad) [inline]

constructor: _act describes the action specifier of the delta, the annotation _ad is added, nothing is removed.

Definition at line 150 of file annotation_delta.h.

10.13.2.8 coco::annotation_delta::annotation_delta (const char * _act, bool _dummy, const annotation & _rm) [inline]

constructor: _act describes the action specifier of the delta, the annotation _rm is removed, nothing is added. The parameter _dummy is ignored.

Definition at line 157 of file annotation_delta.h.

10.13.3 Member Function Documentation

10.13.3.1 bool coco::annotation_delta::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_did*, size_t & *delta_size*) const [virtual]

apply operation: apply the delta with identity _did to work_node _x and construct in _u the undelta information.

Definition at line 35 of file annotation_delta.cc.

10.13.3.2 virtual bool coco::coco::delta_base::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_di*, size_t & *delta_size*) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.13.3.3 virtual bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.13.3.4 bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.13.3.5 virtual bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.13.3.6 virtual void coco::coco::delta_base::convert (work_node & *_x*, delta_base *& *_d*) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.13.3.7 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.13.3.8 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.13.3.9 virtual void coco::annotation_delta::destroy_copy (delta_base * __d) const [inline, virtual]

clone destructor

Definition at line 164 of file annotation_delta.h.

10.13.3.10 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.13.3.11 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.13.3.12 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.13.3.13 delta coco::coco::delta_base::make_delta (const std::string & a) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.13.3.14 delta coco::coco::delta_base::make_delta (const std::string & a) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.13.3.15 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.13.3.16 virtual annotation_delta* coco::annotation_delta::new_copy ( ) const [inline, virtual]

clone operation

Reimplemented from coco::coco::delta_base.

Definition at line 161 of file annotation_delta.h.

10.13.3.17 bool coco::annotation_delta::operator!= (const delta_base & *_c*) const [inline]

Definition at line 177 of file annotation_delta.h.

10.13.3.18 bool coco::annotation_delta::operator!= (const annotation_delta & *_c*) const [inline]

Definition at line 184 of file annotation_delta.h.

10.13.3.19 bool coco::annotation_delta::operator== (const delta_base & *_c*) const [inline]

Comparison operators

Definition at line 174 of file annotation_delta.h.

10.13.3.20 bool coco::annotation_delta::operator== (const annotation_delta & *_c*) const [inline]

Comparison operators

Definition at line 182 of file annotation_delta.h.

10.13.3.21 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.13.3.22 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.13.3.23 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.13.4 Member Data Documentation

10.13.4.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

10.13.4.2 std::vector<annotation> coco::annotation_delta::add

the annotations to be added to the work_node

Definition at line 99 of file annotation_delta.h.

10.13.4.3 std::vector<annotation> coco::annotation_delta::rm

the annotations to be removed from the work_node

Definition at line 101 of file annotation_delta.h.

The documentation for this class was generated from the following files:

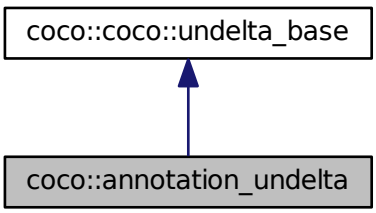
- annotation_delta.h
- annotation_delta.cc

10.14 coco::annotation_undelta Class Reference

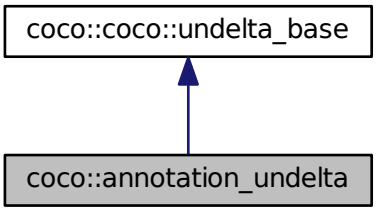
the undelta class for annotation changes

`#include <annotation_delta.h>`

Inheritance diagram for coco::annotation_undelta:



Collaboration diagram for coco::annotation_undelta:



Public Member Functions

- annotation_undelta (const std::vector< annotation > &__a, const std::vector< annotation > &__r)
- annotation_undelta (const std::vector< annotation > &__a)
- annotation_undelta (const annotation &__a)
- annotation_undelta (const annotation_undelta &__d)
- annotation_undelta * new_copy () const
- void destroy_copy (undelta_base *__d) const
- bool unapply (work_node &_x, const delta_id &_did) const
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

Public Attributes

- std::vector< annotation > added_ann
- std::vector< annotation > removed_ann

Friends

- class annotation_delta

10.14.1 Detailed Description

The annotation_undelta class describes the unapply information for changes to the annotation structure in the work_node.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 coco::annotation_undelta::annotation_undelta (const std::vector< annotation > & __a, const std::vector< annotation > & __r) [inline]

constructor explicitly specifying the lists ___a of added annotations and ___r of removed annotations

Definition at line 51 of file annotation_delta.h.

10.14.2.2 coco::annotation_undelta::annotation_undelta (const std::vector< annotation > & __a) [inline]

constructor explicitly specifying the lists ___a of added annotations, keeping the removed annotations empty

Definition at line 58 of file annotation_delta.h.

10.14.2.3 coco::annotation_undelta::annotation_undelta (const annotation & __a) [inline]

constructor for a single added annotation

Definition at line 63 of file annotation_delta.h.

10.14.2.4 coco::annotation_undelta::annotation_undelta (const annotation_undelta & __d) [inline]

standard copy constructor

Definition at line 68 of file annotation_delta.h.

10.14.3 Member Function Documentation

10.14.3.1 void coco::annotation_undelta::destroy_copy (undelta_base * __d) const [inline]

clone destructor

Definition at line 80 of file annotation_delta.h.

10.14.3.2 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.14.3.3 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.14.3.4 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.14.3.5 annotation_undelta* coco::annotation_undelta::new_copy ( ) const [inline, virtual]

cloning operation

Reimplemented from coco::coco::undelta_base.

Definition at line 78 of file annotation_delta.h.

10.14.3.6 bool coco::annotation_undelta::unapply (work_node & _x, const delta_id & _did) const [virtual]

unapply operation: undoing the delta with identity _did from work_node _x

Reimplemented from coco::coco::undelta_base.

Definition at line 63 of file annotation_delta.cc.

10.14.3.7 virtual bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.14.3.8 virtual bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.14.3.9 bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [inline, virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

Definition at line 94 of file api_delta.h.

10.14.4 Friends And Related Function Documentation

10.14.4.1 friend class annotation_delta [friend]

Definition at line 86 of file annotation_delta.h.

10.14.5 Member Data Documentation

10.14.5.1 std::vector<annotation> coco::annotation_undelta::added_ann

these are the annotations added by the corresponding annotation_delta

Definition at line 44 of file annotation_delta.h.

10.14.5.2 std::vector<annotation> coco::annotation_undelta::removed_ann

these are the annotations removed by the corresponding annotation_delta

Definition at line 46 of file annotation_delta.h.

The documentation for this class was generated from the following files:

- annotation_delta.h
- annotation_delta.cc

10.15 coco::api_exception Class Reference

API exception class.

`#include <api_exception.h>`

Public Member Functions

- api_exception (api_exception_type a, char const *m)
- api_exception (api_exception_type a, const std::string &m)
- virtual ~api_exception () throw ()
- virtual char const * what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.15.1 Detailed Description

This is the base class for all exceptions which are thrown by the API and the various modules of the COCONUT environment. Every properly coded COCONUT module should only throw this exception or some of its subclasses.

10.15.2 Constructor & Destructor Documentation

10.15.2.1 coco::api_exception::api_exception (api_exception_type *a*, char const * *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 79 of file api_exception.h.

10.15.2.2 coco::api_exception::api_exception (api_exception_type *a*, const std::string & *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 82 of file api_exception.h.

10.15.2.3 virtual coco::api_exception::~api_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 86 of file api_exception.h.

10.15.3 Member Function Documentation

10.15.3.1 virtual std::string coco::api_exception::message ( ) const throw () [inline, virtual]

This method returns the message as C++-string.

Definition at line 104 of file api_exception.h.

10.15.3.2 virtual api_exception_type coco::api_exception::type ( ) const throw () [inline, virtual]

This method returns the exception type as enum value.

Definition at line 95 of file api_exception.h.

10.15.3.3 virtual const char* coco::api_exception::type_str ( ) const throw () [virtual]

This method returns the exception type as C-string.

10.15.3.4 virtual char const* coco::api_exception::what ( ) const throw () [inline, virtual]

This method returns the message as C-string.

Definition at line 89 of file api_exception.h.

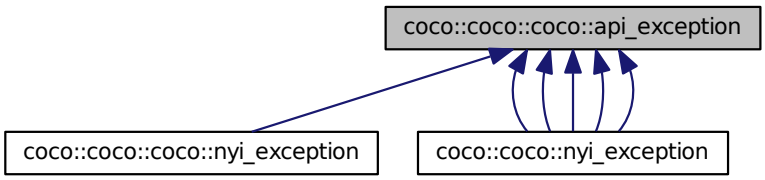
The documentation for this class was generated from the following file:

- api_exception.h

10.16 coco::coco::coco::api_exception Class Reference

API exception class.

Inheritance diagram for coco::coco::coco::api_exception:



Public Member Functions

- api_exception (api_exception_type a, char const ∗m)
- api_exception (api_exception_type a, const std::string &m)
- virtual ∼api_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual std::string message () const throw ()

10.16.1 Detailed Description

This is the base class for all exceptions which are thrown by the API and the various modules of the COCONUT environment. Every properly coded COCONUT module should only throw this exception or some of its subclasses.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 coco::coco::coco::api_exception::api_exception (api_exception_type *a*, char const ∗ *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 79 of file search_graph.cc.

10.16.2.2 coco::coco::coco::api_exception::api_exception (api_exception_type *a*, const std::string & *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 82 of file search_graph.cc.

10.16.2.3 virtual coco::coco::coco::api_exception::~api_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 86 of file search_graph.cc.

10.16.3 Member Function Documentation

10.16.3.1 virtual std::string coco::coco::coco::api_exception::message ( ) const throw () [inline, virtual]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.16.3.2 virtual api_exception_type coco::coco::coco::api_exception::type ( ) const throw () [inline, virtual]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.16.3.3 virtual const char* coco::coco::coco::api_exception::type_str ( ) const throw () [virtual]

This method returns the exception type as C-string.

10.16.3.4 virtual char const* coco::coco::coco::api_exception::what ( ) const throw () [inline, virtual]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following file:

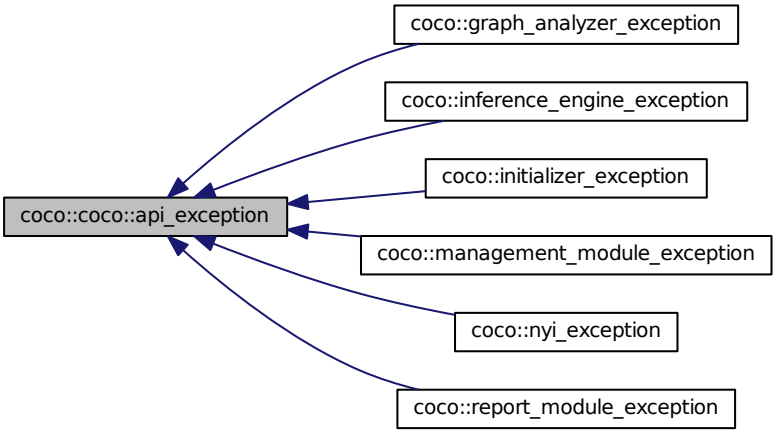
- api_exception.h

10.17 coco::coco::api_exception Class Reference

API exception class.

`#include <expression.h>`

Inheritance diagram for coco::coco::api_exception:



Public Member Functions

- api_exception (api_exception_type a, char const ∗m)
- api_exception (api_exception_type a, const std::string &m)
- virtual ∼api_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual std::string message () const throw ()
- api_exception (api_exception_type a, char const ∗m)
- api_exception (api_exception_type a, const std::string &m)
- virtual ∼api_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual std::string message () const throw ()
- api_exception (api_exception_type a, char const ∗m)
- api_exception (api_exception_type a, const std::string &m)
- virtual ∼api_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual std::string message () const throw ()
- api_exception (api_exception_type a, char const ∗m)
- api_exception (api_exception_type a, const std::string &m)
- virtual ∼api_exception () throw ()
- virtual char const ∗ what () const throw ()

- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()
- api_exception (api_exception_type a, char const *m)
- api_exception (api_exception_type a, const std::string &m)
- virtual ~api_exception () throw ()
- virtual char const * what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.17.1 Detailed Description

This is the base class for all exceptions which are thrown by the API and the various modules of the COCONUT environment. Every properly coded COCONUT module should only throw this exception or some of its subclasses.

Definition at line 69 of file search_graph.cc.

10.17.2 Constructor & Destructor Documentation

10.17.2.1 coco::coco::api_exception::api_exception (api_exception_type *a*, char const * *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 79 of file expression.h.

10.17.2.2 coco::coco::api_exception::api_exception (api_exception_type *a*, const std::string & *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 82 of file expression.h.

10.17.2.3 virtual coco::coco::api_exception::~api_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 86 of file expression.h.

10.17.2.4 coco::coco::api_exception::api_exception (api_exception_type *a*, char const * *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 79 of file search_graph.cc.

10.17.2.5 coco::coco::api_exception::api_exception (api_exception_type *a*, const std::string & *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 82 of file search_graph.cc.

10.17.2.6 virtual coco::coco::api_exception::~api_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 86 of file search_graph.cc.

10.17.2.7 coco::coco::api_exception::api_exception (api_exception_type *a,* char const * *m*) [inline]

Constructor, setting the type to a and the message to m

Definition at line 79 of file search_graph.cc.

10.17.2.8 coco::coco::api_exception::api_exception (api_exception_type *a,* const std::string & *m*) [inline]

Constructor, setting the type to a and the message to m

Definition at line 82 of file search_graph.cc.

10.17.2.9 virtual coco::coco::api_exception::~api_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 86 of file search_graph.cc.

10.17.2.10 coco::coco::api_exception::api_exception (api_exception_type *a,* char const * *m*) [inline]

Constructor, setting the type to a and the message to m

Definition at line 79 of file search_graph.cc.

10.17.2.11 coco::coco::api_exception::api_exception (api_exception_type *a,* const std::string & *m*) [inline]

Constructor, setting the type to a and the message to m

Definition at line 82 of file search_graph.cc.

10.17.2.12 virtual coco::coco::api_exception::~api_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 86 of file search_graph.cc.

10.17.2.13 coco::coco::api_exception::api_exception (api_exception_type *a,* char const * *m*) [inline]

Constructor, setting the type to a and the message to m

Definition at line 79 of file search_graph.cc.

10.17.2.14 coco::coco::api_exception::api_exception (api_exception_type *a*, const std::string & *m*) [inline]

Constructor, setting the type to `a` and the message to `m`

Definition at line 82 of file search_graph.cc.

10.17.2.15 virtual coco::coco::api_exception::~api_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 86 of file search_graph.cc.

10.17.3 Member Function Documentation

10.17.3.1 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual]

This method returns the message as C++-string.

Definition at line 104 of file expression.h.

10.17.3.2 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.17.3.3 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.17.3.4 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.17.3.5 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.17.3.6 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual]

This method returns the exception type as enum value.

Definition at line 95 of file expression.h.

10.17.3.7 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.17.3.8 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.17.3.9 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.17.3.10 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.17.3.11 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual]

This method returns the exception type as C-string.

10.17.3.12 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual]

This method returns the exception type as C-string.

10.17.3.13 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual]

This method returns the exception type as C-string.

10.17.3.14 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual]

This method returns the exception type as C-string.

10.17.3.15 const char * coco::api_exception::type_str ( ) const throw () [virtual]

This method returns the exception type as C-string.

Definition at line 57 of file api_exception.cc.

10.17.3.16 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.17.3.17 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.17.3.18 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual]

This method returns the message as C-string.

Definition at line 89 of file expression.h.

10.17.3.19 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.17.3.20 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following files:

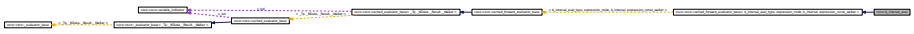
- api_exception.h
- api_exception.cc

10.18 coco::b_interval_eval Class Reference

`#include <bint_evaluator.h>`

Inheritance diagram for coco::b_interval_eval:

Collaboration diagram for coco::b_interval_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- b_interval_eval (const std::vector< b_interval > &__x, const variable_indicator &__v, const model &__m, std::vector< b_interval > *__c)
- b_interval_eval (const b_interval_eval &__v)
- ~b_interval_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const b_interval &__rval)
- int update (const expression_node &__data, const b_interval &__rval)
- b_interval calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.18.1 Member Typedef Documentation

10.18.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.18.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.18.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 coco::b_interval_eval::b_interval_eval (const std::vector< b_interval > & __x, const variable_indicator & __v, const model & __m, std::vector< b_interval > * __c) [inline]

Definition at line 116 of file bint_evaluator.h.

10.18.2.2 coco::b_interval_eval::b_interval_eval (const b_interval_eval & __v) [inline]

Definition at line 128 of file bint_evaluator.h.

10.18.2.3 coco::b_interval_eval::~b_interval_eval ( ) [inline]

Definition at line 130 of file bint_evaluator.h.

10.18.3 Member Function Documentation

10.18.3.1 void coco::b_interval_eval::calculate (const expression_node & __data) [inline]

Definition at line 193 of file bint_evaluator.h.

10.18.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.18.3.3 b_interval coco::b_interval_eval::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 707 of file bint_evaluator.h.

10.18.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.18.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.18.3.6 void coco::b_interval_eval::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 135 of file bint_evaluator.h.

10.18.3.7 int coco::b_interval_eval::initialize (const expression_node & *__data*) [inline]

Definition at line 137 of file bint_evaluator.h.

10.18.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The `__data` parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.18.3.9 bool coco::b_interval_eval::is_cached (const node_data_type & *__data*) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 70 of file bint_evaluator.h.

10.18.3.10 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.18.3.11 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.18.3.12 void coco::b_interval_eval::retrieve_from_cache (const expression_node & *__data*) [inline]

Definition at line 205 of file bint_evaluator.h.

10.18.3.13 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.18.3.14 expression_const_walker coco::b_interval_eval::short_cut_to (const expression_node & *__data*) [inline]

Definition at line 132 of file bint_evaluator.h.

10.18.3.15 int coco::b_interval_eval::update (const b_interval & *__rval*) [inline]

Definition at line 221 of file bint_evaluator.h.

10.18.3.16 int coco::b_interval_eval::update (const expression_node & *__data*, const b_interval & *__rval*) [inline]

Definition at line 227 of file bint_evaluator.h.

10.18.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.18.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.18.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter false.

Definition at line 763 of file search_graph.cc.

10.18.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.18.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.18.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

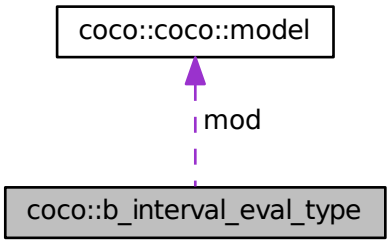
The documentation for this class was generated from the following file:

- bint_evaluator.h

10.19 coco::b_interval_eval_type Struct Reference

`#include <bint_evaluator.h>`

Collaboration diagram for coco::b_interval_eval_type:



Public Attributes

- const std::vector< b_interval > * x
- std::vector< b_interval > * cache
- const model * mod
- void * p
- b_interval d
- int info
- b_interval r
- unsigned int n

10.19.1 Member Data Documentation

10.19.1.1 std::vector<b_interval>* coco::b_interval_eval_type::cache

Definition at line 52 of file bint_evaluator.h.

10.19.1.2 b_interval coco::b_interval_eval_type::d

Definition at line 55 of file bint_evaluator.h.

10.19.1.3 int coco::b_interval_eval_type::info

Definition at line 56 of file bint_evaluator.h.

10.19.1.4 const model* coco::b_interval_eval_type::mod

Definition at line 53 of file bint_evaluator.h.

10.19.1.5 unsigned int coco::b_interval_eval_type::n

Definition at line 58 of file bint_evaluator.h.

10.19.1.6 void* coco::b_interval_eval_type::p

Definition at line 54 of file bint_evaluator.h.

10.19.1.7 b_interval coco::b_interval_eval_type::r

Definition at line 57 of file bint_evaluator.h.

10.19.1.8 const std::vector<b_interval>* coco::b_interval_eval_type::x

Definition at line 51 of file bint_evaluator.h.

The documentation for this struct was generated from the following file:

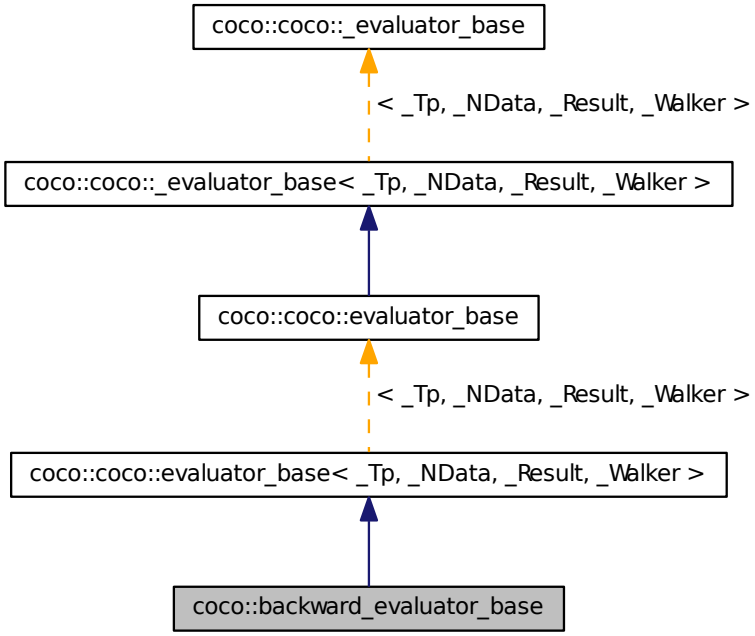
- bint_evaluator.h

10.20 coco::backward_evaluator_base Class Reference

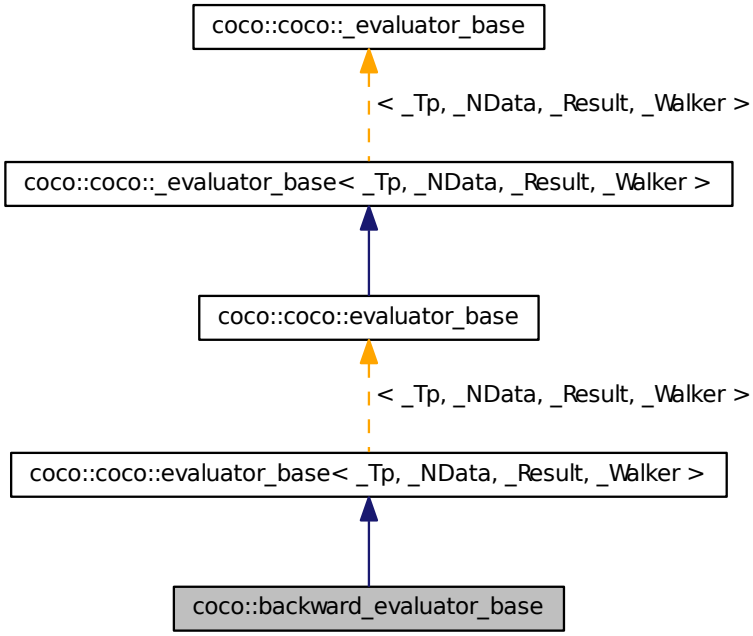
Base class of all (non-caching) backward evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::backward_evaluator_base:



Collaboration diagram for coco::backward_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- backward_evaluator_base ()
- backward_evaluator_base (const _Tp &__x)
- backward_evaluator_base (const _Self &__x)
- virtual ~backward_evaluator_base ()
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual void initialize ()

- virtual void initialize (const node_data_type &__data)
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual return_value calculate_value (bool eval_all)
- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)

10.20.1 Detailed Description

This class is the base class of all non-caching backward evaluators. Basically, it is a visitor to expression_-node nodes in a DAG, based on evaluator_base.

10.20.2 Member Typedef Documentation

10.20.2.1 typedef _Base::const_walker coco::backward_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 601 of file evaluator.h.

10.20.2.2 typedef _Base::node_data_type coco::backward_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 597 of file evaluator.h.

10.20.2.3 typedef _Base::return_value coco::backward_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 599 of file evaluator.h.

10.20.3 Constructor & Destructor Documentation

10.20.3.1 coco::backward_evaluator_base::backward_evaluator_base ( ) [inline]

Standard Constructor

Definition at line 606 of file evaluator.h.

10.20.3.2 coco::backward_evaluator_base::backward_evaluator_base (const _Tp & __x) [inline]

Constructor

Definition at line 608 of file evaluator.h.

10.20.3.3 coco::backward_evaluator_base::backward_evaluator_base (const _Self & __x) [inline]

Standard Copy Constructor

Definition at line 610 of file evaluator.h.

10.20.3.4 virtual coco::backward_evaluator_base::~backward_evaluator_base ( ) [inline, virtual]

Standard Destructor

Definition at line 612 of file evaluator.h.

10.20.4 Member Function Documentation

10.20.4.1 virtual int coco::backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 669 of file evaluator.h.

10.20.4.2 virtual return_value coco::backward_evaluator_base::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Definition at line 700 of file evaluator.h.

10.20.4.3 virtual void coco::backward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 673 of file evaluator.h.

10.20.4.4 int coco::backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 628 of file evaluator.h.

10.20.4.5 virtual void coco::backward_evaluator_base::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Definition at line 654 of file evaluator.h.

10.20.4.6 virtual void coco::backward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 658 of file evaluator.h.

10.20.4.7 void coco::backward_evaluator_base::postorder (const node_data_type & *__data*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 623 of file evaluator.h.

10.20.4.8 virtual int coco::coco::evaluator_base::preorder (const node_data_type & *__data*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 402 of file search_graph.cc.

10.20.4.9 int coco::backward_evaluator_base::preorder (const node_data_type & *__data*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It is translated into calls to initialize and calculate.

Definition at line 618 of file evaluator.h.

10.20.4.10 virtual const_walker coco::coco::evaluator_base::short_cut_to (const node_data_type & *__data*) [inline, virtual, inherited]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the const_walker returned is the target of the short-cut.

Definition at line 406 of file search_graph.cc.

10.20.4.11 virtual int coco::backward_evaluator_base::update (const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The

return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 684 of file evaluator.h.

10.20.4.12 virtual int coco::backward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 695 of file evaluator.h.

10.20.4.13 return_value coco::backward_evaluator_base::value ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 640 of file evaluator.h.

10.20.4.14 int coco::backward_evaluator_base::vcollect (const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 634 of file evaluator.h.

10.20.4.15 void coco::backward_evaluator_base::vinit ( ) [inline]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 649 of file evaluator.h.

10.20.4.16 return_value coco::backward_evaluator_base::vvalue ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 645 of file evaluator.h.

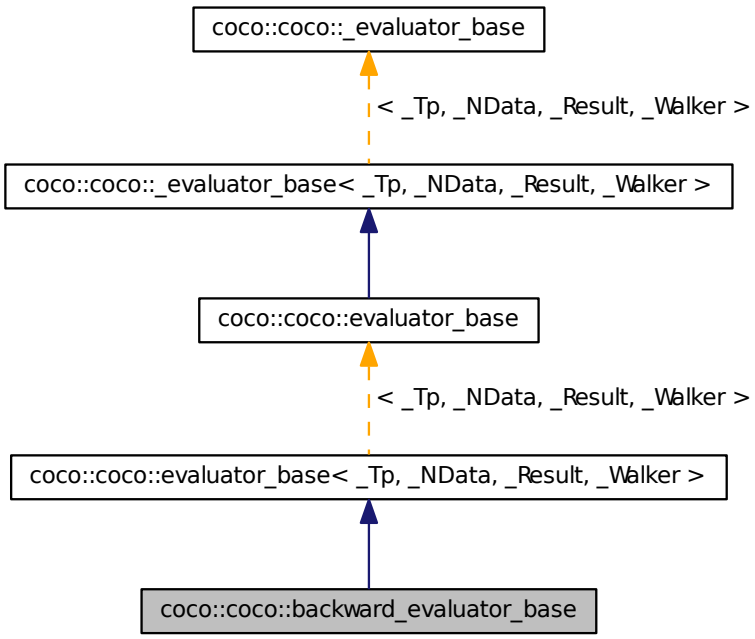
The documentation for this class was generated from the following file:

- evaluator.h

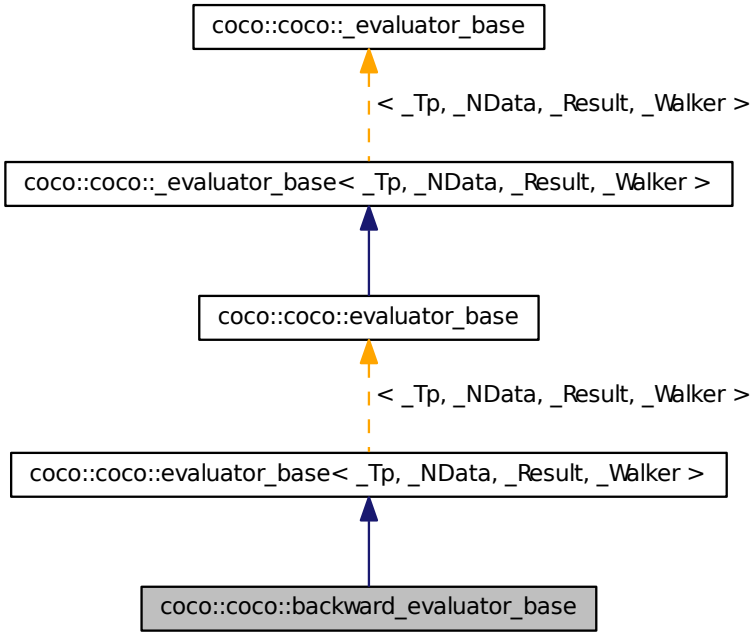
10.21 coco::coco::backward_evaluator_base Class Reference

Base class of all (non-caching) backward evaluators.

Inheritance diagram for coco::coco::backward_evaluator_base:



Collaboration diagram for coco::coco::backward_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- backward_evaluator_base ()
- backward_evaluator_base (const _Tp &__x)
- backward_evaluator_base (const _Self &__x)
- virtual ~backward_evaluator_base ()
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual void initialize ()

- virtual void initialize (const node_data_type &__data)
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual return_value calculate_value (bool eval_all)
- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)

10.21.1 Detailed Description

This class is the base class of all non-caching backward evaluators. Basically, it is a visitor to expression_-node nodes in a DAG, based on evaluator_base.

10.21.2 Member Typedef Documentation

10.21.2.1 typedef _Base::const_walker coco::coco::backward_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 602 of file search_graph.cc.

10.21.2.2 typedef _Base::node_data_type coco::coco::backward_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 598 of file search_graph.cc.

10.21.2.3 typedef _Base::return_value coco::coco::backward_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 600 of file search_graph.cc.

10.21.3 Constructor & Destructor Documentation

10.21.3.1 coco::coco::backward_evaluator_base::backward_evaluator_base ( ) [inline]

Standard Constructor

Definition at line 607 of file search_graph.cc.

10.21.3.2 coco::coco::backward_evaluator_base::backward_evaluator_base (const _Tp & __x) [inline]

Constructor

Definition at line 609 of file search_graph.cc.

10.21.3.3 coco::coco::backward_evaluator_base::backward_evaluator_base (const _Self & __x) [inline]

Standard Copy Constructor

Definition at line 611 of file search_graph.cc.

10.21.3.4 virtual coco::coco::backward_evaluator_base::~backward_evaluator_base ( ) [inline, virtual]

Standard Destructor

Definition at line 613 of file search_graph.cc.

10.21.4 Member Function Documentation

10.21.4.1 virtual int coco::coco::backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 670 of file search_graph.cc.

10.21.4.2 virtual return_value coco::coco::backward_evaluator_base::calculate_value (bool eval_all) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Definition at line 701 of file search_graph.cc.

10.21.4.3 virtual void coco::coco::backward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 674 of file search_graph.cc.

10.21.4.4 int coco::coco::backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 629 of file search_graph.cc.

10.21.4.5 virtual void coco::coco::backward_evaluator_base::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Definition at line 655 of file search_graph.cc.

10.21.4.6 virtual void coco::coco::backward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 659 of file search_graph.cc.

10.21.4.7 void coco::coco::backward_evaluator_base::postorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 624 of file search_graph.cc.

10.21.4.8 virtual int coco::coco::evaluator_base::preorder (const node_data_type & __data) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 402 of file search_graph.cc.

10.21.4.9 int coco::coco::backward_evaluator_base::preorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It is translated into calls to initialize and calculate.

Definition at line 619 of file search_graph.cc.

10.21.4.10 virtual const_walker coco::coco::evaluator_base::short_cut_to (const node_data_type & *__data*) [inline, virtual, inherited]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 406 of file search_graph.cc.

10.21.4.11 virtual int coco::coco::backward_evaluator_base::update (const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 685 of file search_graph.cc.

10.21.4.12 virtual int coco::coco::backward_evaluator_base::update (const node_data_type & *__data,* const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 696 of file search_graph.cc.

10.21.4.13 return_value coco::coco::backward_evaluator_base::value ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 641 of file search_graph.cc.

10.21.4.14 int coco::coco::backward_evaluator_base::vcollect (const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 635 of file search_graph.cc.

10.21.4.15 void coco::coco::backward_evaluator_base::vinit ( ) [inline]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 650 of file search_graph.cc.

10.21.4.16 return_value coco::coco::backward_evaluator_base::vvalue ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 646 of file search_graph.cc.

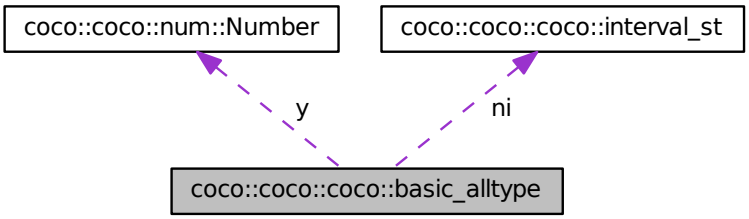
The documentation for this class was generated from the following file:

- evaluator.h

10.22 coco::coco::coco::basic_alltype Class Reference

The basic alltype which can hold any of a number of basic types.

Collaboration diagram for coco::coco::coco::basic_alltype:



Public Member Functions

- basic_alltype ()
- basic_alltype (bool __x)
- basic_alltype (int __x)
- basic_alltype (unsigned int __x)
- basic_alltype (double __x)
- basic_alltype (interval __x)
- basic_alltype (void *__x)
- basic_alltype (const char *__cp)
- basic_alltype (const std::string &__x)

- basic_alltype (const num::Number &__y)
- basic_alltype (const std::vector< bool > &__x)
- basic_alltype (const std::vector< int > &__x)
- basic_alltype (const std::vector< unsigned int > &__x)
- basic_alltype (const std::vector< double > &__x)
- basic_alltype (const std::vector< interval > &__x)
- basic_alltype (const std::vector< std::string > &__x)
- basic_alltype (const std::vector< num::Number > &__x)
- basic_alltype (const std::vector< std::vector< bool > > &__vx)
- basic_alltype (const std::vector< std::vector< int > > &__vx)
- basic_alltype (const std::vector< std::vector< unsigned int > > &__vx)
- basic_alltype (const std::vector< std::vector< double > > &__vx)
- basic_alltype (const std::vector< std::vector< interval > > &__vx)
- basic_alltype (const std::vector< std::vector< std::string > > &__vx)
- basic_alltype (const std::vector< std::vector< num::Number > > &__vx)
- basic_alltype (const vmtl::sparse_vector< bool > &__x)
- basic_alltype (const vmtl::sparse_vector< int > &__x)
- basic_alltype (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype (const vmtl::sparse_vector< double > &__x)
- basic_alltype (const vmtl::sparse_vector< interval > &__x)
- basic_alltype (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype (const vmtl::dense_matrix< double > &__x)
- basic_alltype (const vmtl::dense_matrix< int > &__x)
- basic_alltype (const vmtl::dense_matrix< interval > &__x)
- basic_alltype (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype (const vmtl::sparse_matrix< double > &__x)
- basic_alltype (const vmtl::sparse_matrix< int > &__x)
- basic_alltype (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype (const vmtl::sparse_matrix< num::Number > &__x)
- ~basic_alltype ()
- basic_alltype (const basic_alltype &__a)
- basic_alltype & operator= (bool __x)
- basic_alltype & operator= (int __x)
- basic_alltype & operator= (unsigned int __x)
- basic_alltype & operator= (double __x)
- basic_alltype & operator= (interval __x)
- basic_alltype & operator= (void *__x)
- basic_alltype & operator= (const std::string &__x)
- basic_alltype & operator= (const char *__x)
- basic_alltype & operator= (const num::Number &__x)
- basic_alltype & operator= (const std::vector< bool > &__x)
- basic_alltype & operator= (const std::vector< int > &__x)
- basic_alltype & operator= (const std::vector< unsigned int > &__x)
- basic_alltype & operator= (const std::vector< double > &__x)
- basic_alltype & operator= (const std::vector< interval > &__x)
- basic_alltype & operator= (const std::vector< std::string > &__x)
- basic_alltype & operator= (const std::vector< num::Number > &__x)

- basic_alltype & operator= (const std::vector< std::vector< bool > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< unsigned int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< double > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< interval > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< std::string > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< num::Number > > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< bool > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< num::Number > &__x)
- basic_alltype & operator= (const basic_alltype &__a)
- basic_alltype & clear ()
- basic_alltype & set_dm (vmtl::dense_matrix< double > *__m)
- basic_alltype & set_ndm (vmtl::dense_matrix< int > *__m)
- basic_alltype & set_idm (vmtl::dense_matrix< interval > *__m)
- basic_alltype & set_cdm (vmtl::dense_matrix< std::string > *__m)
- basic_alltype & set_xdm (vmtl::dense_matrix< num::Number > *__m)
- basic_alltype & set_sm (vmtl::sparse_matrix< double > *__m)
- basic_alltype & set_nsm (vmtl::sparse_matrix< int > *__m)
- basic_alltype & set_ism (vmtl::sparse_matrix< interval > *__m)
- basic_alltype & set_csm (vmtl::sparse_matrix< std::string > *__m)
- basic_alltype & set_xsm (vmtl::sparse_matrix< num::Number > *__m)
- bool nb () const BASIC_ALLTYPE_THROW
- int nn () const BASIC_ALLTYPE_THROW
- unsigned int nu () const BASIC_ALLTYPE_THROW
- double nd () const BASIC_ALLTYPE_THROW
- interval ni () const BASIC_ALLTYPE_THROW
- const void * p () const BASIC_ALLTYPE_THROW
- const std::string & s () const BASIC_ALLTYPE_THROW
- const num::Number & y () const BASIC_ALLTYPE_THROW
- const std::vector< bool > & b () const BASIC_ALLTYPE_THROW
- const std::vector< int > & n () const BASIC_ALLTYPE_THROW
- const std::vector< unsigned int > & u () const BASIC_ALLTYPE_THROW
- const std::vector< double > & d () const BASIC_ALLTYPE_THROW
- const std::vector< interval > & i () const BASIC_ALLTYPE_THROW
- const std::vector< std::string > & c () const BASIC_ALLTYPE_THROW

- const std::vector< num::Number > & x () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < bool > > & vb () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < int > > & vn () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < unsigned int > > & vu () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < double > > & vd () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < interval > > & vi () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < std::string > > & vc () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < num::Number > > & vx () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< bool > & sb () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< int > & sn () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < unsigned int > & su () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < double > & sd () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < interval > & si () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < std::string > & sc () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < num::Number > & sx () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < double > & dm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix< int > & ndm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < interval > & idm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < std::string > & cdm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < num::Number > & xdm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < double > & sm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix< int > & nsm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < interval > & ism () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < std::string > & csm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < num::Number > & xsm () const BASIC_ALLTYPE_THROW
- bool & nb () BASIC_ALLTYPE_THROW
- int & nn () BASIC_ALLTYPE_THROW
- unsigned int & nu () BASIC_ALLTYPE_THROW
- double & nd () BASIC_ALLTYPE_THROW
- interval_st & ni () BASIC_ALLTYPE_THROW
- void *& p () BASIC_ALLTYPE_THROW
- std::string & s () BASIC_ALLTYPE_THROW
- num::Number & y () BASIC_ALLTYPE_THROW
- std::vector< bool > & b () BASIC_ALLTYPE_THROW
- std::vector< int > & n () BASIC_ALLTYPE_THROW
- std::vector< unsigned int > & u () BASIC_ALLTYPE_THROW
- std::vector< double > & d () BASIC_ALLTYPE_THROW
- std::vector< interval > & i () BASIC_ALLTYPE_THROW
- std::vector< std::string > & c () BASIC_ALLTYPE_THROW
- std::vector< num::Number > & x () BASIC_ALLTYPE_THROW
- std::vector< std::vector< bool > > & vb () BASIC_ALLTYPE_THROW
- std::vector< std::vector< int > > & vn () BASIC_ALLTYPE_THROW
- std::vector< std::vector < unsigned int > > & vu () BASIC_ALLTYPE_THROW
- std::vector< std::vector < double > > & vd () BASIC_ALLTYPE_THROW
- std::vector< std::vector < interval > > & vi () BASIC_ALLTYPE_THROW
- std::vector< std::vector < std::string > > & vc () BASIC_ALLTYPE_THROW
- std::vector< std::vector < num::Number > > & vx () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< bool > & sb () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< int > & sn () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< unsigned int > & su () BASIC_ALLTYPE_THROW

- vmtl::sparse_vector< double > & sd () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< interval > & si () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < std::string > & sc () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < num::Number > & sx () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< double > & dm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< int > & ndm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< interval > & idm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< std::string > & cdm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< num::Number > & xdm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< double > & sm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< int > & nsm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< interval > & ism () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < std::string > & csm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < num::Number > & xsm () BASIC_ALLTYPE_THROW
- bool is_allocated () const
- bool is_scalar () const
- bool is_vector () const
- bool is_vector_of_vector () const
- bool is_dense_vector () const
- bool is_sparse_vector () const
- bool is_empty_vector () const
- bool is_matrix () const
- bool is_dense_matrix () const
- bool is_sparse_matrix () const
- bool is_empty_matrix () const
- bool empty () const
- int contents_type () const
- std::string type_name () const
- const char * type_cstr () const
- bool operator== (const basic_alltype &b) const
- bool operator!= (const basic_alltype &b) const
- bool less_than (const basic_alltype &b) const
- bool less_equal (const basic_alltype &b) const

10.22.1 Detailed Description

This class is the base of the data handling in the API. It can hold values of different types: bool, int, unsigned int, double, interval, string, Number, vector<bool>, vector<int>, vector<unsigned int>, vector<double>, vector<interval>, vector<string>, vector<Number>, sparse_vector<bool>, sparse_vector<int>, sparse_vector<unsigned int>, sparse_vector<double>, sparse_vector<interval>, sparse_vector<string>, sparse_vector<Number>, dense_matrix<double>, dense_matrix<int>, dense_matrix<interval>, dense_matrix<string>, dense_matrix<Number>, sparse_matrix<double>, sparse_matrix<int>, sparse_matrix<interval>, sparse_matrix<string>, sparse_matrix<Number>. All of these types can be stored and retrieved conveniently.

10.22.2 Constructor & Destructor Documentation

10.22.2.1 coco::coco::coco::basic_alltype::basic_alltype ( ) [inline]

Standard Constructor yielding an empty basic_alltype

Definition at line 480 of file search_graph.cc.

10.22.2.2 coco::coco::coco::basic_alltype::basic_alltype (bool __x) [inline]

Constructor storing the bool ___x

Definition at line 482 of file search_graph.cc.

10.22.2.3 coco::coco::coco::basic_alltype::basic_alltype (int __x) [inline]

Constructor storing the int ___x

Definition at line 485 of file search_graph.cc.

10.22.2.4 coco::coco::coco::basic_alltype::basic_alltype (unsigned int __x) [inline]

Constructor storing the unsigned int ___x

Definition at line 488 of file search_graph.cc.

10.22.2.5 coco::coco::coco::basic_alltype::basic_alltype (double __x) [inline]

Constructor storing the double ___x

Definition at line 491 of file search_graph.cc.

10.22.2.6 coco::coco::coco::basic_alltype::basic_alltype (interval __x) [inline]

Constructor storing the interval ___x

Definition at line 494 of file search_graph.cc.

10.22.2.7 coco::coco::coco::basic_alltype::basic_alltype (void * __x) [inline]

Constructor storing the void* ___x

Definition at line 497 of file search_graph.cc.

10.22.2.8 coco::coco::coco::basic_alltype::basic_alltype (const char * __cp) [inline]

Constructor storing the string ___cp

Definition at line 500 of file search_graph.cc.

10.22.2.9 coco::coco::coco::basic_alltype::basic_alltype (const std::string & __x) [inline]

Constructor storing the string ___x

Definition at line 503 of file search_graph.cc.

10.22.2.10 coco::coco::coco::basic_alltype::basic_alltype (const num::Number & __y) [inline]

Constructor storing the number ___y

Definition at line 506 of file search_graph.cc.

10.22.2.11 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< bool > & __x) [inline]

Constructor storing the vector<bool> __x

Definition at line 510 of file search_graph.cc.

10.22.2.12 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< int > & __x) [inline]

Constructor storing the vector<int> __x

Definition at line 514 of file search_graph.cc.

10.22.2.13 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< unsigned int > & __x) [inline]

Constructor storing the vector<unsigned int> __x

Definition at line 518 of file search_graph.cc.

10.22.2.14 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< double > & __x) [inline]

Constructor storing the vector<double> __x

Definition at line 522 of file search_graph.cc.

10.22.2.15 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< interval > & __x) [inline]

Constructor storing the vector<interval> __x

Definition at line 526 of file search_graph.cc.

10.22.2.16 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< std::string > & __x) [inline]

Constructor storing the vector<string> __x

Definition at line 530 of file search_graph.cc.

10.22.2.17 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< num::Number > & __x) [inline]

Constructor storing the vector<Number> __x

Definition at line 534 of file search_graph.cc.

10.22.2.18 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< bool > > & __vx) [inline]

Constructor storing the vector<vector<bool> > __vx

Definition at line 539 of file search_graph.cc.

10.22.2.19 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< int > > & __vx) [inline]

Constructor storing the vector<vector<int> > __vx

Definition at line 543 of file search_graph.cc.

10.22.2.20 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< unsigned int > > & __vx) [inline]

Constructor storing the vector<vector<unsigned int> > __vx

Definition at line 547 of file search_graph.cc.

10.22.2.21 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< double > > & __vx) [inline]

Constructor storing the vector<vector<double> > __vx

Definition at line 551 of file search_graph.cc.

10.22.2.22 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< interval > > & __vx) [inline]

Constructor storing the vector<vector<interval> > __vx

Definition at line 555 of file search_graph.cc.

10.22.2.23 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< std::string > > & __vx) [inline]

Constructor storing the vector<vector<string> > __vx

Definition at line 559 of file search_graph.cc.

10.22.2.24 coco::coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< num::Number > > & __vx) [inline]

Constructor storing the vector<vector<Number> > __vx

Definition at line 563 of file search_graph.cc.

10.22.2.25 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< bool > & __x) [inline]

Constructor storing the sparse_vector<bool> __x

Definition at line 568 of file search_graph.cc.

10.22.2.26 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< int > & __x) [inline]

Constructor storing the sparse_vector<int> __x

Definition at line 572 of file search_graph.cc.

10.22.2.27 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Constructor storing the sparse_vector<unsigned int> __x

Definition at line 576 of file search_graph.cc.

10.22.2.28 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< double > & __x) [inline]

Constructor storing the sparse_vector<double> __x

Definition at line 580 of file search_graph.cc.

10.22.2.29 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< interval > & __x) [inline]

Constructor storing the sparse_vector<interval> __x

Definition at line 584 of file search_graph.cc.

10.22.2.30 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< std::string > & __x) [inline]

Constructor storing the sparse_vector<string> __x

Definition at line 588 of file search_graph.cc.

10.22.2.31 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< num::Number > & __x) [inline]

Constructor storing the sparse_vector<Number> __x

Definition at line 592 of file search_graph.cc.

10.22.2.32 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< double > & __x) [inline]

Constructor storing the dense_matrix<double> __x

Definition at line 597 of file search_graph.cc.

10.22.2.33 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< int > & __x) [inline]

Constructor storing the dense_matrix<int> __x

Definition at line 601 of file search_graph.cc.

10.22.2.34 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< interval > & __x) [inline]

Constructor storing the dense_matrix<interval> __x

Definition at line 605 of file search_graph.cc.

10.22.2.35 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< std::string > & __x) [inline]

Constructor storing the dense_matrix<string> __x

Definition at line 609 of file search_graph.cc.

10.22.2.36 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< num::Number > & __x) [inline]

Constructor storing the dense_matrix<Number> __x

Definition at line 613 of file search_graph.cc.

10.22.2.37 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< double > & __x) [inline]

Constructor storing the sparse_matrix<double> __x

Definition at line 618 of file search_graph.cc.

10.22.2.38 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< int > & __x) [inline]

Constructor storing the sparse_matrix<int> __x

Definition at line 622 of file search_graph.cc.

10.22.2.39 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< interval > & __x) [inline]

Constructor storing the sparse_matrix<interval> __x

Definition at line 626 of file search_graph.cc.

10.22.2.40 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< std::string > & __x) [inline]

Constructor storing the sparse_matrix<string> __x

Definition at line 630 of file search_graph.cc.

10.22.2.41 coco::coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Constructor storing the sparse_matrix<Number> __x

Definition at line 634 of file search_graph.cc.

10.22.2.42 coco::coco::coco::basic_alltype::~basic_alltype ( ) [inline]

Standard Destructor

Definition at line 639 of file search_graph.cc.

10.22.2.43 coco::coco::coco::basic_alltype::basic_alltype (const basic_alltype & *__a*) [inline]

Standard Copy Constructor

Definition at line 643 of file search_graph.cc.

10.22.3 Member Function Documentation

10.22.3.1 const std::vector<bool>& coco::coco::coco::basic_alltype::b ( ) const [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1085 of file search_graph.cc.

10.22.3.2 std::vector<bool>& coco::coco::coco::basic_alltype::b ( ) [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1357 of file search_graph.cc.

10.22.3.3 const std::vector<std::string>& coco::coco::coco::basic_alltype::c ( ) const [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1141 of file search_graph.cc.

10.22.3.4 std::vector<std::string>& coco::coco::coco::basic_alltype::c ( ) [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1413 of file search_graph.cc.

10.22.3.5 const vmtl::dense_matrix<std::string>& coco::coco::coco::basic_alltype::cdm ( ) const [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1298 of file search_graph.cc.

10.22.3.6 vmtl::dense_matrix<std::string>& coco::coco::coco::basic_alltype::cdm ( ) [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1569 of file search_graph.cc.

10.22.3.7 basic_alltype& coco::coco::coco::basic_alltype::clear ( ) [inline]

Reset the basic_alltype to an empty value

Definition at line 929 of file search_graph.cc.

10.22.3.8 int coco::coco::coco::basic_alltype::contents_type ( ) const [inline]

Return the type of the value contained in the storage, an int which can be compared with the ALLTYPE_... values.

Definition at line 1669 of file search_graph.cc.

10.22.3.9 const vmtl::sparse_matrix<std::string>& coco::coco::coco::basic_alltype::csm ( ) const [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1314 of file search_graph.cc.

10.22.3.10 vmtl::sparse_matrix<std::string>& coco::coco::coco::basic_alltype::csm ( ) [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1585 of file search_graph.cc.

10.22.3.11 const std::vector<double>& coco::coco::coco::basic_alltype::d ( ) const [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1133 of file search_graph.cc.

10.22.3.12 std::vector<double>& coco::coco::coco::basic_alltype::d ( ) [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1405 of file search_graph.cc.

10.22.3.13 const vmtl::dense_matrix<double>& coco::coco::coco::basic_alltype::dm ( ) const [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1289 of file search_graph.cc.

10.22.3.14 vmtl::dense_matrix<double>& coco::coco::coco::basic_alltype::dm ( ) [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1560 of file search_graph.cc.

10.22.3.15 bool coco::coco::coco::basic_alltype::empty ( ) const [inline]

Check, whether the basic_alltype is empty.

Definition at line 1666 of file search_graph.cc.

10.22.3.16 const std::vector<interval>& coco::coco::coco::basic_alltype::i ( ) const [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1137 of file search_graph.cc.

10.22.3.17 std::vector<interval>& coco::coco::coco::basic_alltype::i ( ) [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1409 of file search_graph.cc.

10.22.3.18 const vmtl::dense_matrix<interval>& coco::coco::coco::basic_alltype::idm ( ) const [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1295 of file search_graph.cc.

10.22.3.19 vmtl::dense_matrix<interval>& coco::coco::coco::basic_alltype::idm ( ) [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1566 of file search_graph.cc.

10.22.3.20 bool coco::coco::coco::basic_alltype::is_allocated ( ) const [inline]

Check, whether the basic_alltype contains a complex type, i.e. one which has to be allocated (not bool, int, unsigned int, double, or interval).

Definition at line 1595 of file search_graph.cc.

10.22.3.21 bool coco::coco::coco::basic_alltype::is_dense_matrix ( ) const [inline]

Check, whether the basic_alltype contains a dense matrix type.

Definition at line 1644 of file search_graph.cc.

10.22.3.22 bool coco::coco::coco::basic_alltype::is_dense_vector ( ) const [inline]

Check, whether the basic_alltype contains a dense vector type.

Definition at line 1607 of file search_graph.cc.

10.22.3.23 bool coco::coco::coco::basic_alltype::is_empty_matrix ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1648 of file search_graph.cc.

10.22.3.24 bool coco::coco::coco::basic_alltype::is_empty_vector ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1613 of file search_graph.cc.

10.22.3.25 bool coco::coco::coco::basic_alltype::is_matrix ( ) const [inline]

Check, whether the basic_alltype contains a matrix type.

Definition at line 1642 of file search_graph.cc.

10.22.3.26 bool coco::coco::coco::basic_alltype::is_scalar ( ) const [inline]

Check, whether the basic_alltype contains a scalar type, i.e. neither a vector nor a matrix.

Definition at line 1598 of file search_graph.cc.

10.22.3.27 bool coco::coco::coco::basic_alltype::is_sparse_matrix ( ) const [inline]

Check, whether the basic_alltype contains a sparse matrix type.

Definition at line 1646 of file search_graph.cc.

10.22.3.28 bool coco::coco::coco::basic_alltype::is_sparse_vector ( ) const [inline]

Check, whether the basic_alltype contains a sparse vector type.

Definition at line 1610 of file search_graph.cc.

10.22.3.29 bool coco::coco::coco::basic_alltype::is_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector type.

Definition at line 1601 of file search_graph.cc.

10.22.3.30 bool coco::coco::coco::basic_alltype::is_vector_of_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector of vector type.

Definition at line 1604 of file search_graph.cc.

10.22.3.31 const vmtl::sparse_matrix<interval>& coco::coco::coco::basic_alltype::ism ( ) const [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1311 of file search_graph.cc.

10.22.3.32 vmtl::sparse_matrix<interval>& coco::coco::coco::basic_alltype::ism ( ) [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1582 of file search_graph.cc.

10.22.3.33 bool coco::coco::coco::basic_alltype::less_equal (const basic_alltype & *b*) const

Total order comparison operator

10.22.3.34 bool coco::coco::coco::basic_alltype::less_than (const basic_alltype & *b*) const

Total order comparison operator

10.22.3.35 const std::vector<int>& coco::coco::coco::basic_alltype::n ( ) const [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1089 of file search_graph.cc.

10.22.3.36 std::vector<int>& coco::coco::coco::basic_alltype::n ( ) [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1361 of file search_graph.cc.

10.22.3.37 bool coco::coco::coco::basic_alltype::nb ( ) const [inline]

retrieve a bool from the basic_alltype

Definition at line 1049 of file search_graph.cc.

10.22.3.38 bool& coco::coco::coco::basic_alltype::nb ( ) [inline]

retrieve a bool from the basic_alltype

Definition at line 1321 of file search_graph.cc.

10.22.3.39 double coco::coco::coco::basic_alltype::nd ( ) const [inline]

retrieve a double from the basic_alltype

Definition at line 1070 of file search_graph.cc.

10.22.3.40 double& coco::coco::coco::basic_alltype::nd ( ) [inline]

retrieve a double from the basic_alltype

Definition at line 1342 of file search_graph.cc.

10.22.3.41 const vmtl::dense_matrix<int>& coco::coco::coco::basic_alltype::ndm ( ) const [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1292 of file search_graph.cc.

10.22.3.42 vmtl::dense_matrix<int>& coco::coco::coco::basic_alltype::ndm ( ) [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1563 of file search_graph.cc.

10.22.3.43 interval coco::coco::coco::basic_alltype::ni ( ) const [inline]

retrieve an interval from the basic_alltype

Definition at line 1073 of file search_graph.cc.

10.22.3.44 interval_st& coco::coco::coco::basic_alltype::ni ( ) [inline]

retrieve an interval from the basic_alltype

Definition at line 1345 of file search_graph.cc.

10.22.3.45 int coco::coco::coco::basic_alltype::nn ( ) const [inline]

retrieve an int from the basic_alltype

Definition at line 1052 of file search_graph.cc.

10.22.3.46 int& coco::coco::coco::basic_alltype::nn ( ) [inline]

retrieve an int from the basic_alltype

Definition at line 1324 of file search_graph.cc.

10.22.3.47 const vmtl::sparse_matrix<int>& coco::coco::coco::basic_alltype::nsm ( ) const [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1308 of file search_graph.cc.

10.22.3.48 vmtl::sparse_matrix<int>& coco::coco::coco::basic_alltype::nsm ( ) [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1579 of file search_graph.cc.

10.22.3.49 unsigned int coco::coco::coco::basic_alltype::nu ( ) const [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1061 of file search_graph.cc.

10.22.3.50 unsigned int& coco::coco::coco::basic_alltype::nu ( ) [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1333 of file search_graph.cc.

10.22.3.51 bool coco::coco::coco::basic_alltype::operator!= (const basic_alltype & *b*) const

Standard comparison operator

10.22.3.52 basic_alltype& coco::coco::coco::basic_alltype::operator= (bool *__x*) [inline]

Assignment Operator, assigning the bool value __x

Definition at line 647 of file search_graph.cc.

10.22.3.53 basic_alltype& coco::coco::coco::basic_alltype::operator= (int *__x*) [inline]

Assignment Operator, assigning the int value __x

Definition at line 653 of file search_graph.cc.

10.22.3.54 basic_alltype& coco::coco::coco::basic_alltype::operator= (unsigned int *__x*) [inline]

Assignment Operator, assigning the unsigned int value __x

Definition at line 659 of file search_graph.cc.

10.22.3.55 basic_alltype& coco::coco::coco::basic_alltype::operator= (double __x) [inline]

Assignment Operator, assigning the double value __x

Definition at line 665 of file search_graph.cc.

10.22.3.56 basic_alltype& coco::coco::coco::basic_alltype::operator= (interval __x) [inline]

Assignment Operator, assigning the interval value __x

Definition at line 671 of file search_graph.cc.

10.22.3.57 basic_alltype& coco::coco::coco::basic_alltype::operator= (void * __x) [inline]

Assignment Operator, assigning the void* value __x

Definition at line 677 of file search_graph.cc.

10.22.3.58 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::string & __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 683 of file search_graph.cc.

10.22.3.59 basic_alltype& coco::coco::coco::basic_alltype::operator= (const char * __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 690 of file search_graph.cc.

10.22.3.60 basic_alltype& coco::coco::coco::basic_alltype::operator= (const num::Number & __x) [inline]

Assignment Operator, assigning the Number value __x

Definition at line 697 of file search_graph.cc.

10.22.3.61 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< bool > & __x) [inline]

Assignment Operator, assigning the vector<bool> value __x

Definition at line 704 of file search_graph.cc.

10.22.3.62 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< int > & __x) [inline]

Assignment Operator, assigning the vector<int> value __x

Definition at line 711 of file search_graph.cc.

10.22.3.63 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the vector<unsigned int> value __x

Definition at line 718 of file search_graph.cc.

10.22.3.64 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< double > & __x) [inline]

Assignment Operator, assigning the vector<double> value __x

Definition at line 725 of file search_graph.cc.

10.22.3.65 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< interval > & __x) [inline]

Assignment Operator, assigning the vector<interval> value __x

Definition at line 732 of file search_graph.cc.

10.22.3.66 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< std::string > & __x) [inline]

Assignment Operator, assigning the vector<string> value __x

Definition at line 739 of file search_graph.cc.

10.22.3.67 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< num::Number > & __x) [inline]

Assignment Operator, assigning the vector<Number> value __x

Definition at line 746 of file search_graph.cc.

10.22.3.68 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< std::vector< bool > > & __x) [inline]

Assignment Operator, assigning the vector<vector<bool> > value __x

Definition at line 753 of file search_graph.cc.

10.22.3.69 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< std::vector< int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<int> > value __x

Definition at line 760 of file search_graph.cc.

10.22.3.70 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< std::vector< unsigned int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<unsigned int> > value __x

Definition at line 767 of file search_graph.cc.

10.22.3.71 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< std::vector< double > > & __x) [inline]

Assignment Operator, assigning the vector<vector<double> > value __x

Definition at line 774 of file search_graph.cc.

10.22.3.72 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< std::vector< interval > > & __x) [inline]

Assignment Operator, assigning the vector<vector<interval> > value __x

Definition at line 781 of file search_graph.cc.

10.22.3.73 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< std::vector< std::string > > & __x) [inline]

Assignment Operator, assigning the vector<vector<string> > value __x

Definition at line 788 of file search_graph.cc.

10.22.3.74 basic_alltype& coco::coco::coco::basic_alltype::operator= (const std::vector< std::vector< num::Number > > & __x) [inline]

Assignment Operator, assigning the vector<vector<Number> > value __x

Definition at line 795 of file search_graph.cc.

10.22.3.75 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< bool > & __x) [inline]

Assignment Operator, assigning the sparse_vector<bool> value __x

Definition at line 802 of file search_graph.cc.

10.22.3.76 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<int> value __x

Definition at line 809 of file search_graph.cc.

10.22.3.77 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<unsigned int> value __x

Definition at line 816 of file search_graph.cc.

10.22.3.78 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< double > & __x) [inline]

Assignment Operator, assigning the sparse_vector<double> value __x

Definition at line 823 of file search_graph.cc.

10.22.3.79 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< interval > & __x) [inline]

Assignment Operator, assigning the sparse_vector<interval> value __x

Definition at line 830 of file search_graph.cc.

10.22.3.80 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_vector<string> value __x

Definition at line 837 of file search_graph.cc.

10.22.3.81 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_vector<Number> value __x

Definition at line 844 of file search_graph.cc.

10.22.3.82 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< double > & __x) [inline]

Assignment Operator, assigning the dense_matrix<double> value __x

Definition at line 851 of file search_graph.cc.

10.22.3.83 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< int > & __x) [inline]

Assignment Operator, assigning the dense_matrix<int> value __x

Definition at line 858 of file search_graph.cc.

10.22.3.84 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< interval > & __x) [inline]

Assignment Operator, assigning the dense_matrix<interval> value __x

Definition at line 865 of file search_graph.cc.

10.22.3.85 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the dense_matrix<string> value __x

Definition at line 872 of file search_graph.cc.

10.22.3.86 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the dense_matrix<Number> value __x

Definition at line 879 of file search_graph.cc.

10.22.3.87 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< double > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<double> value __x

Definition at line 886 of file search_graph.cc.

10.22.3.88 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< int > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<int> value ___x

Definition at line 893 of file search_graph.cc.

10.22.3.89 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< interval > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<interval> value ___x

Definition at line 900 of file search_graph.cc.

10.22.3.90 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<string> value ___x

Definition at line 907 of file search_graph.cc.

10.22.3.91 basic_alltype& coco::coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<Number> value ___x

Definition at line 914 of file search_graph.cc.

10.22.3.92 basic_alltype& coco::coco::coco::basic_alltype::operator= (const basic_alltype & __a) [inline]

Standard Assignment Operator

Definition at line 921 of file search_graph.cc.

10.22.3.93 bool coco::coco::coco::basic_alltype::operator== (const basic_alltype & b) const

Standard comparison operator

10.22.3.94 const void* coco::coco::coco::basic_alltype::p ( ) const [inline]

retrieve a void* from the basic_alltype

Definition at line 1076 of file search_graph.cc.

10.22.3.95 void*& coco::coco::coco::basic_alltype::p ( ) [inline]

retrieve a void* reference from the basic_alltype

Definition at line 1348 of file search_graph.cc.

10.22.3.96 const std::string& coco::coco::coco::basic_alltype::s ( ) const [inline]

retrieve a string from the basic_alltype

Definition at line 1079 of file search_graph.cc.

10.22.3.97 std::string& coco::coco::coco::basic_alltype::s ( ) [inline]

retrieve a string from the basic_alltype

Definition at line 1351 of file search_graph.cc.

10.22.3.98 const vmtl::sparse_vector<bool>& coco::coco::coco::basic_alltype::sb ( ) const [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1224 of file search_graph.cc.

10.22.3.99 vmtl::sparse_vector<bool>& coco::coco::coco::basic_alltype::sb ( ) [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1495 of file search_graph.cc.

10.22.3.100 const vmtl::sparse_vector<std::string>& coco::coco::coco::basic_alltype::sc ( ) const [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1280 of file search_graph.cc.

10.22.3.101 vmtl::sparse_vector<std::string>& coco::coco::coco::basic_alltype::sc ( ) [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1551 of file search_graph.cc.

10.22.3.102 const vmtl::sparse_vector<double>& coco::coco::coco::basic_alltype::sd ( ) const [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1272 of file search_graph.cc.

10.22.3.103 vmtl::sparse_vector<double>& coco::coco::coco::basic_alltype::sd ( ) [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1543 of file search_graph.cc.

10.22.3.104 basic_alltype& coco::coco::coco::basic_alltype::set_cdm (vmtl::dense_matrix< std::string > * __m) [inline]

This method assigns a dense_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<string>.

Definition at line 973 of file search_graph.cc.

10.22.3.105 basic_alltype& coco::coco::coco::basic_alltype::set_csm (vmtl::sparse_matrix< std::string > * __m) [inline]

This method assigns a sparse_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<string>.

Definition at line 1028 of file search_graph.cc.

10.22.3.106 basic_alltype& coco::coco::coco::basic_alltype::set_dm (vmtl::dense_matrix< double > * __m) [inline]

This method assigns a dense_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<double>.

Definition at line 940 of file search_graph.cc.

10.22.3.107 basic_alltype& coco::coco::coco::basic_alltype::set_idm (vmtl::dense_matrix< interval > * __m) [inline]

This method assigns a dense_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<interval>.

Definition at line 962 of file search_graph.cc.

10.22.3.108 basic_alltype& coco::coco::coco::basic_alltype::set_ism (vmtl::sparse_matrix< interval > * __m) [inline]

This method assigns a sparse_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<interval>.

Definition at line 1017 of file search_graph.cc.

10.22.3.109 basic_alltype& coco::coco::coco::basic_alltype::set_ndm (vmtl::dense_matrix< int > * __m) [inline]

This method assigns a dense_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<int>.

Definition at line 951 of file search_graph.cc.

10.22.3.110 basic_alltype& coco::coco::coco::basic_alltype::set_nsm (vmtl::sparse_matrix< int > * __m) [inline]

This method assigns a sparse_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<int>.

Definition at line 1006 of file search_graph.cc.

10.22.3.111 basic_alltype& coco::coco::coco::basic_alltype::set_sm (vmtl::sparse_matrix< double > * __m) [inline]

This method assigns a sparse_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<double>.

Definition at line 995 of file search_graph.cc.

10.22.3.112 basic_alltype& coco::coco::coco::basic_alltype::set_xdm (vmtl::dense_matrix< num::Number > * __m) [inline]

This method assigns a dense_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<Number>.

Definition at line 984 of file search_graph.cc.

10.22.3.113 basic_alltype& coco::coco::coco::basic_alltype::set_xsm (vmtl::sparse_matrix< num::Number > * __m) [inline]

This method assigns a sparse_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<Number>.

Definition at line 1039 of file search_graph.cc.

10.22.3.114 const vmtl::sparse_vector<interval>& coco::coco::coco::basic_alltype::si ( ) const [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1276 of file search_graph.cc.

10.22.3.115 vmtl::sparse_vector<interval>& coco::coco::coco::basic_alltype::si ( ) [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1547 of file search_graph.cc.

10.22.3.116 const vmtl::sparse_matrix<double>& coco::coco::coco::basic_alltype::sm ( ) const [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1305 of file search_graph.cc.

10.22.3.117 vmtl::sparse_matrix<double>& coco::coco::coco::basic_alltype::sm ( ) [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1576 of file search_graph.cc.

10.22.3.118 const vmtl::sparse_vector<int>& coco::coco::coco::basic_alltype::sn ( ) const [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1228 of file search_graph.cc.

10.22.3.119 vmtl::sparse_vector<int>& coco::coco::coco::basic_alltype::sn ( ) [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1499 of file search_graph.cc.

10.22.3.120 const vmtl::sparse_vector<unsigned int>& coco::coco::coco::basic_alltype::su ( ) const [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1250 of file search_graph.cc.

10.22.3.121 vmtl::sparse_vector<unsigned int>& coco::coco::coco::basic_alltype::su ( ) [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1521 of file search_graph.cc.

10.22.3.122 const vmtl::sparse_vector<num::Number>& coco::coco::coco::basic_alltype::sx ( ) const [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1284 of file search_graph.cc.

10.22.3.123 vmtl::sparse_vector<num::Number>& coco::coco::coco::basic_alltype::sx ( ) [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1555 of file search_graph.cc.

10.22.3.124 const char* coco::coco::coco::basic_alltype::type_cstr ( ) const [inline]

Return the type of the value contained in the storage as a C-string

Definition at line 1674 of file search_graph.cc.

10.22.3.125 std::string coco::coco::coco::basic_alltype::type_name ( ) const [inline]

Return the type of the value contained in the storage as a C++-string

Definition at line 1671 of file search_graph.cc.

10.22.3.126 const std::vector<unsigned int>& coco::coco::coco::basic_alltype::u ( ) const [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1111 of file search_graph.cc.

10.22.3.127 std::vector<unsigned int>& coco::coco::coco::basic_alltype::u ( ) [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1383 of file search_graph.cc.

10.22.3.128 const std::vector<std::vector<bool> >& coco::coco::coco::basic_alltype::vb ( ) const [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1150 of file search_graph.cc.

10.22.3.129 std::vector<std::vector<bool> >& coco::coco::coco::basic_alltype::vb ( ) [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1422 of file search_graph.cc.

10.22.3.130 const std::vector<std::vector<std::string> >& coco::coco::coco::basic_alltype::vc ( ) const [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1214 of file search_graph.cc.

10.22.3.131 std::vector<std::vector<std::string> >& coco::coco::coco::basic_alltype::vc ( ) [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1486 of file search_graph.cc.

10.22.3.132 const std::vector<std::vector<double> >& coco::coco::coco::basic_alltype::vd ( ) const [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1206 of file search_graph.cc.

10.22.3.133 std::vector<std::vector<double> >& coco::coco::coco::basic_alltype::vd ( ) [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1478 of file search_graph.cc.

10.22.3.134 const std::vector<std::vector<interval> >& coco::coco::coco::basic_alltype::vi ( ) const [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1210 of file search_graph.cc.

10.22.3.135 std::vector<std::vector<interval> >& coco::coco::coco::basic_alltype::vi ( ) [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1482 of file search_graph.cc.

10.22.3.136 const std::vector<std::vector<int> >& coco::coco::coco::basic_alltype::vn ( ) const [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1154 of file search_graph.cc.

10.22.3.137 std::vector<std::vector<int> >& coco::coco::coco::basic_alltype::vn ( ) [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1426 of file search_graph.cc.

10.22.3.138 const std::vector<std::vector<unsigned int> >& coco::coco::coco::basic_alltype::vu ( ) const [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1180 of file search_graph.cc.

10.22.3.139 std::vector<std::vector<unsigned int> >& coco::coco::coco::basic_alltype::vu ( ) [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1452 of file search_graph.cc.

10.22.3.140 const std::vector<std::vector<num::Number> >& coco::coco::coco::basic_alltype::vx ( ) const [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1218 of file search_graph.cc.

10.22.3.141 std::vector<std::vector<num::Number> >& coco::coco::coco::basic_alltype::vx ( ) [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1490 of file search_graph.cc.

10.22.3.142 const std::vector<num::Number>& coco::coco::coco::basic_alltype::x ( ) const [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1145 of file search_graph.cc.

10.22.3.143 std::vector<num::Number>& coco::coco::coco::basic_alltype::x ( ) [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1417 of file search_graph.cc.

10.22.3.144 const vmtl::dense_matrix<num::Number>& coco::coco::coco::basic_alltype::xdm ( ) const [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1301 of file search_graph.cc.

10.22.3.145 vmtl::dense_matrix<num::Number>& coco::coco::coco::basic_alltype::xdm ( ) [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1572 of file search_graph.cc.

10.22.3.146 const vmtl::sparse_matrix<num::Number>& coco::coco::coco::basic_alltype::xsm ( ) const [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1317 of file search_graph.cc.

10.22.3.147 vmtl::sparse_matrix<num::Number>& coco::coco::coco::basic_alltype::xsm ( ) [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1588 of file search_graph.cc.

10.22.3.148 const num::Number& coco::coco::coco::basic_alltype::y ( ) const [inline]

retrieve a Number from the basic_alltype

Definition at line 1082 of file search_graph.cc.

10.22.3.149 num::Number& coco::coco::coco::basic_alltype::y ( ) [inline]

retrieve a Number from the basic_alltype

Definition at line 1354 of file search_graph.cc.

10.22.4 Member Data Documentation

10.22.4.1 std::vector<bool>* coco::coco::coco::basic_alltype::b

Definition at line 191 of file search_graph.cc.

10.22.4.2 std::vector<std::string>* coco::coco::coco::basic_alltype::c

Definition at line 196 of file search_graph.cc.

10.22.4.3 vmtl::dense_matrix<std::string>* coco::coco::coco::basic_alltype::cdm

Definition at line 215 of file search_graph.cc.

10.22.4.4 vmtl::sparse_matrix<std::string>* coco::coco::coco::basic_alltype::csm

Definition at line 220 of file search_graph.cc.

10.22.4.5 std::vector<double>* coco::coco::coco::basic_alltype::d

Definition at line 194 of file search_graph.cc.

10.22.4.6 vmtl::dense_matrix<double>* coco::coco::coco::basic_alltype::dm

Definition at line 212 of file search_graph.cc.

10.22.4.7 std::vector<interval>* coco::coco::coco::basic_alltype::i

Definition at line 195 of file search_graph.cc.

10.22.4.8 vmtl::dense_matrix<interval>* coco::coco::coco::basic_alltype::idm

Definition at line 214 of file search_graph.cc.

10.22.4.9 vmtl::sparse_matrix<interval>* coco::coco::coco::basic_alltype::ism

Definition at line 219 of file search_graph.cc.

10.22.4.10 std::vector<int>* coco::coco::coco::basic_alltype::n

Definition at line 192 of file search_graph.cc.

10.22.4.11 bool coco::coco::coco::basic_alltype::nb

Definition at line 183 of file search_graph.cc.

10.22.4.12 double coco::coco::coco::basic_alltype::nd

Definition at line 186 of file search_graph.cc.

10.22.4.13 vmtl::dense_matrix<int>* coco::coco::coco::basic_alltype::ndm

Definition at line 213 of file search_graph.cc.

10.22.4.14 interval_st coco::coco::coco::basic_alltype::ni

Definition at line 187 of file search_graph.cc.

10.22.4.15 int coco::coco::coco::basic_alltype::nn

Definition at line 184 of file search_graph.cc.

10.22.4.16 vmtl::sparse_matrix<int>* coco::coco::coco::basic_alltype::nsm

Definition at line 218 of file search_graph.cc.

10.22.4.17 unsigned int coco::coco::coco::basic_alltype::nu

Definition at line 185 of file search_graph.cc.

10.22.4.18 void* coco::coco::coco::basic_alltype::p

Definition at line 188 of file search_graph.cc.

10.22.4.19 std::string* coco::coco::coco::basic_alltype::s

Definition at line 189 of file search_graph.cc.

10.22.4.20 vmtl::sparse_vector<bool>* coco::coco::coco::basic_alltype::sb

Definition at line 205 of file search_graph.cc.

10.22.4.21 vmtl::sparse_vector<std::string>* coco::coco::coco::basic_alltype::sc

Definition at line 210 of file search_graph.cc.

10.22.4.22 vmtl::sparse_vector<double>* coco::coco::coco::basic_alltype::sd

Definition at line 208 of file search_graph.cc.

10.22.4.23 vmtl::sparse_vector<interval>* coco::coco::coco::basic_alltype::si

Definition at line 209 of file search_graph.cc.

10.22.4.24 vmtl::sparse_matrix<double>* coco::coco::coco::basic_alltype::sm

Definition at line 217 of file search_graph.cc.

10.22.4.25 vmtl::sparse_vector<int>* coco::coco::coco::basic_alltype::sn

Definition at line 206 of file search_graph.cc.

10.22.4.26 vmtl::sparse_vector<unsigned int>* coco::coco::coco::basic_alltype::su

Definition at line 207 of file search_graph.cc.

10.22.4.27 vmtl::sparse_vector<num::Number>* coco::coco::coco::basic_alltype::sx

Definition at line 211 of file search_graph.cc.

10.22.4.28 std::vector<unsigned int>* coco::coco::coco::basic_alltype::u

Definition at line 193 of file search_graph.cc.

10.22.4.29 std::vector<std::vector<bool> >* coco::coco::coco::basic_alltype::vb

Definition at line 198 of file search_graph.cc.

10.22.4.30 std::vector<std::vector<std::string> >* coco::coco::coco::basic_alltype::vc

Definition at line 203 of file search_graph.cc.

10.22.4.31 std::vector<std::vector<double> >* coco::coco::coco::basic_alltype::vd

Definition at line 201 of file search_graph.cc.

10.22.4.32 std::vector<std::vector<interval> >* coco::coco::coco::basic_alltype::vi

Definition at line 202 of file search_graph.cc.

10.22.4.33 std::vector<std::vector<int> >* coco::coco::coco::basic_alltype::vn

Definition at line 199 of file search_graph.cc.

10.22.4.34 std::vector<std::vector<unsigned int> >* coco::coco::coco::basic_alltype::vu

Definition at line 200 of file search_graph.cc.

10.22.4.35 std::vector<std::vector<num::Number> >* coco::coco::coco::basic_alltype::vx

Definition at line 204 of file search_graph.cc.

10.22.4.36 std::vector<num::Number>* coco::coco::coco::basic_alltype::x

Definition at line 197 of file search_graph.cc.

10.22.4.37 vmtl::dense_matrix<num::Number>* coco::coco::coco::basic_alltype::xdm

Definition at line 216 of file search_graph.cc.

10.22.4.38 vmtl::sparse_matrix<num::Number>* coco::coco::coco::basic_alltype::xsm

Definition at line 221 of file search_graph.cc.

10.22.4.39 num::Number* coco::coco::coco::basic_alltype::y

Definition at line 190 of file search_graph.cc.

The documentation for this class was generated from the following files:

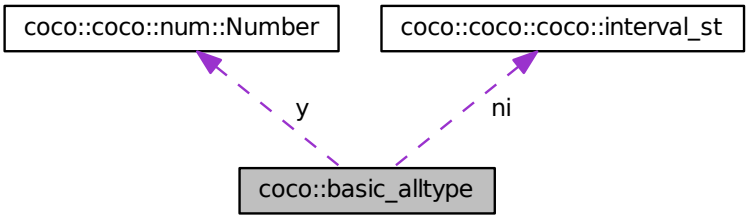
- basic_alltype.h
- basic_alltype.cc

10.23 coco::basic_alltype Class Reference

The basic alltype which can hold any of a number of basic types.

`#include <basic_alltype.h>`

Collaboration diagram for coco::basic_alltype:



Public Member Functions

- basic_alltype ()
- basic_alltype (bool __x)
- basic_alltype (int __x)
- basic_alltype (unsigned int __x)
- basic_alltype (double __x)
- basic_alltype (interval __x)
- basic_alltype (void *__x)
- basic_alltype (const char *__cp)
- basic_alltype (const std::string &__x)
- basic_alltype (const num::Number &__y)
- basic_alltype (const std::vector< bool > &__x)
- basic_alltype (const std::vector< int > &__x)
- basic_alltype (const std::vector< unsigned int > &__x)
- basic_alltype (const std::vector< double > &__x)
- basic_alltype (const std::vector< interval > &__x)
- basic_alltype (const std::vector< std::string > &__x)
- basic_alltype (const std::vector< num::Number > &__x)
- basic_alltype (const std::vector< std::vector< bool > > &__vx)
- basic_alltype (const std::vector< std::vector< int > > &__vx)
- basic_alltype (const std::vector< std::vector< unsigned int > > &__vx)
- basic_alltype (const std::vector< std::vector< double > > &__vx)
- basic_alltype (const std::vector< std::vector< interval > > &__vx)
- basic_alltype (const std::vector< std::vector< std::string > > &__vx)
- basic_alltype (const std::vector< std::vector< num::Number > > &__vx)
- basic_alltype (const vmtl::sparse_vector< bool > &__x)
- basic_alltype (const vmtl::sparse_vector< int > &__x)
- basic_alltype (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype (const vmtl::sparse_vector< double > &__x)
- basic_alltype (const vmtl::sparse_vector< interval > &__x)
- basic_alltype (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype (const vmtl::sparse_vector< num::Number > &__x)

- basic_alltype (const vmtl::dense_matrix< double > &__x)
- basic_alltype (const vmtl::dense_matrix< int > &__x)
- basic_alltype (const vmtl::dense_matrix< interval > &__x)
- basic_alltype (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype (const vmtl::sparse_matrix< double > &__x)
- basic_alltype (const vmtl::sparse_matrix< int > &__x)
- basic_alltype (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype (const vmtl::sparse_matrix< num::Number > &__x)
- ~basic_alltype ()
- basic_alltype (const basic_alltype &__a)
- basic_alltype & operator= (bool __x)
- basic_alltype & operator= (int __x)
- basic_alltype & operator= (unsigned int __x)
- basic_alltype & operator= (double __x)
- basic_alltype & operator= (interval __x)
- basic_alltype & operator= (void *__x)
- basic_alltype & operator= (const std::string &__x)
- basic_alltype & operator= (const char *__x)
- basic_alltype & operator= (const num::Number &__x)
- basic_alltype & operator= (const std::vector< bool > &__x)
- basic_alltype & operator= (const std::vector< int > &__x)
- basic_alltype & operator= (const std::vector< unsigned int > &__x)
- basic_alltype & operator= (const std::vector< double > &__x)
- basic_alltype & operator= (const std::vector< interval > &__x)
- basic_alltype & operator= (const std::vector< std::string > &__x)
- basic_alltype & operator= (const std::vector< num::Number > &__x)
- basic_alltype & operator= (const std::vector< std::vector< bool > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< unsigned int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< double > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< interval > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< std::string > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< num::Number > > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< bool > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< interval > &__x)

- basic_alltype & operator= (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< num::Number > &__x)
- basic_alltype & operator= (const basic_alltype &__a)
- basic_alltype & clear ()
- basic_alltype & set_dm (vmtl::dense_matrix< double > *__m)
- basic_alltype & set_ndm (vmtl::dense_matrix< int > *__m)
- basic_alltype & set_idm (vmtl::dense_matrix< interval > *__m)
- basic_alltype & set_cdm (vmtl::dense_matrix< std::string > *__m)
- basic_alltype & set_xdm (vmtl::dense_matrix< num::Number > *__m)
- basic_alltype & set_sm (vmtl::sparse_matrix< double > *__m)
- basic_alltype & set_nsm (vmtl::sparse_matrix< int > *__m)
- basic_alltype & set_ism (vmtl::sparse_matrix< interval > *__m)
- basic_alltype & set_csm (vmtl::sparse_matrix< std::string > *__m)
- basic_alltype & set_xsm (vmtl::sparse_matrix< num::Number > *__m)
- bool nb () const BASIC_ALLTYPE_THROW
- int nn () const BASIC_ALLTYPE_THROW
- unsigned int nu () const BASIC_ALLTYPE_THROW
- double nd () const BASIC_ALLTYPE_THROW
- interval ni () const BASIC_ALLTYPE_THROW
- const void * p () const BASIC_ALLTYPE_THROW
- const std::string & s () const BASIC_ALLTYPE_THROW
- const num::Number & y () const BASIC_ALLTYPE_THROW
- const std::vector< bool > & b () const BASIC_ALLTYPE_THROW
- const std::vector< int > & n () const BASIC_ALLTYPE_THROW
- const std::vector< unsigned int > & u () const BASIC_ALLTYPE_THROW
- const std::vector< double > & d () const BASIC_ALLTYPE_THROW
- const std::vector< interval > & i () const BASIC_ALLTYPE_THROW
- const std::vector< std::string > & c () const BASIC_ALLTYPE_THROW
- const std::vector< num::Number > & x () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < bool > > & vb () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < int > > & vn () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < unsigned int > > & vu () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < double > > & vd () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < interval > > & vi () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < std::string > > & vc () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < num::Number > > & vx () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< bool > & sb () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< int > & sn () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < unsigned int > & su () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < double > & sd () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < interval > & si () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < std::string > & sc () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < num::Number > & sx () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < double > & dm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix< int > & ndm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < interval > & idm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < std::string > & cdm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < num::Number > & xdm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < double > & sm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix< int > & nsm () const BASIC_ALLTYPE_THROW

- const vmtl::sparse_matrix < interval > & ism () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < std::string > & csm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < num::Number > & xsm () const BASIC_ALLTYPE_THROW
- bool & nb () BASIC_ALLTYPE_THROW
- int & nn () BASIC_ALLTYPE_THROW
- unsigned int & nu () BASIC_ALLTYPE_THROW
- double & nd () BASIC_ALLTYPE_THROW
- interval_st & ni () BASIC_ALLTYPE_THROW
- void *& p () BASIC_ALLTYPE_THROW
- std::string & s () BASIC_ALLTYPE_THROW
- num::Number & y () BASIC_ALLTYPE_THROW
- std::vector< bool > & b () BASIC_ALLTYPE_THROW
- std::vector< int > & n () BASIC_ALLTYPE_THROW
- std::vector< unsigned int > & u () BASIC_ALLTYPE_THROW
- std::vector< double > & d () BASIC_ALLTYPE_THROW
- std::vector< interval > & i () BASIC_ALLTYPE_THROW
- std::vector< std::string > & c () BASIC_ALLTYPE_THROW
- std::vector< num::Number > & x () BASIC_ALLTYPE_THROW
- std::vector< std::vector< bool > > & vb () BASIC_ALLTYPE_THROW
- std::vector< std::vector< int > > & vn () BASIC_ALLTYPE_THROW
- std::vector< std::vector < unsigned int > > & vu () BASIC_ALLTYPE_THROW
- std::vector< std::vector < double > > & vd () BASIC_ALLTYPE_THROW
- std::vector< std::vector < interval > > & vi () BASIC_ALLTYPE_THROW
- std::vector< std::vector < std::string > > & vc () BASIC_ALLTYPE_THROW
- std::vector< std::vector < num::Number > > & vx () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< bool > & sb () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< int > & sn () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< unsigned int > & su () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< double > & sd () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< interval > & si () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < std::string > & sc () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < num::Number > & sx () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< double > & dm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< int > & ndm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< interval > & idm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< std::string > & cdm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< num::Number > & xdm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< double > & sm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< int > & nsm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< interval > & ism () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < std::string > & csm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < num::Number > & xsm () BASIC_ALLTYPE_THROW
- bool is_allocated () const
- bool is_scalar () const
- bool is_vector () const
- bool is_vector_of_vector () const
- bool is_dense_vector () const
- bool is_sparse_vector () const
- bool is_empty_vector () const
- bool is_matrix () const

- bool is_dense_matrix () const
- bool is_sparse_matrix () const
- bool is_empty_matrix () const
- bool empty () const
- int contents_type () const
- std::string type_name () const
- const char * type_cstr () const
- bool operator== (const basic_alltype &b) const
- bool operator!= (const basic_alltype &b) const
- bool less_than (const basic_alltype &b) const
- bool less_equal (const basic_alltype &b) const

10.23.1 Detailed Description

This class is the base of the data handling in the API. It can hold values of different types: bool, int, unsigned int, double, interval, string, Number, vector<bool>, vector<int>, vector<unsigned int>, vector<double>, vector<interval>, vector<string>, vector<Number>, sparse_vector<bool>, sparse_vector<int>, sparse_vector<unsigned int>, sparse_vector<double>, sparse_vector<interval>, sparse_vector<string>, sparse_vector<Number>, dense_matrix<double>, dense_matrix<int>, dense_matrix<interval>, dense_matrix<string>, dense_matrix<Number>, sparse_matrix<double>, sparse_matrix<int>, sparse_matrix<interval>, sparse_matrix<string>, sparse_matrix<Number>. All of these types can be stored and retrieved conveniently.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 coco::basic_alltype::basic_alltype ( ) [inline]

Standard Constructor yielding an empty basic_alltype

Definition at line 480 of file basic_alltype.h.

10.23.2.2 coco::basic_alltype::basic_alltype (bool __x) [inline]

Constructor storing the bool ___x

Definition at line 482 of file basic_alltype.h.

10.23.2.3 coco::basic_alltype::basic_alltype (int __x) [inline]

Constructor storing the int ___x

Definition at line 485 of file basic_alltype.h.

10.23.2.4 coco::basic_alltype::basic_alltype (unsigned int __x) [inline]

Constructor storing the unsigned int ___x

Definition at line 488 of file basic_alltype.h.

10.23.2.5 coco::basic_alltype::basic_alltype (double __x) [inline]

Constructor storing the double ___x

Definition at line 491 of file basic_alltype.h.

10.23.2.6 coco::basic_alltype::basic_alltype (interval __x) [inline]

Constructor storing the interval ___x

Definition at line 494 of file basic_alltype.h.

10.23.2.7 coco::basic_alltype::basic_alltype (void * __x) [inline]

Constructor storing the void* ___x

Definition at line 497 of file basic_alltype.h.

10.23.2.8 coco::basic_alltype::basic_alltype (const char * __cp) [inline]

Constructor storing the string ___cp

Definition at line 500 of file basic_alltype.h.

10.23.2.9 coco::basic_alltype::basic_alltype (const std::string & __x) [inline]

Constructor storing the string ___x

Definition at line 503 of file basic_alltype.h.

10.23.2.10 coco::basic_alltype::basic_alltype (const num::Number & __y) [inline]

Constructor storing the number ___y

Definition at line 506 of file basic_alltype.h.

10.23.2.11 coco::basic_alltype::basic_alltype (const std::vector< bool > & __x) [inline]

Constructor storing the vector<bool> ___x

Definition at line 510 of file basic_alltype.h.

10.23.2.12 coco::basic_alltype::basic_alltype (const std::vector< int > & __x) [inline]

Constructor storing the vector<int> ___x

Definition at line 514 of file basic_alltype.h.

10.23.2.13 coco::basic_alltype::basic_alltype (const std::vector< unsigned int > & __x) [inline]

Constructor storing the vector<unsigned int> ___x

Definition at line 518 of file basic_alltype.h.

10.23.2.14 coco::basic_alltype::basic_alltype (const std::vector< double > & __x) [inline]

Constructor storing the vector<double> ___x

Definition at line 522 of file basic_alltype.h.

10.23.2.15 coco::basic_alltype::basic_alltype (const std::vector< interval > & __x) [inline]

Constructor storing the vector<interval> __x

Definition at line 526 of file basic_alltype.h.

10.23.2.16 coco::basic_alltype::basic_alltype (const std::vector< std::string > & __x) [inline]

Constructor storing the vector<string> __x

Definition at line 530 of file basic_alltype.h.

10.23.2.17 coco::basic_alltype::basic_alltype (const std::vector< num::Number > & __x) [inline]

Constructor storing the vector<Number> __x

Definition at line 534 of file basic_alltype.h.

10.23.2.18 coco::basic_alltype::basic_alltype (const std::vector< std::vector< bool > > & __vx) [inline]

Constructor storing the vector<vector<bool> > __vx

Definition at line 539 of file basic_alltype.h.

10.23.2.19 coco::basic_alltype::basic_alltype (const std::vector< std::vector< int > > & __vx) [inline]

Constructor storing the vector<vector<int> > __vx

Definition at line 543 of file basic_alltype.h.

10.23.2.20 coco::basic_alltype::basic_alltype (const std::vector< std::vector< unsigned int > > & __vx) [inline]

Constructor storing the vector<vector<unsigned int> > __vx

Definition at line 547 of file basic_alltype.h.

10.23.2.21 coco::basic_alltype::basic_alltype (const std::vector< std::vector< double > > & __vx) [inline]

Constructor storing the vector<vector<double> > __vx

Definition at line 551 of file basic_alltype.h.

10.23.2.22 coco::basic_alltype::basic_alltype (const std::vector< std::vector< interval > > & __vx) [inline]

Constructor storing the vector<vector<interval> > __vx

Definition at line 555 of file basic_alltype.h.

10.23.2.23 coco::basic_alltype::basic_alltype (const std::vector< std::vector< std::string > > & __vx) [inline]

Constructor storing the vector<vector<string> > __vx

Definition at line 559 of file basic_alltype.h.

10.23.2.24 coco::basic_alltype::basic_alltype (const std::vector< std::vector< num::Number > > & __vx) [inline]

Constructor storing the vector<vector<Number> > __vx

Definition at line 563 of file basic_alltype.h.

10.23.2.25 coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< bool > & __x) [inline]

Constructor storing the sparse_vector<bool> __x

Definition at line 568 of file basic_alltype.h.

10.23.2.26 coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< int > & __x) [inline]

Constructor storing the sparse_vector<int> __x

Definition at line 572 of file basic_alltype.h.

10.23.2.27 coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Constructor storing the sparse_vector<unsigned int> __x

Definition at line 576 of file basic_alltype.h.

10.23.2.28 coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< double > & __x) [inline]

Constructor storing the sparse_vector<double> __x

Definition at line 580 of file basic_alltype.h.

10.23.2.29 coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< interval > & __x) [inline]

Constructor storing the sparse_vector<interval> __x

Definition at line 584 of file basic_alltype.h.

10.23.2.30 coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< std::string > & __x) [inline]

Constructor storing the sparse_vector<string> __x

Definition at line 588 of file basic_alltype.h.

10.23.2.31 coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< num::Number > & __x) [inline]

Constructor storing the sparse_vector<Number> __x

Definition at line 592 of file basic_alltype.h.

10.23.2.32 coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< double > & __x) [inline]

Constructor storing the dense_matrix<double> __x

Definition at line 597 of file basic_alltype.h.

10.23.2.33 coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< int > & __x) [inline]

Constructor storing the dense_matrix<int> __x

Definition at line 601 of file basic_alltype.h.

10.23.2.34 coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< interval > & __x) [inline]

Constructor storing the dense_matrix<interval> __x

Definition at line 605 of file basic_alltype.h.

10.23.2.35 coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< std::string > & __x) [inline]

Constructor storing the dense_matrix<string> __x

Definition at line 609 of file basic_alltype.h.

10.23.2.36 coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< num::Number > & __x) [inline]

Constructor storing the dense_matrix<Number> __x

Definition at line 613 of file basic_alltype.h.

10.23.2.37 coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< double > & __x) [inline]

Constructor storing the sparse_matrix<double> __x

Definition at line 618 of file basic_alltype.h.

10.23.2.38 coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< int > & __x) [inline]

Constructor storing the sparse_matrix<int> __x

Definition at line 622 of file basic_alltype.h.

10.23.2.39 coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< interval > & __x) [inline]

Constructor storing the sparse_matrix<interval> __x

Definition at line 626 of file basic_alltype.h.

10.23.2.40 coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< std::string > & __x) [inline]

Constructor storing the sparse_matrix<string> ___x

Definition at line 630 of file basic_alltype.h.

10.23.2.41 coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Constructor storing the sparse_matrix<Number> ___x

Definition at line 634 of file basic_alltype.h.

10.23.2.42 coco::basic_alltype::~basic_alltype ( ) [inline]

Standard Destructor

Definition at line 639 of file basic_alltype.h.

10.23.2.43 coco::basic_alltype::basic_alltype (const basic_alltype & __a) [inline]

Standard Copy Constructor

Definition at line 643 of file basic_alltype.h.

10.23.3 Member Function Documentation

10.23.3.1 const std::vector<bool>& coco::basic_alltype::b ( ) const [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1085 of file basic_alltype.h.

10.23.3.2 std::vector<bool>& coco::basic_alltype::b ( ) [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1357 of file basic_alltype.h.

10.23.3.3 const std::vector<std::string>& coco::basic_alltype::c ( ) const [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1141 of file basic_alltype.h.

10.23.3.4 std::vector<std::string>& coco::basic_alltype::c ( ) [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1413 of file basic_alltype.h.

10.23.3.5 const vmtl::dense_matrix<std::string>& coco::basic_alltype::cdm ( ) const [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1298 of file basic_alltype.h.

10.23.3.6 vmtl::dense_matrix<std::string>& coco::basic_alltype::cdm ( ) [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1569 of file basic_alltype.h.

10.23.3.7 basic_alltype& coco::basic_alltype::clear ( ) [inline]

Reset the basic_alltype to an empty value

Definition at line 929 of file basic_alltype.h.

10.23.3.8 int coco::basic_alltype::contents_type ( ) const [inline]

Return the type of the value contained in the storage, an int which can be compared with the ALLTYPE_... values.

Definition at line 1669 of file basic_alltype.h.

10.23.3.9 const vmtl::sparse_matrix<std::string>& coco::basic_alltype::csm ( ) const [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1314 of file basic_alltype.h.

10.23.3.10 vmtl::sparse_matrix<std::string>& coco::basic_alltype::csm ( ) [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1585 of file basic_alltype.h.

10.23.3.11 const std::vector<double>& coco::basic_alltype::d ( ) const [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1133 of file basic_alltype.h.

10.23.3.12 std::vector<double>& coco::basic_alltype::d ( ) [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1405 of file basic_alltype.h.

10.23.3.13 const vmtl::dense_matrix<double>& coco::basic_alltype::dm ( ) const [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1289 of file basic_alltype.h.

10.23.3.14 vmtl::dense_matrix<double>& coco::basic_alltype::dm ( ) [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1560 of file basic_alltype.h.

10.23.3.15 bool coco::basic_alltype::empty ( ) const [inline]

Check, whether the basic_alltype is empty.

Definition at line 1666 of file basic_alltype.h.

10.23.3.16 const std::vector<interval>& coco::basic_alltype::i ( ) const [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1137 of file basic_alltype.h.

10.23.3.17 std::vector<interval>& coco::basic_alltype::i ( ) [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1409 of file basic_alltype.h.

10.23.3.18 const vmtl::dense_matrix<interval>& coco::basic_alltype::idm ( ) const [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1295 of file basic_alltype.h.

10.23.3.19 vmtl::dense_matrix<interval>& coco::basic_alltype::idm ( ) [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1566 of file basic_alltype.h.

10.23.3.20 bool coco::basic_alltype::is_allocated ( ) const [inline]

Check, whether the basic_alltype contains a complex type, i.e. one which has to be allocated (not bool, int, unsigned int, double, or interval).

Definition at line 1595 of file basic_alltype.h.

10.23.3.21 bool coco::basic_alltype::is_dense_matrix ( ) const [inline]

Check, whether the basic_alltype contains a dense matrix type.

Definition at line 1644 of file basic_alltype.h.

10.23.3.22 bool coco::basic_alltype::is_dense_vector ( ) const [inline]

Check, whether the basic_alltype contains a dense vector type.

Definition at line 1607 of file basic_alltype.h.

10.23.3.23 bool coco::basic_alltype::is_empty_matrix ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1648 of file basic_alltype.h.

10.23.3.24 bool coco::basic_alltype::is_empty_vector ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1613 of file basic_alltype.h.

10.23.3.25 bool coco::basic_alltype::is_matrix ( ) const [inline]

Check, whether the basic_alltype contains a matrix type.

Definition at line 1642 of file basic_alltype.h.

10.23.3.26 bool coco::basic_alltype::is_scalar ( ) const [inline]

Check, whether the basic_alltype contains a scalar type, i.e. neither a vector nor a matrix.

Definition at line 1598 of file basic_alltype.h.

10.23.3.27 bool coco::basic_alltype::is_sparse_matrix ( ) const [inline]

Check, whether the basic_alltype contains a sparse matrix type.

Definition at line 1646 of file basic_alltype.h.

10.23.3.28 bool coco::basic_alltype::is_sparse_vector ( ) const [inline]

Check, whether the basic_alltype contains a sparse vector type.

Definition at line 1610 of file basic_alltype.h.

10.23.3.29 bool coco::basic_alltype::is_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector type.

Definition at line 1601 of file basic_alltype.h.

10.23.3.30 bool coco::basic_alltype::is_vector_of_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector of vector type.

Definition at line 1604 of file basic_alltype.h.

10.23.3.31 const vmtl::sparse_matrix<interval>& coco::basic_alltype::ism ( ) const [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1311 of file basic_alltype.h.

10.23.3.32 vmtl::sparse_matrix<interval>& coco::basic_alltype::ism ( ) [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1582 of file basic_alltype.h.

10.23.3.33 bool coco::basic_alltype::less_equal (const basic_alltype & *b*) const

Total order comparison operator

10.23.3.34 bool coco::basic_alltype::less_than (const basic_alltype & *b*) const

Total order comparison operator

10.23.3.35 const std::vector<int>& coco::basic_alltype::n ( ) const [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1089 of file basic_alltype.h.

10.23.3.36 std::vector<int>& coco::basic_alltype::n ( ) [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1361 of file basic_alltype.h.

10.23.3.37 bool coco::basic_alltype::nb ( ) const [inline]

retrieve a bool from the basic_alltype

Definition at line 1049 of file basic_alltype.h.

10.23.3.38 bool& coco::basic_alltype::nb ( ) [inline]

retrieve a bool from the basic_alltype

Definition at line 1321 of file basic_alltype.h.

10.23.3.39 double coco::basic_alltype::nd ( ) const [inline]

retrieve a double from the basic_alltype

Definition at line 1070 of file basic_alltype.h.

10.23.3.40 double& coco::basic_alltype::nd ( ) [inline]

retrieve a double from the basic_alltype

Definition at line 1342 of file basic_alltype.h.

10.23.3.41 const vmtl::dense_matrix<int>& coco::basic_alltype::ndm ( ) const [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1292 of file basic_alltype.h.

10.23.3.42 vmtl::dense_matrix<int>& coco::basic_alltype::ndm ( ) [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1563 of file basic_alltype.h.

10.23.3.43 interval coco::basic_alltype::ni ( ) const [inline]

retrieve an interval from the basic_alltype

Definition at line 1073 of file basic_alltype.h.

10.23.3.44 interval_st& coco::basic_alltype::ni ( ) [inline]

retrieve an interval from the basic_alltype

Definition at line 1345 of file basic_alltype.h.

10.23.3.45 int coco::basic_alltype::nn ( ) const [inline]

retrieve an int from the basic_alltype

Definition at line 1052 of file basic_alltype.h.

10.23.3.46 int& coco::basic_alltype::nn ( ) [inline]

retrieve an int from the basic_alltype

Definition at line 1324 of file basic_alltype.h.

10.23.3.47 const vmtl::sparse_matrix<int>& coco::basic_alltype::nsm ( ) const [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1308 of file basic_alltype.h.

10.23.3.48 vmtl::sparse_matrix<int>& coco::basic_alltype::nsm ( ) [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1579 of file basic_alltype.h.

10.23.3.49 unsigned int coco::basic_alltype::nu ( ) const [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1061 of file basic_alltype.h.

10.23.3.50 unsigned int& coco::basic_alltype::nu ( ) [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1333 of file basic_alltype.h.

10.23.3.51 bool coco::basic_alltype::operator!= (const basic_alltype & *b*) const

Standard comparison operator

10.23.3.52 basic_alltype& coco::basic_alltype::operator= (bool *__x*) [inline]

Assignment Operator, assigning the bool value ___x

Definition at line 647 of file basic_alltype.h.

10.23.3.53 basic_alltype& coco::basic_alltype::operator= (int *__x*) [inline]

Assignment Operator, assigning the int value ___x

Definition at line 653 of file basic_alltype.h.

10.23.3.54 basic_alltype& coco::basic_alltype::operator= (unsigned int *__x*) [inline]

Assignment Operator, assigning the unsigned int value ___x

Definition at line 659 of file basic_alltype.h.

10.23.3.55 basic_alltype& coco::basic_alltype::operator= (double *__x*) [inline]

Assignment Operator, assigning the double value ___x

Definition at line 665 of file basic_alltype.h.

10.23.3.56 basic_alltype& coco::basic_alltype::operator= (interval *__x*) [inline]

Assignment Operator, assigning the interval value ___x

Definition at line 671 of file basic_alltype.h.

10.23.3.57 basic_alltype& coco::basic_alltype::operator= (void * *__x*) [inline]

Assignment Operator, assigning the void* value ___x

Definition at line 677 of file basic_alltype.h.

10.23.3.58 basic_alltype& coco::basic_alltype::operator= (const std::string & *__x*) [inline]

Assignment Operator, assigning the string value ___x

Definition at line 683 of file basic_alltype.h.

10.23.3.59 basic_alltype& coco::basic_alltype::operator= (const char * *__x*) [inline]

Assignment Operator, assigning the string value ___x

Definition at line 690 of file basic_alltype.h.

10.23.3.60 basic_alltype& coco::basic_alltype::operator= (const num::Number & *__x*) [inline]

Assignment Operator, assigning the Number value ___x

Definition at line 697 of file basic_alltype.h.

10.23.3.61 basic_alltype& coco::basic_alltype::operator= (const std::vector< bool > & __x) [inline]

Assignment Operator, assigning the vector<bool> value ___x

Definition at line 704 of file basic_alltype.h.

10.23.3.62 basic_alltype& coco::basic_alltype::operator= (const std::vector< int > & __x) [inline]

Assignment Operator, assigning the vector<int> value ___x

Definition at line 711 of file basic_alltype.h.

10.23.3.63 basic_alltype& coco::basic_alltype::operator= (const std::vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the vector<unsigned int> value ___x

Definition at line 718 of file basic_alltype.h.

10.23.3.64 basic_alltype& coco::basic_alltype::operator= (const std::vector< double > & __x) [inline]

Assignment Operator, assigning the vector<double> value ___x

Definition at line 725 of file basic_alltype.h.

10.23.3.65 basic_alltype& coco::basic_alltype::operator= (const std::vector< interval > & __x) [inline]

Assignment Operator, assigning the vector<interval> value ___x

Definition at line 732 of file basic_alltype.h.

10.23.3.66 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::string > & __x) [inline]

Assignment Operator, assigning the vector<string> value ___x

Definition at line 739 of file basic_alltype.h.

10.23.3.67 basic_alltype& coco::basic_alltype::operator= (const std::vector< num::Number > & __x) [inline]

Assignment Operator, assigning the vector<Number> value ___x

Definition at line 746 of file basic_alltype.h.

10.23.3.68 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::vector< bool > > & __x) [inline]

Assignment Operator, assigning the vector<vector<bool> > value ___x

Definition at line 753 of file basic_alltype.h.

10.23.3.69 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::vector< int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<int> > value ___x

Definition at line 760 of file basic_alltype.h.

10.23.3.70 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::vector< unsigned int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<unsigned int> > value ___x

Definition at line 767 of file basic_alltype.h.

10.23.3.71 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::vector< double > > & __x) [inline]

Assignment Operator, assigning the vector<vector<double> > value ___x

Definition at line 774 of file basic_alltype.h.

10.23.3.72 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::vector< interval > > & __x) [inline]

Assignment Operator, assigning the vector<vector<interval> > value ___x

Definition at line 781 of file basic_alltype.h.

10.23.3.73 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::vector< std::string > > & __x) [inline]

Assignment Operator, assigning the vector<vector<string> > value ___x

Definition at line 788 of file basic_alltype.h.

10.23.3.74 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::vector< num::Number > > & __x) [inline]

Assignment Operator, assigning the vector<vector<Number> > value ___x

Definition at line 795 of file basic_alltype.h.

10.23.3.75 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_vector< bool > & __x) [inline]

Assignment Operator, assigning the sparse_vector<bool> value ___x

Definition at line 802 of file basic_alltype.h.

10.23.3.76 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_vector< int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<int> value ___x

Definition at line 809 of file basic_alltype.h.

10.23.3.77 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<unsigned int> value __x

Definition at line 816 of file basic_alltype.h.

10.23.3.78 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_vector< double > & __x) [inline]

Assignment Operator, assigning the sparse_vector<double> value __x

Definition at line 823 of file basic_alltype.h.

10.23.3.79 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_vector< interval > & __x) [inline]

Assignment Operator, assigning the sparse_vector<interval> value __x

Definition at line 830 of file basic_alltype.h.

10.23.3.80 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_vector< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_vector<string> value __x

Definition at line 837 of file basic_alltype.h.

10.23.3.81 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_vector< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_vector<Number> value __x

Definition at line 844 of file basic_alltype.h.

10.23.3.82 basic_alltype& coco::basic_alltype::operator= (const vmtl::dense_matrix< double > & __x) [inline]

Assignment Operator, assigning the dense_matrix<double> value __x

Definition at line 851 of file basic_alltype.h.

10.23.3.83 basic_alltype& coco::basic_alltype::operator= (const vmtl::dense_matrix< int > & __x) [inline]

Assignment Operator, assigning the dense_matrix<int> value __x

Definition at line 858 of file basic_alltype.h.

10.23.3.84 basic_alltype& coco::basic_alltype::operator= (const vmtl::dense_matrix< interval > & __x) [inline]

Assignment Operator, assigning the dense_matrix<interval> value __x

Definition at line 865 of file basic_alltype.h.

10.23.3.85 basic_alltype& coco::basic_alltype::operator= (const vmtl::dense_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the dense_matrix<string> value __x

Definition at line 872 of file basic_alltype.h.

10.23.3.86 basic_alltype& coco::basic_alltype::operator= (const vmtl::dense_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the dense_matrix<Number> value __x

Definition at line 879 of file basic_alltype.h.

10.23.3.87 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_matrix< double > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<double> value __x

Definition at line 886 of file basic_alltype.h.

10.23.3.88 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_matrix< int > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<int> value __x

Definition at line 893 of file basic_alltype.h.

10.23.3.89 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_matrix< interval > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<interval> value __x

Definition at line 900 of file basic_alltype.h.

10.23.3.90 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<string> value __x

Definition at line 907 of file basic_alltype.h.

10.23.3.91 basic_alltype& coco::basic_alltype::operator= (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<Number> value __x

Definition at line 914 of file basic_alltype.h.

10.23.3.92 basic_alltype& coco::basic_alltype::operator= (const basic_alltype & __a) [inline]

Standard Assignment Operator

Definition at line 921 of file basic_alltype.h.

10.23.3.93 bool coco::basic_alltype::operator== (const basic_alltype & *b*) const

Standard comparison operator

10.23.3.94 const void* coco::basic_alltype::p ( ) const [inline]

retrieve a void* from the basic_alltype

Definition at line 1076 of file basic_alltype.h.

10.23.3.95 void*& coco::basic_alltype::p ( ) [inline]

retrieve a void* reference from the basic_alltype

Definition at line 1348 of file basic_alltype.h.

10.23.3.96 const std::string& coco::basic_alltype::s ( ) const [inline]

retrieve a string from the basic_alltype

Definition at line 1079 of file basic_alltype.h.

10.23.3.97 std::string& coco::basic_alltype::s ( ) [inline]

retrieve a string from the basic_alltype

Definition at line 1351 of file basic_alltype.h.

10.23.3.98 const vmtl::sparse_vector<bool>& coco::basic_alltype::sb ( ) const [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1224 of file basic_alltype.h.

10.23.3.99 vmtl::sparse_vector<bool>& coco::basic_alltype::sb ( ) [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1495 of file basic_alltype.h.

10.23.3.100 const vmtl::sparse_vector<std::string>& coco::basic_alltype::sc ( ) const [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1280 of file basic_alltype.h.

10.23.3.101 vmtl::sparse_vector<std::string>& coco::basic_alltype::sc ( ) [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1551 of file basic_alltype.h.

10.23.3.102 const vmtl::sparse_vector<double>& coco::basic_alltype::sd ( ) const [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1272 of file basic_alltype.h.

10.23.3.103 vmtl::sparse_vector<double>& coco::basic_alltype::sd ( ) [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1543 of file basic_alltype.h.

10.23.3.104 basic_alltype& coco::basic_alltype::set_cdm (vmtl::dense_matrix< std::string > * __m) [inline]

This method assigns a dense_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<string>.

Definition at line 973 of file basic_alltype.h.

10.23.3.105 basic_alltype& coco::basic_alltype::set_csm (vmtl::sparse_matrix< std::string > * __m) [inline]

This method assigns a sparse_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<string>.

Definition at line 1028 of file basic_alltype.h.

10.23.3.106 basic_alltype& coco::basic_alltype::set_dm (vmtl::dense_matrix< double > * __m) [inline]

This method assigns a dense_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<double>.

Definition at line 940 of file basic_alltype.h.

10.23.3.107 basic_alltype& coco::basic_alltype::set_idm (vmtl::dense_matrix< interval > * __m) [inline]

This method assigns a dense_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<interval>.

Definition at line 962 of file basic_alltype.h.

10.23.3.108 basic_alltype& coco::basic_alltype::set_ism (vmtl::sparse_matrix< interval > * __m) [inline]

This method assigns a sparse_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<interval>.

Definition at line 1017 of file basic_alltype.h.

10.23.3.109 basic_alltype& coco::basic_alltype::set_ndm (vmtl::dense_matrix< int > * __m) [inline]

This method assigns a dense_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<int>.

Definition at line 951 of file basic_alltype.h.

10.23.3.110 basic_alltype& coco::basic_alltype::set_nsm (vmtl::sparse_matrix< int > * __m) [inline]

This method assigns a sparse_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<int>.

Definition at line 1006 of file basic_alltype.h.

10.23.3.111 basic_alltype& coco::basic_alltype::set_sm (vmtl::sparse_matrix< double > * __m) [inline]

This method assigns a sparse_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<double>.

Definition at line 995 of file basic_alltype.h.

10.23.3.112 basic_alltype& coco::basic_alltype::set_xdm (vmtl::dense_matrix< num::Number > * __m) [inline]

This method assigns a dense_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<Number>.

Definition at line 984 of file basic_alltype.h.

10.23.3.113 basic_alltype& coco::basic_alltype::set_xsm (vmtl::sparse_matrix< num::Number > * __m) [inline]

This method assigns a sparse_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<Number>.

Definition at line 1039 of file basic_alltype.h.

10.23.3.114 const vmtl::sparse_vector<interval>& coco::basic_alltype::si ( ) const [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1276 of file basic_alltype.h.

10.23.3.115 vmtl::sparse_vector<interval>& coco::basic_alltype::si ( ) [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1547 of file basic_alltype.h.

10.23.3.116 const vmtl::sparse_matrix<double>& coco::basic_alltype::sm ( ) const [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1305 of file basic_alltype.h.

10.23.3.117 vmtl::sparse_matrix<double>& coco::basic_alltype::sm ( ) [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1576 of file basic_alltype.h.

10.23.3.118 const vmtl::sparse_vector<int>& coco::basic_alltype::sn ( ) const [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1228 of file basic_alltype.h.

10.23.3.119 vmtl::sparse_vector<int>& coco::basic_alltype::sn ( ) [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1499 of file basic_alltype.h.

10.23.3.120 const vmtl::sparse_vector<unsigned int>& coco::basic_alltype::su ( ) const [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1250 of file basic_alltype.h.

10.23.3.121 vmtl::sparse_vector<unsigned int>& coco::basic_alltype::su ( ) [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1521 of file basic_alltype.h.

10.23.3.122 const vmtl::sparse_vector<num::Number>& coco::basic_alltype::sx ( ) const [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1284 of file basic_alltype.h.

10.23.3.123 vmtl::sparse_vector<num::Number>& coco::basic_alltype::sx ( ) [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1555 of file basic_alltype.h.

10.23.3.124 const char* coco::basic_alltype::type_cstr ( ) const [inline]

Return the type of the value contained in the storage as a C-string

Definition at line 1674 of file basic_alltype.h.

10.23.3.125 std::string coco::basic_alltype::type_name ( ) const [inline]

Return the type of the value contained in the storage as a C++-string

Definition at line 1671 of file basic_alltype.h.

10.23.3.126 const std::vector<unsigned int>& coco::basic_alltype::u ( ) const [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1111 of file basic_alltype.h.

10.23.3.127 std::vector<unsigned int>& coco::basic_alltype::u ( ) [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1383 of file basic_alltype.h.

10.23.3.128 const std::vector<std::vector<bool> >& coco::basic_alltype::vb ( ) const [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1150 of file basic_alltype.h.

10.23.3.129 std::vector<std::vector<bool> >& coco::basic_alltype::vb ( ) [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1422 of file basic_alltype.h.

10.23.3.130 const std::vector<std::vector<std::string> >& coco::basic_alltype::vc ( ) const [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1214 of file basic_alltype.h.

10.23.3.131 std::vector<std::vector<std::string> >& coco::basic_alltype::vc ( ) [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1486 of file basic_alltype.h.

10.23.3.132 const std::vector<std::vector<double> >& coco::basic_alltype::vd ( ) const [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1206 of file basic_alltype.h.

10.23.3.133 std::vector<std::vector<double> >& coco::basic_alltype::vd ( ) [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1478 of file basic_alltype.h.

10.23.3.134 const std::vector<std::vector<interval> >& coco::basic_alltype::vi ( ) const [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1210 of file basic_alltype.h.

10.23.3.135 std::vector<std::vector<interval> >& coco::basic_alltype::vi ( ) [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1482 of file basic_alltype.h.

10.23.3.136 const std::vector<std::vector<int> >& coco::basic_alltype::vn ( ) const [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1154 of file basic_alltype.h.

10.23.3.137 std::vector<std::vector<int> >& coco::basic_alltype::vn ( ) [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1426 of file basic_alltype.h.

10.23.3.138 const std::vector<std::vector<unsigned int> >& coco::basic_alltype::vu ( ) const [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1180 of file basic_alltype.h.

10.23.3.139 std::vector<std::vector<unsigned int> >& coco::basic_alltype::vu ( ) [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1452 of file basic_alltype.h.

10.23.3.140 const std::vector<std::vector<num::Number> >& coco::basic_alltype::vx ( ) const [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1218 of file basic_alltype.h.

10.23.3.141 std::vector<std::vector<num::Number> >& coco::basic_alltype::vx ( ) [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1490 of file basic_alltype.h.

10.23.3.142 const std::vector<num::Number>& coco::basic_alltype::x ( ) const [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1145 of file basic_alltype.h.

10.23.3.143 std::vector<num::Number>& coco::basic_alltype::x ( ) [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1417 of file basic_alltype.h.

10.23.3.144 const vmtl::dense_matrix<num::Number>& coco::basic_alltype::xdm ( ) const [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1301 of file basic_alltype.h.

10.23.3.145 vmtl::dense_matrix<num::Number>& coco::basic_alltype::xdm ( ) [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1572 of file basic_alltype.h.

10.23.3.146 const vmtl::sparse_matrix<num::Number>& coco::basic_alltype::xsm ( ) const [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1317 of file basic_alltype.h.

10.23.3.147 vmtl::sparse_matrix<num::Number>& coco::basic_alltype::xsm ( ) [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1588 of file basic_alltype.h.

10.23.3.148 const num::Number& coco::basic_alltype::y ( ) const [inline]

retrieve a Number from the basic_alltype

Definition at line 1082 of file basic_alltype.h.

10.23.3.149 num::Number& coco::basic_alltype::y ( ) [inline]

retrieve a Number from the basic_alltype

Definition at line 1354 of file basic_alltype.h.

10.23.4 Member Data Documentation

10.23.4.1 std::vector<bool>* coco::basic_alltype::b

Definition at line 191 of file basic_alltype.h.

10.23.4.2 std::vector<std::string>* coco::basic_alltype::c

Definition at line 196 of file basic_alltype.h.

10.23.4.3 vmtl::dense_matrix<std::string>* coco::basic_alltype::cdm

Definition at line 215 of file basic_alltype.h.

10.23.4.4 vmtl::sparse_matrix<std::string>* coco::basic_alltype::csm

Definition at line 220 of file basic_alltype.h.

10.23.4.5 std::vector<double>* coco::basic_alltype::d

Definition at line 194 of file basic_alltype.h.

10.23.4.6 vmtl::dense_matrix<double>* coco::basic_alltype::dm

Definition at line 212 of file basic_alltype.h.

10.23.4.7 std::vector<interval>* coco::basic_alltype::i

Definition at line 195 of file basic_alltype.h.

10.23.4.8 vmtl::dense_matrix<interval>* coco::basic_alltype::idm

Definition at line 214 of file basic_alltype.h.

10.23.4.9 vmtl::sparse_matrix<interval>* coco::basic_alltype::ism

Definition at line 219 of file basic_alltype.h.

10.23.4.10 std::vector<int>* coco::basic_alltype::n

Definition at line 192 of file basic_alltype.h.

10.23.4.11 bool coco::basic_alltype::nb

Definition at line 183 of file basic_alltype.h.

10.23.4.12 double coco::basic_alltype::nd

Definition at line 186 of file basic_alltype.h.

10.23.4.13 vmtl::dense_matrix<int>* coco::basic_alltype::ndm

Definition at line 213 of file basic_alltype.h.

10.23.4.14 interval_st coco::basic_alltype::ni

Definition at line 187 of file basic_alltype.h.

10.23.4.15 int coco::basic_alltype::nn

Definition at line 184 of file basic_alltype.h.

10.23.4.16 vmtl::sparse_matrix<int>* coco::basic_alltype::nsm

Definition at line 218 of file basic_alltype.h.

10.23.4.17 unsigned int coco::basic_alltype::nu

Definition at line 185 of file basic_alltype.h.

10.23.4.18 void* coco::basic_alltype::p

Definition at line 188 of file basic_alltype.h.

10.23.4.19 std::string* coco::basic_alltype::s

Definition at line 189 of file basic_alltype.h.

10.23.4.20 vmtl::sparse_vector<bool>* coco::basic_alltype::sb

Definition at line 205 of file basic_alltype.h.

10.23.4.21 vmtl::sparse_vector<std::string>* coco::basic_alltype::sc

Definition at line 210 of file basic_alltype.h.

10.23.4.22 vmtl::sparse_vector<double>* coco::basic_alltype::sd

Definition at line 208 of file basic_alltype.h.

10.23.4.23 vmtl::sparse_vector<interval>* coco::basic_alltype::si

Definition at line 209 of file basic_alltype.h.

10.23.4.24 vmtl::sparse_matrix<double>* coco::basic_alltype::sm

Definition at line 217 of file basic_alltype.h.

10.23.4.25 vmtl::sparse_vector<int>* coco::basic_alltype::sn

Definition at line 206 of file basic_alltype.h.

10.23.4.26 vmtl::sparse_vector<unsigned int>* coco::basic_alltype::su

Definition at line 207 of file basic_alltype.h.

10.23.4.27 vmtl::sparse_vector<num::Number>* coco::basic_alltype::sx

Definition at line 211 of file basic_alltype.h.

10.23.4.28 std::vector<unsigned int>* coco::basic_alltype::u

Definition at line 193 of file basic_alltype.h.

10.23.4.29 std::vector<std::vector<bool> >* coco::basic_alltype::vb

Definition at line 198 of file basic_alltype.h.

10.23.4.30 std::vector<std::vector<std::string> >* coco::basic_alltype::vc

Definition at line 203 of file basic_alltype.h.

10.23.4.31 std::vector<std::vector<double> >* coco::basic_alltype::vd

Definition at line 201 of file basic_alltype.h.

10.23.4.32 std::vector<std::vector<interval> >* coco::basic_alltype::vi

Definition at line 202 of file basic_alltype.h.

10.23.4.33 std::vector<std::vector<int> >* coco::basic_alltype::vn

Definition at line 199 of file basic_alltype.h.

10.23.4.34 std::vector<std::vector<unsigned int> >* coco::basic_alltype::vu

Definition at line 200 of file basic_alltype.h.

10.23.4.35 std::vector<std::vector<num::Number> >* coco::basic_alltype::vx

Definition at line 204 of file basic_alltype.h.

10.23.4.36 std::vector<num::Number>* coco::basic_alltype::x

Definition at line 197 of file basic_alltype.h.

10.23.4.37 vmtl::dense_matrix<num::Number>* coco::basic_alltype::xdm

Definition at line 216 of file basic_alltype.h.

10.23.4.38 vmtl::sparse_matrix<num::Number>* coco::basic_alltype::xsm

Definition at line 221 of file basic_alltype.h.

10.23.4.39 num::Number* coco::basic_alltype::y

Definition at line 190 of file basic_alltype.h.

The documentation for this class was generated from the following file:

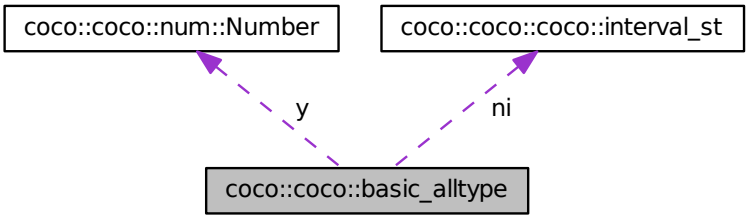
- basic_alltype.h

10.24 coco::coco::basic_alltype Class Reference

The basic alltype which can hold any of a number of basic types.

`#include <expression.h>`

Collaboration diagram for coco::coco::basic_alltype:



Public Member Functions

- basic_alltype ()
- basic_alltype (bool __x)
- basic_alltype (int __x)
- basic_alltype (unsigned int __x)
- basic_alltype (double __x)
- basic_alltype (interval __x)
- basic_alltype (void *__x)
- basic_alltype (const char *__cp)
- basic_alltype (const std::string &__x)
- basic_alltype (const num::Number &__y)
- basic_alltype (const std::vector< bool > &__x)
- basic_alltype (const std::vector< int > &__x)
- basic_alltype (const std::vector< unsigned int > &__x)
- basic_alltype (const std::vector< double > &__x)
- basic_alltype (const std::vector< interval > &__x)
- basic_alltype (const std::vector< std::string > &__x)
- basic_alltype (const std::vector< num::Number > &__x)
- basic_alltype (const std::vector< std::vector< bool > > &__vx)
- basic_alltype (const std::vector< std::vector< int > > &__vx)
- basic_alltype (const std::vector< std::vector< unsigned int > > &__vx)
- basic_alltype (const std::vector< std::vector< double > > &__vx)
- basic_alltype (const std::vector< std::vector< interval > > &__vx)
- basic_alltype (const std::vector< std::vector< std::string > > &__vx)
- basic_alltype (const std::vector< std::vector< num::Number > > &__vx)
- basic_alltype (const vmtl::sparse_vector< bool > &__x)
- basic_alltype (const vmtl::sparse_vector< int > &__x)
- basic_alltype (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype (const vmtl::sparse_vector< double > &__x)
- basic_alltype (const vmtl::sparse_vector< interval > &__x)
- basic_alltype (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype (const vmtl::sparse_vector< num::Number > &__x)

- basic_alltype (const vmtl::dense_matrix< double > &__x)
- basic_alltype (const vmtl::dense_matrix< int > &__x)
- basic_alltype (const vmtl::dense_matrix< interval > &__x)
- basic_alltype (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype (const vmtl::sparse_matrix< double > &__x)
- basic_alltype (const vmtl::sparse_matrix< int > &__x)
- basic_alltype (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype (const vmtl::sparse_matrix< num::Number > &__x)
- ~basic_alltype ()
- basic_alltype (const basic_alltype &__a)
- basic_alltype & operator= (bool __x)
- basic_alltype & operator= (int __x)
- basic_alltype & operator= (unsigned int __x)
- basic_alltype & operator= (double __x)
- basic_alltype & operator= (interval __x)
- basic_alltype & operator= (void *__x)
- basic_alltype & operator= (const std::string &__x)
- basic_alltype & operator= (const char *__x)
- basic_alltype & operator= (const num::Number &__x)
- basic_alltype & operator= (const std::vector< bool > &__x)
- basic_alltype & operator= (const std::vector< int > &__x)
- basic_alltype & operator= (const std::vector< unsigned int > &__x)
- basic_alltype & operator= (const std::vector< double > &__x)
- basic_alltype & operator= (const std::vector< interval > &__x)
- basic_alltype & operator= (const std::vector< std::string > &__x)
- basic_alltype & operator= (const std::vector< num::Number > &__x)
- basic_alltype & operator= (const std::vector< std::vector< bool > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< unsigned int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< double > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< interval > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< std::string > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< num::Number > > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< bool > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< interval > &__x)

- basic_alltype & operator= (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< num::Number > &__x)
- basic_alltype & operator= (const basic_alltype &__a)
- basic_alltype & clear ()
- basic_alltype & set_dm (vmtl::dense_matrix< double > *__m)
- basic_alltype & set_ndm (vmtl::dense_matrix< int > *__m)
- basic_alltype & set_idm (vmtl::dense_matrix< interval > *__m)
- basic_alltype & set_cdm (vmtl::dense_matrix< std::string > *__m)
- basic_alltype & set_xdm (vmtl::dense_matrix< num::Number > *__m)
- basic_alltype & set_sm (vmtl::sparse_matrix< double > *__m)
- basic_alltype & set_nsm (vmtl::sparse_matrix< int > *__m)
- basic_alltype & set_ism (vmtl::sparse_matrix< interval > *__m)
- basic_alltype & set_csm (vmtl::sparse_matrix< std::string > *__m)
- basic_alltype & set_xsm (vmtl::sparse_matrix< num::Number > *__m)
- bool nb () const BASIC_ALLTYPE_THROW
- int nn () const BASIC_ALLTYPE_THROW
- unsigned int nu () const BASIC_ALLTYPE_THROW
- double nd () const BASIC_ALLTYPE_THROW
- interval ni () const BASIC_ALLTYPE_THROW
- const void * p () const BASIC_ALLTYPE_THROW
- const std::string & s () const BASIC_ALLTYPE_THROW
- const num::Number & y () const BASIC_ALLTYPE_THROW
- const std::vector< bool > & b () const BASIC_ALLTYPE_THROW
- const std::vector< int > & n () const BASIC_ALLTYPE_THROW
- const std::vector< unsigned int > & u () const BASIC_ALLTYPE_THROW
- const std::vector< double > & d () const BASIC_ALLTYPE_THROW
- const std::vector< interval > & i () const BASIC_ALLTYPE_THROW
- const std::vector< std::string > & c () const BASIC_ALLTYPE_THROW
- const std::vector< num::Number > & x () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < bool > > & vb () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < int > > & vn () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < unsigned int > > & vu () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < double > > & vd () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < interval > > & vi () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < std::string > > & vc () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < num::Number > > & vx () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< bool > & sb () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< int > & sn () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < unsigned int > & su () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < double > & sd () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < interval > & si () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < std::string > & sc () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < num::Number > & sx () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < double > & dm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix< int > & ndm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < interval > & idm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < std::string > & cdm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < num::Number > & xdm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < double > & sm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix< int > & nsm () const BASIC_ALLTYPE_THROW

- const vmtl::sparse_matrix < interval > & ism () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < std::string > & csm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < num::Number > & xsm () const BASIC_ALLTYPE_THROW
- bool & nb () BASIC_ALLTYPE_THROW
- int & nn () BASIC_ALLTYPE_THROW
- unsigned int & nu () BASIC_ALLTYPE_THROW
- double & nd () BASIC_ALLTYPE_THROW
- interval_st & ni () BASIC_ALLTYPE_THROW
- void *& p () BASIC_ALLTYPE_THROW
- std::string & s () BASIC_ALLTYPE_THROW
- num::Number & y () BASIC_ALLTYPE_THROW
- std::vector< bool > & b () BASIC_ALLTYPE_THROW
- std::vector< int > & n () BASIC_ALLTYPE_THROW
- std::vector< unsigned int > & u () BASIC_ALLTYPE_THROW
- std::vector< double > & d () BASIC_ALLTYPE_THROW
- std::vector< interval > & i () BASIC_ALLTYPE_THROW
- std::vector< std::string > & c () BASIC_ALLTYPE_THROW
- std::vector< num::Number > & x () BASIC_ALLTYPE_THROW
- std::vector< std::vector< bool > > & vb () BASIC_ALLTYPE_THROW
- std::vector< std::vector< int > > & vn () BASIC_ALLTYPE_THROW
- std::vector< std::vector < unsigned int > > & vu () BASIC_ALLTYPE_THROW
- std::vector< std::vector < double > > & vd () BASIC_ALLTYPE_THROW
- std::vector< std::vector < interval > > & vi () BASIC_ALLTYPE_THROW
- std::vector< std::vector < std::string > > & vc () BASIC_ALLTYPE_THROW
- std::vector< std::vector < num::Number > > & vx () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< bool > & sb () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< int > & sn () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< unsigned int > & su () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< double > & sd () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< interval > & si () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < std::string > & sc () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < num::Number > & sx () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< double > & dm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< int > & ndm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< interval > & idm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< std::string > & cdm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< num::Number > & xdm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< double > & sm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< int > & nsm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< interval > & ism () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < std::string > & csm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < num::Number > & xsm () BASIC_ALLTYPE_THROW
- bool is_allocated () const
- bool is_scalar () const
- bool is_vector () const
- bool is_vector_of_vector () const
- bool is_dense_vector () const
- bool is_sparse_vector () const
- bool is_empty_vector () const
- bool is_matrix () const

- bool is_dense_matrix () const
- bool is_sparse_matrix () const
- bool is_empty_matrix () const
- bool empty () const
- int contents_type () const
- std::string type_name () const
- const char * type_cstr () const
- bool operator== (const basic_alltype &b) const
- bool operator!= (const basic_alltype &b) const
- bool less_than (const basic_alltype &b) const
- bool less_equal (const basic_alltype &b) const
- basic_alltype ()
- basic_alltype (bool __x)
- basic_alltype (int __x)
- basic_alltype (unsigned int __x)
- basic_alltype (double __x)
- basic_alltype (interval __x)
- basic_alltype (void *__x)
- basic_alltype (const char *__cp)
- basic_alltype (const std::string &__x)
- basic_alltype (const num::Number &__y)
- basic_alltype (const std::vector< bool > &__x)
- basic_alltype (const std::vector< int > &__x)
- basic_alltype (const std::vector< unsigned int > &__x)
- basic_alltype (const std::vector< double > &__x)
- basic_alltype (const std::vector< interval > &__x)
- basic_alltype (const std::vector< std::string > &__x)
- basic_alltype (const std::vector< num::Number > &__x)
- basic_alltype (const std::vector< std::vector< bool > > &__vx)
- basic_alltype (const std::vector< std::vector< int > > &__vx)
- basic_alltype (const std::vector< std::vector< unsigned int > > &__vx)
- basic_alltype (const std::vector< std::vector< double > > &__vx)
- basic_alltype (const std::vector< std::vector< interval > > &__vx)
- basic_alltype (const std::vector< std::vector< std::string > > &__vx)
- basic_alltype (const std::vector< std::vector< num::Number > > &__vx)
- basic_alltype (const vmtl::sparse_vector< bool > &__x)
- basic_alltype (const vmtl::sparse_vector< int > &__x)
- basic_alltype (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype (const vmtl::sparse_vector< double > &__x)
- basic_alltype (const vmtl::sparse_vector< interval > &__x)
- basic_alltype (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype (const vmtl::dense_matrix< double > &__x)
- basic_alltype (const vmtl::dense_matrix< int > &__x)
- basic_alltype (const vmtl::dense_matrix< interval > &__x)
- basic_alltype (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype (const vmtl::sparse_matrix< double > &__x)
- basic_alltype (const vmtl::sparse_matrix< int > &__x)
- basic_alltype (const vmtl::sparse_matrix< interval > &__x)

- basic_alltype (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype (const vmtl::sparse_matrix< num::Number > &__x)
- ~basic_alltype ()
- basic_alltype (const basic_alltype &__a)
- basic_alltype & operator= (bool __x)
- basic_alltype & operator= (int __x)
- basic_alltype & operator= (unsigned int __x)
- basic_alltype & operator= (double __x)
- basic_alltype & operator= (interval __x)
- basic_alltype & operator= (void *__x)
- basic_alltype & operator= (const std::string &__x)
- basic_alltype & operator= (const char *__x)
- basic_alltype & operator= (const num::Number &__x)
- basic_alltype & operator= (const std::vector< bool > &__x)
- basic_alltype & operator= (const std::vector< int > &__x)
- basic_alltype & operator= (const std::vector< unsigned int > &__x)
- basic_alltype & operator= (const std::vector< double > &__x)
- basic_alltype & operator= (const std::vector< interval > &__x)
- basic_alltype & operator= (const std::vector< std::string > &__x)
- basic_alltype & operator= (const std::vector< num::Number > &__x)
- basic_alltype & operator= (const std::vector< std::vector< bool > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< unsigned int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< double > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< interval > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< std::string > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< num::Number > > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< bool > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< num::Number > &__x)
- basic_alltype & operator= (const basic_alltype &__a)
- basic_alltype & clear ()
- basic_alltype & set_dm (vmtl::dense_matrix< double > *__m)
- basic_alltype & set_ndm (vmtl::dense_matrix< int > *__m)
- basic_alltype & set_idm (vmtl::dense_matrix< interval > *__m)
- basic_alltype & set_cdm (vmtl::dense_matrix< std::string > *__m)

- basic_alltype & set_xdm (vmtl::dense_matrix< num::Number > *__m)
- basic_alltype & set_sm (vmtl::sparse_matrix< double > *__m)
- basic_alltype & set_nsm (vmtl::sparse_matrix< int > *__m)
- basic_alltype & set_ism (vmtl::sparse_matrix< interval > *__m)
- basic_alltype & set_csm (vmtl::sparse_matrix< std::string > *__m)
- basic_alltype & set_xsm (vmtl::sparse_matrix< num::Number > *__m)
- bool nb () const BASIC_ALLTYPE_THROW
- int nn () const BASIC_ALLTYPE_THROW
- unsigned int nu () const BASIC_ALLTYPE_THROW
- double nd () const BASIC_ALLTYPE_THROW
- interval ni () const BASIC_ALLTYPE_THROW
- const void * p () const BASIC_ALLTYPE_THROW
- const std::string & s () const BASIC_ALLTYPE_THROW
- const num::Number & y () const BASIC_ALLTYPE_THROW
- const std::vector< bool > & b () const BASIC_ALLTYPE_THROW
- const std::vector< int > & n () const BASIC_ALLTYPE_THROW
- const std::vector< unsigned int > & u () const BASIC_ALLTYPE_THROW
- const std::vector< double > & d () const BASIC_ALLTYPE_THROW
- const std::vector< interval > & i () const BASIC_ALLTYPE_THROW
- const std::vector< std::string > & c () const BASIC_ALLTYPE_THROW
- const std::vector< num::Number > & x () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < bool > > & vb () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < int > > & vn () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < unsigned int > > & vu () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < double > > & vd () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < interval > > & vi () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < std::string > > & vc () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < num::Number > > & vx () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< bool > & sb () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< int > & sn () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < unsigned int > & su () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < double > & sd () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < interval > & si () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < std::string > & sc () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < num::Number > & sx () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < double > & dm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix< int > & ndm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < interval > & idm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < std::string > & cdm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < num::Number > & xdm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < double > & sm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix< int > & nsm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < interval > & ism () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < std::string > & csm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < num::Number > & xsm () const BASIC_ALLTYPE_THROW
- bool & nb () BASIC_ALLTYPE_THROW
- int & nn () BASIC_ALLTYPE_THROW
- unsigned int & nu () BASIC_ALLTYPE_THROW
- double & nd () BASIC_ALLTYPE_THROW
- interval_st & ni () BASIC_ALLTYPE_THROW

- void *& p () BASIC_ALLTYPE_THROW
- std::string & s () BASIC_ALLTYPE_THROW
- num::Number & y () BASIC_ALLTYPE_THROW
- std::vector< bool > & b () BASIC_ALLTYPE_THROW
- std::vector< int > & n () BASIC_ALLTYPE_THROW
- std::vector< unsigned int > & u () BASIC_ALLTYPE_THROW
- std::vector< double > & d () BASIC_ALLTYPE_THROW
- std::vector< interval > & i () BASIC_ALLTYPE_THROW
- std::vector< std::string > & c () BASIC_ALLTYPE_THROW
- std::vector< num::Number > & x () BASIC_ALLTYPE_THROW
- std::vector< std::vector< bool > > & vb () BASIC_ALLTYPE_THROW
- std::vector< std::vector< int > > & vn () BASIC_ALLTYPE_THROW
- std::vector< std::vector < unsigned int > > & vu () BASIC_ALLTYPE_THROW
- std::vector< std::vector < double > > & vd () BASIC_ALLTYPE_THROW
- std::vector< std::vector < interval > > & vi () BASIC_ALLTYPE_THROW
- std::vector< std::vector < std::string > > & vc () BASIC_ALLTYPE_THROW
- std::vector< std::vector < num::Number > > & vx () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< bool > & sb () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< int > & sn () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< unsigned int > & su () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< double > & sd () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< interval > & si () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < std::string > & sc () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < num::Number > & sx () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< double > & dm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< int > & ndm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< interval > & idm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< std::string > & cdm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< num::Number > & xdm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< double > & sm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< int > & nsm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< interval > & ism () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < std::string > & csm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < num::Number > & xsm () BASIC_ALLTYPE_THROW
- bool is_allocated () const
- bool is_scalar () const
- bool is_vector () const
- bool is_vector_of_vector () const
- bool is_dense_vector () const
- bool is_sparse_vector () const
- bool is_empty_vector () const
- bool is_matrix () const
- bool is_dense_matrix () const
- bool is_sparse_matrix () const
- bool is_empty_matrix () const
- bool empty () const
- int contents_type () const
- std::string type_name () const
- const char * type_cstr () const
- bool operator== (const basic_alltype &b) const

- bool operator!= (const basic_alltype &b) const
- bool less_than (const basic_alltype &b) const
- bool less_equal (const basic_alltype &b) const
- basic_alltype ()
- basic_alltype (bool __x)
- basic_alltype (int __x)
- basic_alltype (unsigned int __x)
- basic_alltype (double __x)
- basic_alltype (interval __x)
- basic_alltype (void *__x)
- basic_alltype (const char *__cp)
- basic_alltype (const std::string &__x)
- basic_alltype (const num::Number &__y)
- basic_alltype (const std::vector< bool > &__x)
- basic_alltype (const std::vector< int > &__x)
- basic_alltype (const std::vector< unsigned int > &__x)
- basic_alltype (const std::vector< double > &__x)
- basic_alltype (const std::vector< interval > &__x)
- basic_alltype (const std::vector< std::string > &__x)
- basic_alltype (const std::vector< num::Number > &__x)
- basic_alltype (const std::vector< std::vector< bool > > &__vx)
- basic_alltype (const std::vector< std::vector< int > > &__vx)
- basic_alltype (const std::vector< std::vector< unsigned int > > &__vx)
- basic_alltype (const std::vector< std::vector< double > > &__vx)
- basic_alltype (const std::vector< std::vector< interval > > &__vx)
- basic_alltype (const std::vector< std::vector< std::string > > &__vx)
- basic_alltype (const std::vector< std::vector< num::Number > > &__vx)
- basic_alltype (const vmtl::sparse_vector< bool > &__x)
- basic_alltype (const vmtl::sparse_vector< int > &__x)
- basic_alltype (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype (const vmtl::sparse_vector< double > &__x)
- basic_alltype (const vmtl::sparse_vector< interval > &__x)
- basic_alltype (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype (const vmtl::dense_matrix< double > &__x)
- basic_alltype (const vmtl::dense_matrix< int > &__x)
- basic_alltype (const vmtl::dense_matrix< interval > &__x)
- basic_alltype (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype (const vmtl::sparse_matrix< double > &__x)
- basic_alltype (const vmtl::sparse_matrix< int > &__x)
- basic_alltype (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype (const vmtl::sparse_matrix< num::Number > &__x)
- ~basic_alltype ()
- basic_alltype (const basic_alltype &__a)
- basic_alltype & operator= (bool __x)
- basic_alltype & operator= (int __x)
- basic_alltype & operator= (unsigned int __x)
- basic_alltype & operator= (double __x)

- basic_alltype & operator= (interval __x)
- basic_alltype & operator= (void *__x)
- basic_alltype & operator= (const std::string &__x)
- basic_alltype & operator= (const char *__x)
- basic_alltype & operator= (const num::Number &__x)
- basic_alltype & operator= (const std::vector< bool > &__x)
- basic_alltype & operator= (const std::vector< int > &__x)
- basic_alltype & operator= (const std::vector< unsigned int > &__x)
- basic_alltype & operator= (const std::vector< double > &__x)
- basic_alltype & operator= (const std::vector< interval > &__x)
- basic_alltype & operator= (const std::vector< std::string > &__x)
- basic_alltype & operator= (const std::vector< num::Number > &__x)
- basic_alltype & operator= (const std::vector< std::vector< bool > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< unsigned int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< double > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< interval > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< std::string > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< num::Number > > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< bool > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< num::Number > &__x)
- basic_alltype & operator= (const basic_alltype &__a)
- basic_alltype & clear ()
- basic_alltype & set_dm (vmtl::dense_matrix< double > *__m)
- basic_alltype & set_ndm (vmtl::dense_matrix< int > *__m)
- basic_alltype & set_idm (vmtl::dense_matrix< interval > *__m)
- basic_alltype & set_cdm (vmtl::dense_matrix< std::string > *__m)
- basic_alltype & set_xdm (vmtl::dense_matrix< num::Number > *__m)
- basic_alltype & set_sm (vmtl::sparse_matrix< double > *__m)
- basic_alltype & set_nsm (vmtl::sparse_matrix< int > *__m)
- basic_alltype & set_ism (vmtl::sparse_matrix< interval > *__m)
- basic_alltype & set_csm (vmtl::sparse_matrix< std::string > *__m)
- basic_alltype & set_xsm (vmtl::sparse_matrix< num::Number > *__m)
- bool nb () const BASIC_ALLTYPE_THROW
- int nn () const BASIC_ALLTYPE_THROW

- unsigned int nu () const BASIC_ALLTYPE_THROW
- double nd () const BASIC_ALLTYPE_THROW
- interval ni () const BASIC_ALLTYPE_THROW
- const void * p () const BASIC_ALLTYPE_THROW
- const std::string & s () const BASIC_ALLTYPE_THROW
- const num::Number & y () const BASIC_ALLTYPE_THROW
- const std::vector< bool > & b () const BASIC_ALLTYPE_THROW
- const std::vector< int > & n () const BASIC_ALLTYPE_THROW
- const std::vector< unsigned int > & u () const BASIC_ALLTYPE_THROW
- const std::vector< double > & d () const BASIC_ALLTYPE_THROW
- const std::vector< interval > & i () const BASIC_ALLTYPE_THROW
- const std::vector< std::string > & c () const BASIC_ALLTYPE_THROW
- const std::vector< num::Number > & x () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < bool > > & vb () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < int > > & vn () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < unsigned int > > & vu () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < double > > & vd () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < interval > > & vi () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < std::string > > & vc () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < num::Number > > & vx () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< bool > & sb () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< int > & sn () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < unsigned int > & su () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < double > & sd () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < interval > & si () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < std::string > & sc () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < num::Number > & sx () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < double > & dm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix< int > & ndm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < interval > & idm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < std::string > & cdm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < num::Number > & xdm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < double > & sm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix< int > & nsm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < interval > & ism () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < std::string > & csm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < num::Number > & xsm () const BASIC_ALLTYPE_THROW
- bool & nb () BASIC_ALLTYPE_THROW
- int & nn () BASIC_ALLTYPE_THROW
- unsigned int & nu () BASIC_ALLTYPE_THROW
- double & nd () BASIC_ALLTYPE_THROW
- interval_st & ni () BASIC_ALLTYPE_THROW
- void *& p () BASIC_ALLTYPE_THROW
- std::string & s () BASIC_ALLTYPE_THROW
- num::Number & y () BASIC_ALLTYPE_THROW
- std::vector< bool > & b () BASIC_ALLTYPE_THROW
- std::vector< int > & n () BASIC_ALLTYPE_THROW
- std::vector< unsigned int > & u () BASIC_ALLTYPE_THROW
- std::vector< double > & d () BASIC_ALLTYPE_THROW
- std::vector< interval > & i () BASIC_ALLTYPE_THROW

- std::vector< std::string > & c () BASIC_ALLTYPE_THROW
- std::vector< num::Number > & x () BASIC_ALLTYPE_THROW
- std::vector< std::vector< bool > > & vb () BASIC_ALLTYPE_THROW
- std::vector< std::vector< int > > & vn () BASIC_ALLTYPE_THROW
- std::vector< std::vector < unsigned int > > & vu () BASIC_ALLTYPE_THROW
- std::vector< std::vector < double > > & vd () BASIC_ALLTYPE_THROW
- std::vector< std::vector < interval > > & vi () BASIC_ALLTYPE_THROW
- std::vector< std::vector < std::string > > & vc () BASIC_ALLTYPE_THROW
- std::vector< std::vector < num::Number > > & vx () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< bool > & sb () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< int > & sn () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< unsigned int > & su () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< double > & sd () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< interval > & si () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < std::string > & sc () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < num::Number > & sx () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< double > & dm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< int > & ndm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< interval > & idm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< std::string > & cdm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< num::Number > & xdm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< double > & sm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< int > & nsm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< interval > & ism () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < std::string > & csm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < num::Number > & xsm () BASIC_ALLTYPE_THROW
- bool is_allocated () const
- bool is_scalar () const
- bool is_vector () const
- bool is_vector_of_vector () const
- bool is_dense_vector () const
- bool is_sparse_vector () const
- bool is_empty_vector () const
- bool is_matrix () const
- bool is_dense_matrix () const
- bool is_sparse_matrix () const
- bool is_empty_matrix () const
- bool empty () const
- int contents_type () const
- std::string type_name () const
- const char * type_cstr () const
- bool operator== (const basic_alltype &b) const
- bool operator!= (const basic_alltype &b) const
- bool less_than (const basic_alltype &b) const
- bool less_equal (const basic_alltype &b) const
- basic_alltype ()
- basic_alltype (bool __x)
- basic_alltype (int __x)
- basic_alltype (unsigned int __x)
- basic_alltype (double __x)

- basic_alltype (interval __x)
- basic_alltype (void *__x)
- basic_alltype (const char *__cp)
- basic_alltype (const std::string &__x)
- basic_alltype (const num::Number &__y)
- basic_alltype (const std::vector< bool > &__x)
- basic_alltype (const std::vector< int > &__x)
- basic_alltype (const std::vector< unsigned int > &__x)
- basic_alltype (const std::vector< double > &__x)
- basic_alltype (const std::vector< interval > &__x)
- basic_alltype (const std::vector< std::string > &__x)
- basic_alltype (const std::vector< num::Number > &__x)
- basic_alltype (const std::vector< std::vector< bool > > &__vx)
- basic_alltype (const std::vector< std::vector< int > > &__vx)
- basic_alltype (const std::vector< std::vector< unsigned int > > &__vx)
- basic_alltype (const std::vector< std::vector< double > > &__vx)
- basic_alltype (const std::vector< std::vector< interval > > &__vx)
- basic_alltype (const std::vector< std::vector< std::string > > &__vx)
- basic_alltype (const std::vector< std::vector< num::Number > > &__vx)
- basic_alltype (const vmtl::sparse_vector< bool > &__x)
- basic_alltype (const vmtl::sparse_vector< int > &__x)
- basic_alltype (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype (const vmtl::sparse_vector< double > &__x)
- basic_alltype (const vmtl::sparse_vector< interval > &__x)
- basic_alltype (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype (const vmtl::dense_matrix< double > &__x)
- basic_alltype (const vmtl::dense_matrix< int > &__x)
- basic_alltype (const vmtl::dense_matrix< interval > &__x)
- basic_alltype (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype (const vmtl::sparse_matrix< double > &__x)
- basic_alltype (const vmtl::sparse_matrix< int > &__x)
- basic_alltype (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype (const vmtl::sparse_matrix< num::Number > &__x)
- ~basic_alltype ()
- basic_alltype (const basic_alltype &__a)
- basic_alltype & operator= (bool __x)
- basic_alltype & operator= (int __x)
- basic_alltype & operator= (unsigned int __x)
- basic_alltype & operator= (double __x)
- basic_alltype & operator= (interval __x)
- basic_alltype & operator= (void *__x)
- basic_alltype & operator= (const std::string &__x)
- basic_alltype & operator= (const char *__x)
- basic_alltype & operator= (const num::Number &__x)
- basic_alltype & operator= (const std::vector< bool > &__x)
- basic_alltype & operator= (const std::vector< int > &__x)
- basic_alltype & operator= (const std::vector< unsigned int > &__x)

- basic_alltype & operator= (const std::vector< double > &__x)
- basic_alltype & operator= (const std::vector< interval > &__x)
- basic_alltype & operator= (const std::vector< std::string > &__x)
- basic_alltype & operator= (const std::vector< num::Number > &__x)
- basic_alltype & operator= (const std::vector< std::vector< bool > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< unsigned int > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< double > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< interval > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< std::string > > &__x)
- basic_alltype & operator= (const std::vector< std::vector< num::Number > > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< bool > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< unsigned int > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_vector< num::Number > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::dense_matrix< num::Number > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< double > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< int > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< interval > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< std::string > &__x)
- basic_alltype & operator= (const vmtl::sparse_matrix< num::Number > &__x)
- basic_alltype & operator= (const basic_alltype &__a)
- basic_alltype & clear ()
- basic_alltype & set_dm (vmtl::dense_matrix< double > *__m)
- basic_alltype & set_ndm (vmtl::dense_matrix< int > *__m)
- basic_alltype & set_idm (vmtl::dense_matrix< interval > *__m)
- basic_alltype & set_cdm (vmtl::dense_matrix< std::string > *__m)
- basic_alltype & set_xdm (vmtl::dense_matrix< num::Number > *__m)
- basic_alltype & set_sm (vmtl::sparse_matrix< double > *__m)
- basic_alltype & set_nsm (vmtl::sparse_matrix< int > *__m)
- basic_alltype & set_ism (vmtl::sparse_matrix< interval > *__m)
- basic_alltype & set_csm (vmtl::sparse_matrix< std::string > *__m)
- basic_alltype & set_xsm (vmtl::sparse_matrix< num::Number > *__m)
- bool nb () const BASIC_ALLTYPE_THROW
- int nn () const BASIC_ALLTYPE_THROW
- unsigned int nu () const BASIC_ALLTYPE_THROW
- double nd () const BASIC_ALLTYPE_THROW
- interval ni () const BASIC_ALLTYPE_THROW
- const void * p () const BASIC_ALLTYPE_THROW
- const std::string & s () const BASIC_ALLTYPE_THROW
- const num::Number & y () const BASIC_ALLTYPE_THROW
- const std::vector< bool > & b () const BASIC_ALLTYPE_THROW
- const std::vector< int > & n () const BASIC_ALLTYPE_THROW

- const std::vector< unsigned int > & u () const BASIC_ALLTYPE_THROW
- const std::vector< double > & d () const BASIC_ALLTYPE_THROW
- const std::vector< interval > & i () const BASIC_ALLTYPE_THROW
- const std::vector< std::string > & c () const BASIC_ALLTYPE_THROW
- const std::vector< num::Number > & x () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < bool > > & vb () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < int > > & vn () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < unsigned int > > & vu () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < double > > & vd () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < interval > > & vi () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < std::string > > & vc () const BASIC_ALLTYPE_THROW
- const std::vector< std::vector < num::Number > > & vx () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< bool > & sb () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector< int > & sn () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < unsigned int > & su () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < double > & sd () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < interval > & si () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < std::string > & sc () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_vector < num::Number > & sx () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < double > & dm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix< int > & ndm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < interval > & idm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < std::string > & cdm () const BASIC_ALLTYPE_THROW
- const vmtl::dense_matrix < num::Number > & xdm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < double > & sm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix< int > & nsm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < interval > & ism () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < std::string > & csm () const BASIC_ALLTYPE_THROW
- const vmtl::sparse_matrix < num::Number > & xsm () const BASIC_ALLTYPE_THROW
- bool & nb () BASIC_ALLTYPE_THROW
- int & nn () BASIC_ALLTYPE_THROW
- unsigned int & nu () BASIC_ALLTYPE_THROW
- double & nd () BASIC_ALLTYPE_THROW
- interval_st & ni () BASIC_ALLTYPE_THROW
- void *& p () BASIC_ALLTYPE_THROW
- std::string & s () BASIC_ALLTYPE_THROW
- num::Number & y () BASIC_ALLTYPE_THROW
- std::vector< bool > & b () BASIC_ALLTYPE_THROW
- std::vector< int > & n () BASIC_ALLTYPE_THROW
- std::vector< unsigned int > & u () BASIC_ALLTYPE_THROW
- std::vector< double > & d () BASIC_ALLTYPE_THROW
- std::vector< interval > & i () BASIC_ALLTYPE_THROW
- std::vector< std::string > & c () BASIC_ALLTYPE_THROW
- std::vector< num::Number > & x () BASIC_ALLTYPE_THROW
- std::vector< std::vector< bool > > & vb () BASIC_ALLTYPE_THROW
- std::vector< std::vector< int > > & vn () BASIC_ALLTYPE_THROW
- std::vector< std::vector < unsigned int > > & vu () BASIC_ALLTYPE_THROW
- std::vector< std::vector < double > > & vd () BASIC_ALLTYPE_THROW
- std::vector< std::vector < interval > > & vi () BASIC_ALLTYPE_THROW
- std::vector< std::vector < std::string > > & vc () BASIC_ALLTYPE_THROW

- std::vector< std::vector < num::Number > > & vx () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< bool > & sb () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< int > & sn () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< unsigned int > & su () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< double > & sd () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector< interval > & si () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < std::string > & sc () BASIC_ALLTYPE_THROW
- vmtl::sparse_vector < num::Number > & sx () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< double > & dm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< int > & ndm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< interval > & idm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< std::string > & cdm () BASIC_ALLTYPE_THROW
- vmtl::dense_matrix< num::Number > & xdm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< double > & sm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< int > & nsm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix< interval > & ism () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < std::string > & csm () BASIC_ALLTYPE_THROW
- vmtl::sparse_matrix < num::Number > & xsm () BASIC_ALLTYPE_THROW
- bool is_allocated () const
- bool is_scalar () const
- bool is_vector () const
- bool is_vector_of_vector () const
- bool is_dense_vector () const
- bool is_sparse_vector () const
- bool is_empty_vector () const
- bool is_matrix () const
- bool is_dense_matrix () const
- bool is_sparse_matrix () const
- bool is_empty_matrix () const
- bool empty () const
- int contents_type () const
- std::string type_name () const
- const char * type_cstr () const
- bool operator== (const basic_alltype &b) const
- bool operator!= (const basic_alltype &b) const
- bool less_than (const basic_alltype &b) const
- bool less_equal (const basic_alltype &b) const

10.24.1 Detailed Description

This class is the base of the data handling in the API. It can hold values of different types: bool, int, unsigned int, double, interval, string, Number, vector<bool>, vector<int>, vector<unsigned int>, vector<double>, vector<interval>, vector<string>, vector<Number>, sparse_vector<bool>, sparse_vector<int>, sparse_vector<unsigned int>, sparse_vector<double>, sparse_vector<interval>, sparse_vector<string>, sparse_vector<Number>, dense_matrix<double>, dense_matrix<int>, dense_matrix<interval>, dense_matrix<string>, dense_matrix<Number>, sparse_matrix<double>, sparse_matrix<int>, sparse_matrix<interval>, sparse_matrix<string>, sparse_matrix<Number>. All of these types can be stored and retrieved conveniently.

Definition at line 175 of file search_graph.cc.

10.24.2 Constructor & Destructor Documentation

10.24.2.1 coco::coco::basic_alltype::basic_alltype ( ) [inline]

Standard Constructor yielding an empty basic_alltype

Definition at line 480 of file expression.h.

10.24.2.2 coco::coco::basic_alltype::basic_alltype (bool __x) [inline]

Constructor storing the bool ___x

Definition at line 482 of file expression.h.

10.24.2.3 coco::coco::basic_alltype::basic_alltype (int __x) [inline]

Constructor storing the int ___x

Definition at line 485 of file expression.h.

10.24.2.4 coco::coco::basic_alltype::basic_alltype (unsigned int __x) [inline]

Constructor storing the unsigned int ___x

Definition at line 488 of file expression.h.

10.24.2.5 coco::coco::basic_alltype::basic_alltype (double __x) [inline]

Constructor storing the double ___x

Definition at line 491 of file expression.h.

10.24.2.6 coco::coco::basic_alltype::basic_alltype (interval __x) [inline]

Constructor storing the interval ___x

Definition at line 494 of file expression.h.

10.24.2.7 coco::coco::basic_alltype::basic_alltype (void * __x) [inline]

Constructor storing the void* ___x

Definition at line 497 of file expression.h.

10.24.2.8 coco::coco::basic_alltype::basic_alltype (const char * __cp) [inline]

Constructor storing the string ___cp

Definition at line 500 of file expression.h.

10.24.2.9 coco::coco::basic_alltype::basic_alltype (const std::string & __x) [inline]

Constructor storing the string ___x

Definition at line 503 of file expression.h.

10.24.2.10 coco::coco::basic_alltype::basic_alltype (const num::Number & __y) [inline]

Constructor storing the number __y

Definition at line 506 of file expression.h.

10.24.2.11 coco::coco::basic_alltype::basic_alltype (const std::vector< bool > & __x) [inline]

Constructor storing the vector<bool> __x

Definition at line 510 of file expression.h.

10.24.2.12 coco::coco::basic_alltype::basic_alltype (const std::vector< int > & __x) [inline]

Constructor storing the vector<int> __x

Definition at line 514 of file expression.h.

10.24.2.13 coco::coco::basic_alltype::basic_alltype (const std::vector< unsigned int > & __x) [inline]

Constructor storing the vector<unsigned int> __x

Definition at line 518 of file expression.h.

10.24.2.14 coco::coco::basic_alltype::basic_alltype (const std::vector< double > & __x) [inline]

Constructor storing the vector<double> __x

Definition at line 522 of file expression.h.

10.24.2.15 coco::coco::basic_alltype::basic_alltype (const std::vector< interval > & __x) [inline]

Constructor storing the vector<interval> __x

Definition at line 526 of file expression.h.

10.24.2.16 coco::coco::basic_alltype::basic_alltype (const std::vector< std::string > & __x) [inline]

Constructor storing the vector<string> __x

Definition at line 530 of file expression.h.

10.24.2.17 coco::coco::basic_alltype::basic_alltype (const std::vector< num::Number > & __x) [inline]

Constructor storing the vector<Number> __x

Definition at line 534 of file expression.h.

10.24.2.18 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< bool > > & __vx) [inline]

Constructor storing the vector<vector<bool> > __vx

Definition at line 539 of file expression.h.

10.24.2.19 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< int > > & __vx) [inline]

Constructor storing the vector<vector<int> > __vx

Definition at line 543 of file expression.h.

10.24.2.20 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< unsigned int > > & __vx) [inline]

Constructor storing the vector<vector<unsigned int> > __vx

Definition at line 547 of file expression.h.

10.24.2.21 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< double > > & __vx) [inline]

Constructor storing the vector<vector<double> > __vx

Definition at line 551 of file expression.h.

10.24.2.22 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< interval > > & __vx) [inline]

Constructor storing the vector<vector<interval> > __vx

Definition at line 555 of file expression.h.

10.24.2.23 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< std::string > > & __vx) [inline]

Constructor storing the vector<vector<string> > __vx

Definition at line 559 of file expression.h.

10.24.2.24 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< num::Number > > & __vx) [inline]

Constructor storing the vector<vector<Number> > __vx

Definition at line 563 of file expression.h.

10.24.2.25 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< bool > & __x) [inline]

Constructor storing the sparse_vector<bool> __x

Definition at line 568 of file expression.h.

10.24.2.26 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< int > & __x) [inline]

Constructor storing the sparse_vector<int> __x

Definition at line 572 of file expression.h.

10.24.2.27 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Constructor storing the sparse_vector<unsigned int> __x

Definition at line 576 of file expression.h.

10.24.2.28 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< double > & __x) [inline]

Constructor storing the sparse_vector<double> __x

Definition at line 580 of file expression.h.

10.24.2.29 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< interval > & __x) [inline]

Constructor storing the sparse_vector<interval> __x

Definition at line 584 of file expression.h.

10.24.2.30 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< std::string > & __x) [inline]

Constructor storing the sparse_vector<string> __x

Definition at line 588 of file expression.h.

10.24.2.31 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< num::Number > & __x) [inline]

Constructor storing the sparse_vector<Number> __x

Definition at line 592 of file expression.h.

10.24.2.32 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< double > & __x) [inline]

Constructor storing the dense_matrix<double> __x

Definition at line 597 of file expression.h.

10.24.2.33 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< int > & __x) [inline]

Constructor storing the dense_matrix<int> __x

Definition at line 601 of file expression.h.

10.24.2.34 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< interval > & __x) [inline]

Constructor storing the dense_matrix<interval> __x

Definition at line 605 of file expression.h.

10.24.2.35 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< std::string > & __x) [inline]

Constructor storing the dense_matrix<string> __x

Definition at line 609 of file expression.h.

10.24.2.36 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< num::Number > & __x) [inline]

Constructor storing the dense_matrix<Number> __x

Definition at line 613 of file expression.h.

10.24.2.37 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< double > & __x) [inline]

Constructor storing the sparse_matrix<double> __x

Definition at line 618 of file expression.h.

10.24.2.38 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< int > & __x) [inline]

Constructor storing the sparse_matrix<int> __x

Definition at line 622 of file expression.h.

10.24.2.39 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< interval > & __x) [inline]

Constructor storing the sparse_matrix<interval> __x

Definition at line 626 of file expression.h.

10.24.2.40 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< std::string > & __x) [inline]

Constructor storing the sparse_matrix<string> __x

Definition at line 630 of file expression.h.

10.24.2.41 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Constructor storing the sparse_matrix<Number> __x

Definition at line 634 of file expression.h.

10.24.2.42 coco::coco::basic_alltype::~basic_alltype ( ) [inline]

Standard Destructor

Definition at line 639 of file expression.h.

10.24.2.43 coco::coco::basic_alltype::basic_alltype (const basic_alltype & __a) [inline]

Standard Copy Constructor

Definition at line 643 of file expression.h.

10.24.2.44 coco::coco::basic_alltype::basic_alltype ( ) [inline]

Standard Constructor yielding an empty basic_alltype

Definition at line 480 of file search_graph.cc.

10.24.2.45 coco::coco::basic_alltype::basic_alltype (bool __x) [inline]

Constructor storing the bool ___x

Definition at line 482 of file search_graph.cc.

10.24.2.46 coco::coco::basic_alltype::basic_alltype (int __x) [inline]

Constructor storing the int ___x

Definition at line 485 of file search_graph.cc.

10.24.2.47 coco::coco::basic_alltype::basic_alltype (unsigned int __x) [inline]

Constructor storing the unsigned int ___x

Definition at line 488 of file search_graph.cc.

10.24.2.48 coco::coco::basic_alltype::basic_alltype (double __x) [inline]

Constructor storing the double ___x

Definition at line 491 of file search_graph.cc.

10.24.2.49 coco::coco::basic_alltype::basic_alltype (interval __x) [inline]

Constructor storing the interval ___x

Definition at line 494 of file search_graph.cc.

10.24.2.50 coco::coco::basic_alltype::basic_alltype (void * __x) [inline]

Constructor storing the void* ___x

Definition at line 497 of file search_graph.cc.

10.24.2.51 coco::coco::basic_alltype::basic_alltype (const char * __cp) [inline]

Constructor storing the string ___cp

Definition at line 500 of file search_graph.cc.

10.24.2.52 coco::coco::basic_alltype::basic_alltype (const std::string & __x) [inline]

Constructor storing the string ___x

Definition at line 503 of file search_graph.cc.

10.24.2.53 coco::coco::basic_alltype::basic_alltype (const num::Number & __y) [inline]

Constructor storing the number ___y

Definition at line 506 of file search_graph.cc.

10.24.2.54 coco::coco::basic_alltype::basic_alltype (const std::vector< bool > & __x) [inline]

Constructor storing the vector<bool> ___x

Definition at line 510 of file search_graph.cc.

10.24.2.55 coco::coco::basic_alltype::basic_alltype (const std::vector< int > & __x) [inline]

Constructor storing the vector<int> ___x

Definition at line 514 of file search_graph.cc.

10.24.2.56 coco::coco::basic_alltype::basic_alltype (const std::vector< unsigned int > & __x) [inline]

Constructor storing the vector<unsigned int> ___x

Definition at line 518 of file search_graph.cc.

10.24.2.57 coco::coco::basic_alltype::basic_alltype (const std::vector< double > & __x) [inline]

Constructor storing the vector<double> ___x

Definition at line 522 of file search_graph.cc.

10.24.2.58 coco::coco::basic_alltype::basic_alltype (const std::vector< interval > & __x) [inline]

Constructor storing the vector<interval> ___x

Definition at line 526 of file search_graph.cc.

10.24.2.59 coco::coco::basic_alltype::basic_alltype (const std::vector< std::string > & __x) [inline]

Constructor storing the vector<string> ___x

Definition at line 530 of file search_graph.cc.

10.24.2.60 coco::coco::basic_alltype::basic_alltype (const std::vector< num::Number > & __x) [inline]

Constructor storing the vector<Number> ___x

Definition at line 534 of file search_graph.cc.

10.24.2.61 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< bool > > & __vx) [inline]

Constructor storing the vector<vector<bool> > __vx

Definition at line 539 of file search_graph.cc.

10.24.2.62 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< int > > & __vx) [inline]

Constructor storing the vector<vector<int> > __vx

Definition at line 543 of file search_graph.cc.

10.24.2.63 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< unsigned int > > & __vx) [inline]

Constructor storing the vector<vector<unsigned int> > __vx

Definition at line 547 of file search_graph.cc.

10.24.2.64 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< double > > & __vx) [inline]

Constructor storing the vector<vector<double> > __vx

Definition at line 551 of file search_graph.cc.

10.24.2.65 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< interval > > & __vx) [inline]

Constructor storing the vector<vector<interval> > __vx

Definition at line 555 of file search_graph.cc.

10.24.2.66 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< std::string > > & __vx) [inline]

Constructor storing the vector<vector<string> > __vx

Definition at line 559 of file search_graph.cc.

10.24.2.67 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< num::Number > > & __vx) [inline]

Constructor storing the vector<vector<Number> > __vx

Definition at line 563 of file search_graph.cc.

10.24.2.68 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< bool > & __x) [inline]

Constructor storing the sparse_vector<bool> __x

Definition at line 568 of file search_graph.cc.

10.24.2.69 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< int > & __x) [inline]

Constructor storing the sparse_vector<int> __x

Definition at line 572 of file search_graph.cc.

10.24.2.70 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Constructor storing the sparse_vector<unsigned int> __x

Definition at line 576 of file search_graph.cc.

10.24.2.71 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< double > & __x) [inline]

Constructor storing the sparse_vector<double> __x

Definition at line 580 of file search_graph.cc.

10.24.2.72 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< interval > & __x) [inline]

Constructor storing the sparse_vector<interval> __x

Definition at line 584 of file search_graph.cc.

10.24.2.73 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< std::string > & __x) [inline]

Constructor storing the sparse_vector<string> __x

Definition at line 588 of file search_graph.cc.

10.24.2.74 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< num::Number > & __x) [inline]

Constructor storing the sparse_vector<Number> __x

Definition at line 592 of file search_graph.cc.

10.24.2.75 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< double > & __x) [inline]

Constructor storing the dense_matrix<double> __x

Definition at line 597 of file search_graph.cc.

10.24.2.76 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< int > & __x) [inline]

Constructor storing the dense_matrix<int> __x

Definition at line 601 of file search_graph.cc.

10.24.2.77 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< interval > & __x) [inline]

Constructor storing the dense_matrix<interval> __x

Definition at line 605 of file search_graph.cc.

10.24.2.78 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< std::string > & __x) [inline]

Constructor storing the dense_matrix<string> __x

Definition at line 609 of file search_graph.cc.

10.24.2.79 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< num::Number > & __x) [inline]

Constructor storing the dense_matrix<Number> __x

Definition at line 613 of file search_graph.cc.

10.24.2.80 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< double > & __x) [inline]

Constructor storing the sparse_matrix<double> __x

Definition at line 618 of file search_graph.cc.

10.24.2.81 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< int > & __x) [inline]

Constructor storing the sparse_matrix<int> __x

Definition at line 622 of file search_graph.cc.

10.24.2.82 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< interval > & __x) [inline]

Constructor storing the sparse_matrix<interval> __x

Definition at line 626 of file search_graph.cc.

10.24.2.83 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< std::string > & __x) [inline]

Constructor storing the sparse_matrix<string> __x

Definition at line 630 of file search_graph.cc.

10.24.2.84 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Constructor storing the sparse_matrix<Number> __x

Definition at line 634 of file search_graph.cc.

10.24.2.85 coco::coco::basic_alltype::~basic_alltype ( ) [inline]

Standard Destructor

Definition at line 639 of file search_graph.cc.

10.24.2.86 coco::coco::basic_alltype::basic_alltype (const basic_alltype & __a) [inline]

Standard Copy Constructor

Definition at line 643 of file search_graph.cc.

10.24.2.87 coco::coco::basic_alltype::basic_alltype ( ) [inline]

Standard Constructor yielding an empty basic_alltype

Definition at line 480 of file search_graph.cc.

10.24.2.88 coco::coco::basic_alltype::basic_alltype (bool __x) [inline]

Constructor storing the bool ___x

Definition at line 482 of file search_graph.cc.

10.24.2.89 coco::coco::basic_alltype::basic_alltype (int __x) [inline]

Constructor storing the int ___x

Definition at line 485 of file search_graph.cc.

10.24.2.90 coco::coco::basic_alltype::basic_alltype (unsigned int __x) [inline]

Constructor storing the unsigned int ___x

Definition at line 488 of file search_graph.cc.

10.24.2.91 coco::coco::basic_alltype::basic_alltype (double __x) [inline]

Constructor storing the double ___x

Definition at line 491 of file search_graph.cc.

10.24.2.92 coco::coco::basic_alltype::basic_alltype (interval __x) [inline]

Constructor storing the interval ___x

Definition at line 494 of file search_graph.cc.

10.24.2.93 coco::coco::basic_alltype::basic_alltype (void * __x) [inline]

Constructor storing the void* ___x

Definition at line 497 of file search_graph.cc.

10.24.2.94 coco::coco::basic_alltype::basic_alltype (const char * __cp) [inline]

Constructor storing the string ___cp

Definition at line 500 of file search_graph.cc.

10.24.2.95 coco::coco::basic_alltype::basic_alltype (const std::string & __x) [inline]

Constructor storing the string ___x

Definition at line 503 of file search_graph.cc.

10.24.2.96 coco::coco::basic_alltype::basic_alltype (const num::Number & __y) [inline]

Constructor storing the number ___y

Definition at line 506 of file search_graph.cc.

10.24.2.97 coco::coco::basic_alltype::basic_alltype (const std::vector< bool > & __x) [inline]

Constructor storing the vector<bool> ___x

Definition at line 510 of file search_graph.cc.

10.24.2.98 coco::coco::basic_alltype::basic_alltype (const std::vector< int > & __x) [inline]

Constructor storing the vector<int> ___x

Definition at line 514 of file search_graph.cc.

10.24.2.99 coco::coco::basic_alltype::basic_alltype (const std::vector< unsigned int > & __x) [inline]

Constructor storing the vector<unsigned int> ___x

Definition at line 518 of file search_graph.cc.

10.24.2.100 coco::coco::basic_alltype::basic_alltype (const std::vector< double > & __x) [inline]

Constructor storing the vector<double> ___x

Definition at line 522 of file search_graph.cc.

10.24.2.101 coco::coco::basic_alltype::basic_alltype (const std::vector< interval > & __x) [inline]

Constructor storing the vector<interval> ___x

Definition at line 526 of file search_graph.cc.

10.24.2.102 coco::coco::basic_alltype::basic_alltype (const std::vector< std::string > & __x) [inline]

Constructor storing the vector<string> ___x

Definition at line 530 of file search_graph.cc.

10.24.2.103 coco::coco::basic_alltype::basic_alltype (const std::vector< num::Number > & __x) [inline]

Constructor storing the vector<Number> __x

Definition at line 534 of file search_graph.cc.

10.24.2.104 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< bool > > & __vx) [inline]

Constructor storing the vector<vector<bool> > __vx

Definition at line 539 of file search_graph.cc.

10.24.2.105 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< int > > & __vx) [inline]

Constructor storing the vector<vector<int> > __vx

Definition at line 543 of file search_graph.cc.

10.24.2.106 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< unsigned int > > & __vx) [inline]

Constructor storing the vector<vector<unsigned int> > __vx

Definition at line 547 of file search_graph.cc.

10.24.2.107 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< double > > & __vx) [inline]

Constructor storing the vector<vector<double> > __vx

Definition at line 551 of file search_graph.cc.

10.24.2.108 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< interval > > & __vx) [inline]

Constructor storing the vector<vector<interval> > __vx

Definition at line 555 of file search_graph.cc.

10.24.2.109 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< std::string > > & __vx) [inline]

Constructor storing the vector<vector<string> > __vx

Definition at line 559 of file search_graph.cc.

10.24.2.110 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< num::Number > > & __vx) [inline]

Constructor storing the vector<vector<Number> > __vx

Definition at line 563 of file search_graph.cc.

10.24.2.111 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< bool > & __x) [inline]

Constructor storing the sparse_vector<bool> __x

Definition at line 568 of file search_graph.cc.

10.24.2.112 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< int > & __x) [inline]

Constructor storing the sparse_vector<int> __x

Definition at line 572 of file search_graph.cc.

10.24.2.113 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Constructor storing the sparse_vector<unsigned int> __x

Definition at line 576 of file search_graph.cc.

10.24.2.114 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< double > & __x) [inline]

Constructor storing the sparse_vector<double> __x

Definition at line 580 of file search_graph.cc.

10.24.2.115 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< interval > & __x) [inline]

Constructor storing the sparse_vector<interval> __x

Definition at line 584 of file search_graph.cc.

10.24.2.116 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< std::string > & __x) [inline]

Constructor storing the sparse_vector<string> __x

Definition at line 588 of file search_graph.cc.

10.24.2.117 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< num::Number > & __x) [inline]

Constructor storing the sparse_vector<Number> __x

Definition at line 592 of file search_graph.cc.

10.24.2.118 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< double > & __x) [inline]

Constructor storing the dense_matrix<double> __x

Definition at line 597 of file search_graph.cc.

10.24.2.119 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< int > & __x) [inline]

Constructor storing the dense_matrix<int> __x

Definition at line 601 of file search_graph.cc.

10.24.2.120 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< interval > & __x) [inline]

Constructor storing the dense_matrix<interval> __x

Definition at line 605 of file search_graph.cc.

10.24.2.121 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< std::string > & __x) [inline]

Constructor storing the dense_matrix<string> __x

Definition at line 609 of file search_graph.cc.

10.24.2.122 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< num::Number > & __x) [inline]

Constructor storing the dense_matrix<Number> __x

Definition at line 613 of file search_graph.cc.

10.24.2.123 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< double > & __x) [inline]

Constructor storing the sparse_matrix<double> __x

Definition at line 618 of file search_graph.cc.

10.24.2.124 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< int > & __x) [inline]

Constructor storing the sparse_matrix<int> __x

Definition at line 622 of file search_graph.cc.

10.24.2.125 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< interval > & __x) [inline]

Constructor storing the sparse_matrix<interval> __x

Definition at line 626 of file search_graph.cc.

10.24.2.126 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< std::string > & __x) [inline]

Constructor storing the sparse_matrix<string> __x

Definition at line 630 of file search_graph.cc.

10.24.2.127 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Constructor storing the sparse_matrix<Number> __x

Definition at line 634 of file search_graph.cc.

10.24.2.128 coco::coco::basic_alltype::~basic_alltype ( ) [inline]

Standard Destructor

Definition at line 639 of file search_graph.cc.

10.24.2.129 coco::coco::basic_alltype::basic_alltype (const basic_alltype & __a) [inline]

Standard Copy Constructor

Definition at line 643 of file search_graph.cc.

10.24.2.130 coco::coco::basic_alltype::basic_alltype ( ) [inline]

Standard Constructor yielding an empty basic_alltype

Definition at line 480 of file search_graph.cc.

10.24.2.131 coco::coco::basic_alltype::basic_alltype (bool __x) [inline]

Constructor storing the bool __x

Definition at line 482 of file search_graph.cc.

10.24.2.132 coco::coco::basic_alltype::basic_alltype (int __x) [inline]

Constructor storing the int __x

Definition at line 485 of file search_graph.cc.

10.24.2.133 coco::coco::basic_alltype::basic_alltype (unsigned int __x) [inline]

Constructor storing the unsigned int __x

Definition at line 488 of file search_graph.cc.

10.24.2.134 coco::coco::basic_alltype::basic_alltype (double __x) [inline]

Constructor storing the double __x

Definition at line 491 of file search_graph.cc.

10.24.2.135 coco::coco::basic_alltype::basic_alltype (interval __x) [inline]

Constructor storing the interval __x

Definition at line 494 of file search_graph.cc.

10.24.2.136 coco::coco::basic_alltype::basic_alltype (void * __x) [inline]

Constructor storing the void* ___x

Definition at line 497 of file search_graph.cc.

10.24.2.137 coco::coco::basic_alltype::basic_alltype (const char * __cp) [inline]

Constructor storing the string ___cp

Definition at line 500 of file search_graph.cc.

10.24.2.138 coco::coco::basic_alltype::basic_alltype (const std::string & __x) [inline]

Constructor storing the string ___x

Definition at line 503 of file search_graph.cc.

10.24.2.139 coco::coco::basic_alltype::basic_alltype (const num::Number & __y) [inline]

Constructor storing the number ___y

Definition at line 506 of file search_graph.cc.

10.24.2.140 coco::coco::basic_alltype::basic_alltype (const std::vector< bool > & __x) [inline]

Constructor storing the vector<bool> ___x

Definition at line 510 of file search_graph.cc.

10.24.2.141 coco::coco::basic_alltype::basic_alltype (const std::vector< int > & __x) [inline]

Constructor storing the vector<int> ___x

Definition at line 514 of file search_graph.cc.

10.24.2.142 coco::coco::basic_alltype::basic_alltype (const std::vector< unsigned int > & __x) [inline]

Constructor storing the vector<unsigned int> ___x

Definition at line 518 of file search_graph.cc.

10.24.2.143 coco::coco::basic_alltype::basic_alltype (const std::vector< double > & __x) [inline]

Constructor storing the vector<double> ___x

Definition at line 522 of file search_graph.cc.

10.24.2.144 coco::coco::basic_alltype::basic_alltype (const std::vector< interval > & __x) [inline]

Constructor storing the vector<interval> ___x

Definition at line 526 of file search_graph.cc.

10.24.2.145 coco::coco::basic_alltype::basic_alltype (const std::vector< std::string > & __x) [inline]

Constructor storing the vector<string> __x

Definition at line 530 of file search_graph.cc.

10.24.2.146 coco::coco::basic_alltype::basic_alltype (const std::vector< num::Number > & __x) [inline]

Constructor storing the vector<Number> __x

Definition at line 534 of file search_graph.cc.

10.24.2.147 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< bool > > & __vx) [inline]

Constructor storing the vector<vector<bool> > __vx

Definition at line 539 of file search_graph.cc.

10.24.2.148 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< int > > & __vx) [inline]

Constructor storing the vector<vector<int> > __vx

Definition at line 543 of file search_graph.cc.

10.24.2.149 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< unsigned int > > & __vx) [inline]

Constructor storing the vector<vector<unsigned int> > __vx

Definition at line 547 of file search_graph.cc.

10.24.2.150 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< double > > & __vx) [inline]

Constructor storing the vector<vector<double> > __vx

Definition at line 551 of file search_graph.cc.

10.24.2.151 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< interval > > & __vx) [inline]

Constructor storing the vector<vector<interval> > __vx

Definition at line 555 of file search_graph.cc.

10.24.2.152 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< std::string > > & __vx) [inline]

Constructor storing the vector<vector<string> > __vx

Definition at line 559 of file search_graph.cc.

10.24.2.153 coco::coco::basic_alltype::basic_alltype (const std::vector< std::vector< num::Number > > & __vx) [inline]

Constructor storing the vector<vector<Number> > __vx

Definition at line 563 of file search_graph.cc.

10.24.2.154 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< bool > & __x) [inline]

Constructor storing the sparse_vector<bool> __x

Definition at line 568 of file search_graph.cc.

10.24.2.155 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< int > & __x) [inline]

Constructor storing the sparse_vector<int> __x

Definition at line 572 of file search_graph.cc.

10.24.2.156 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Constructor storing the sparse_vector<unsigned int> __x

Definition at line 576 of file search_graph.cc.

10.24.2.157 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< double > & __x) [inline]

Constructor storing the sparse_vector<double> __x

Definition at line 580 of file search_graph.cc.

10.24.2.158 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< interval > & __x) [inline]

Constructor storing the sparse_vector<interval> __x

Definition at line 584 of file search_graph.cc.

10.24.2.159 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< std::string > & __x) [inline]

Constructor storing the sparse_vector<string> __x

Definition at line 588 of file search_graph.cc.

10.24.2.160 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_vector< num::Number > & __x) [inline]

Constructor storing the sparse_vector<Number> __x

Definition at line 592 of file search_graph.cc.

10.24.2.161 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< double > & __x) [inline]

Constructor storing the dense_matrix<double> __x

Definition at line 597 of file search_graph.cc.

10.24.2.162 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< int > & __x) [inline]

Constructor storing the dense_matrix<int> __x

Definition at line 601 of file search_graph.cc.

10.24.2.163 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< interval > & __x) [inline]

Constructor storing the dense_matrix<interval> __x

Definition at line 605 of file search_graph.cc.

10.24.2.164 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< std::string > & __x) [inline]

Constructor storing the dense_matrix<string> __x

Definition at line 609 of file search_graph.cc.

10.24.2.165 coco::coco::basic_alltype::basic_alltype (const vmtl::dense_matrix< num::Number > & __x) [inline]

Constructor storing the dense_matrix<Number> __x

Definition at line 613 of file search_graph.cc.

10.24.2.166 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< double > & __x) [inline]

Constructor storing the sparse_matrix<double> __x

Definition at line 618 of file search_graph.cc.

10.24.2.167 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< int > & __x) [inline]

Constructor storing the sparse_matrix<int> __x

Definition at line 622 of file search_graph.cc.

10.24.2.168 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< interval > & __x) [inline]

Constructor storing the sparse_matrix<interval> __x

Definition at line 626 of file search_graph.cc.

10.24.2.169 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< std::string > & __x) [inline]

Constructor storing the sparse_matrix<string> __x

Definition at line 630 of file search_graph.cc.

10.24.2.170 coco::coco::basic_alltype::basic_alltype (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Constructor storing the sparse_matrix<Number> __x

Definition at line 634 of file search_graph.cc.

10.24.2.171 coco::coco::basic_alltype::~basic_alltype ( ) [inline]

Standard Destructor

Definition at line 639 of file search_graph.cc.

10.24.2.172 coco::coco::basic_alltype::basic_alltype (const basic_alltype & __a) [inline]

Standard Copy Constructor

Definition at line 643 of file search_graph.cc.

10.24.3 Member Function Documentation

10.24.3.1 const std::vector<bool>& coco::coco::basic_alltype::b ( ) const [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1085 of file expression.h.

10.24.3.2 const std::vector<bool>& coco::coco::basic_alltype::b ( ) const [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1085 of file search_graph.cc.

10.24.3.3 const std::vector<bool>& coco::coco::basic_alltype::b ( ) const [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1085 of file search_graph.cc.

10.24.3.4 const std::vector<bool>& coco::coco::basic_alltype::b ( ) const [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1085 of file search_graph.cc.

10.24.3.5 std::vector<bool>& coco::coco::basic_alltype::b ( ) [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1357 of file search_graph.cc.

10.24.3.6 std::vector<bool>& coco::coco::basic_alltype::b ( ) [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1357 of file expression.h.

10.24.3.7 std::vector<bool>& coco::coco::basic_alltype::b ( ) [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1357 of file search_graph.cc.

10.24.3.8 std::vector<bool>& coco::coco::basic_alltype::b ( ) [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 1357 of file search_graph.cc.

10.24.3.9 const std::vector<std::string>& coco::coco::basic_alltype::c ( ) const [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1141 of file search_graph.cc.

10.24.3.10 const std::vector<std::string>& coco::coco::basic_alltype::c ( ) const [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1141 of file expression.h.

10.24.3.11 const std::vector<std::string>& coco::coco::basic_alltype::c ( ) const [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1141 of file search_graph.cc.

10.24.3.12 const std::vector<std::string>& coco::coco::basic_alltype::c ( ) const [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1141 of file search_graph.cc.

10.24.3.13 std::vector<std::string>& coco::coco::basic_alltype::c ( ) [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1413 of file search_graph.cc.

10.24.3.14 std::vector<std::string>& coco::coco::basic_alltype::c ( ) [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1413 of file search_graph.cc.

10.24.3.15 std::vector<std::string>& coco::coco::basic_alltype::c ( ) [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1413 of file expression.h.

10.24.3.16 std::vector<std::string>& coco::coco::basic_alltype::c ( ) [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 1413 of file search_graph.cc.

10.24.3.17 const vmtl::dense_matrix<std::string>& coco::coco::basic_alltype::cdm ( ) const [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1298 of file search_graph.cc.

10.24.3.18 const vmtl::dense_matrix<std::string>& coco::coco::basic_alltype::cdm ( ) const [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1298 of file expression.h.

10.24.3.19 const vmtl::dense_matrix<std::string>& coco::coco::basic_alltype::cdm ( ) const [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1298 of file search_graph.cc.

10.24.3.20 const vmtl::dense_matrix<std::string>& coco::coco::basic_alltype::cdm ( ) const [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1298 of file search_graph.cc.

10.24.3.21 vmtl::dense_matrix<std::string>& coco::coco::basic_alltype::cdm ( ) [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1569 of file search_graph.cc.

10.24.3.22 vmtl::dense_matrix<std::string>& coco::coco::basic_alltype::cdm ( ) [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1569 of file search_graph.cc.

10.24.3.23 vmtl::dense_matrix<std::string>& coco::coco::basic_alltype::cdm ( ) [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1569 of file search_graph.cc.

10.24.3.24 vmtl::dense_matrix<std::string>& coco::coco::basic_alltype::cdm ( ) [inline]

retrieve a dense_matrix<string> from the basic_alltype

Definition at line 1569 of file expression.h.

10.24.3.25 basic_alltype& coco::coco::basic_alltype::clear ( ) [inline]

Reset the basic_alltype to an empty value

Definition at line 929 of file search_graph.cc.

10.24.3.26 basic_alltype& coco::coco::basic_alltype::clear ( ) [inline]

Reset the basic_alltype to an empty value

Definition at line 929 of file search_graph.cc.

10.24.3.27 basic_alltype& coco::coco::basic_alltype::clear ( ) [inline]

Reset the basic_alltype to an empty value

Definition at line 929 of file expression.h.

10.24.3.28 basic_alltype& coco::coco::basic_alltype::clear ( ) [inline]

Reset the basic_alltype to an empty value

Definition at line 929 of file search_graph.cc.

10.24.3.29 int coco::coco::basic_alltype::contents_type ( ) const [inline]

Return the type of the value contained in the storage, an int which can be compared with the ALLTYPE_... values.

Definition at line 1669 of file expression.h.

10.24.3.30 int coco::coco::basic_alltype::contents_type ( ) const [inline]

Return the type of the value contained in the storage, an int which can be compared with the ALLTYPE_... values.

Definition at line 1669 of file search_graph.cc.

10.24.3.31 int coco::coco::basic_alltype::contents_type ( ) const [inline]

Return the type of the value contained in the storage, an int which can be compared with the ALLTYPE_... values.

Definition at line 1669 of file search_graph.cc.

10.24.3.32 int coco::coco::basic_alltype::contents_type ( ) const [inline]

Return the type of the value contained in the storage, an int which can be compared with the ALLTYPE_... values.

Definition at line 1669 of file search_graph.cc.

10.24.3.33 const vmtl::sparse_matrix<std::string>& coco::coco::basic_alltype::csm ( ) const [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1314 of file search_graph.cc.

10.24.3.34 const vmtl::sparse_matrix<std::string>& coco::coco::basic_alltype::csm ( ) const [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1314 of file expression.h.

10.24.3.35 const vmtl::sparse_matrix<std::string>& coco::coco::basic_alltype::csm ( ) const [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1314 of file search_graph.cc.

10.24.3.36 const vmtl::sparse_matrix<std::string>& coco::coco::basic_alltype::csm ( ) const [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1314 of file search_graph.cc.

10.24.3.37 vmtl::sparse_matrix<std::string>& coco::coco::basic_alltype::csm ( ) [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1585 of file search_graph.cc.

10.24.3.38 vmtl::sparse_matrix<std::string>& coco::coco::basic_alltype::csm ( ) [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1585 of file expression.h.

10.24.3.39 vmtl::sparse_matrix<std::string>& coco::coco::basic_alltype::csm ( ) [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1585 of file search_graph.cc.

10.24.3.40 vmtl::sparse_matrix<std::string>& coco::coco::basic_alltype::csm ( ) [inline]

retrieve a sparse_matrix<string> from the basic_alltype

Definition at line 1585 of file search_graph.cc.

10.24.3.41 const std::vector<double>& coco::coco::basic_alltype::d ( ) const [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1133 of file expression.h.

10.24.3.42 const std::vector<double>& coco::coco::basic_alltype::d ( ) const [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1133 of file search_graph.cc.

10.24.3.43 const std::vector<double>& coco::coco::basic_alltype::d ( ) const [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1133 of file search_graph.cc.

10.24.3.44 const std::vector<double>& coco::coco::basic_alltype::d ( ) const [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1133 of file search_graph.cc.

10.24.3.45 std::vector<double>& coco::coco::basic_alltype::d ( ) [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1405 of file search_graph.cc.

10.24.3.46 std::vector<double>& coco::coco::basic_alltype::d ( ) [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1405 of file expression.h.

10.24.3.47 std::vector<double>& coco::coco::basic_alltype::d ( ) [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1405 of file search_graph.cc.

10.24.3.48 std::vector<double>& coco::coco::basic_alltype::d ( ) [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 1405 of file search_graph.cc.

10.24.3.49 const vmtl::dense_matrix<double>& coco::coco::basic_alltype::dm ( ) const [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1289 of file search_graph.cc.

10.24.3.50 const vmtl::dense_matrix<double>& coco::coco::basic_alltype::dm ( ) const [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1289 of file expression.h.

10.24.3.51 const vmtl::dense_matrix<double>& coco::coco::basic_alltype::dm ( ) const [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1289 of file search_graph.cc.

10.24.3.52 const vmtl::dense_matrix<double>& coco::coco::basic_alltype::dm ( ) const [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1289 of file search_graph.cc.

10.24.3.53 vmtl::dense_matrix<double>& coco::coco::basic_alltype::dm ( ) [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1560 of file search_graph.cc.

10.24.3.54 vmtl::dense_matrix<double>& coco::coco::basic_alltype::dm ( ) [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1560 of file search_graph.cc.

10.24.3.55 vmtl::dense_matrix<double>& coco::coco::basic_alltype::dm ( ) [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1560 of file search_graph.cc.

10.24.3.56 vmtl::dense_matrix<double>& coco::coco::basic_alltype::dm ( ) [inline]

retrieve a dense_matrix<double> from the basic_alltype

Definition at line 1560 of file expression.h.

10.24.3.57 bool coco::coco::basic_alltype::empty ( ) const [inline]

Check, whether the basic_alltype is empty.

Definition at line 1666 of file expression.h.

10.24.3.58 bool coco::coco::basic_alltype::empty ( ) const [inline]

Check, whether the basic_alltype is empty.

Definition at line 1666 of file search_graph.cc.

10.24.3.59 bool coco::coco::basic_alltype::empty ( ) const [inline]

Check, whether the basic_alltype is empty.

Definition at line 1666 of file search_graph.cc.

10.24.3.60 bool coco::coco::basic_alltype::empty ( ) const [inline]

Check, whether the basic_alltype is empty.

Definition at line 1666 of file search_graph.cc.

10.24.3.61 const std::vector<interval>& coco::coco::basic_alltype::i ( ) const [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1137 of file search_graph.cc.

10.24.3.62 const std::vector<interval>& coco::coco::basic_alltype::i ( ) const [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1137 of file expression.h.

10.24.3.63 const std::vector<interval>& coco::coco::basic_alltype::i ( ) const [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1137 of file search_graph.cc.

10.24.3.64 const std::vector<interval>& coco::coco::basic_alltype::i ( ) const [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1137 of file search_graph.cc.

10.24.3.65 std::vector<interval>& coco::coco::basic_alltype::i ( ) [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1409 of file search_graph.cc.

10.24.3.66 std::vector<interval>& coco::coco::basic_alltype::i ( ) [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1409 of file search_graph.cc.

10.24.3.67 std::vector<interval>& coco::coco::basic_alltype::i ( ) [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1409 of file expression.h.

10.24.3.68 std::vector<interval>& coco::coco::basic_alltype::i ( ) [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 1409 of file search_graph.cc.

10.24.3.69 const vmtl::dense_matrix<interval>& coco::coco::basic_alltype::idm ( ) const [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1295 of file search_graph.cc.

10.24.3.70 const vmtl::dense_matrix<interval>& coco::coco::basic_alltype::idm ( ) const [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1295 of file expression.h.

10.24.3.71 const vmtl::dense_matrix<interval>& coco::coco::basic_alltype::idm ( ) const [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1295 of file search_graph.cc.

10.24.3.72 const vmtl::dense_matrix<interval>& coco::coco::basic_alltype::idm ( ) const [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1295 of file search_graph.cc.

10.24.3.73 vmtl::dense_matrix<interval>& coco::coco::basic_alltype::idm ( ) [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1566 of file search_graph.cc.

10.24.3.74 vmtl::dense_matrix<interval>& coco::coco::basic_alltype::idm ( ) [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1566 of file search_graph.cc.

10.24.3.75 vmtl::dense_matrix<interval>& coco::coco::basic_alltype::idm ( ) [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1566 of file search_graph.cc.

10.24.3.76 vmtl::dense_matrix<interval>& coco::coco::basic_alltype::idm ( ) [inline]

retrieve a dense_matrix<interval> from the basic_alltype

Definition at line 1566 of file expression.h.

10.24.3.77 bool coco::coco::basic_alltype::is_allocated ( ) const [inline]

Check, whether the basic_alltype contains a complex type, i.e. one which has to be allocated (not bool, int, unsigned int, double, or interval).

Definition at line 1595 of file expression.h.

10.24.3.78 bool coco::coco::basic_alltype::is_allocated ( ) const [inline]

Check, whether the basic_alltype contains a complex type, i.e. one which has to be allocated (not bool, int, unsigned int, double, or interval).

Definition at line 1595 of file search_graph.cc.

10.24.3.79 bool coco::coco::basic_alltype::is_allocated ( ) const [inline]

Check, whether the basic_alltype contains a complex type, i.e. one which has to be allocated (not bool, int, unsigned int, double, or interval).

Definition at line 1595 of file search_graph.cc.

10.24.3.80 bool coco::coco::basic_alltype::is_allocated ( ) const [inline]

Check, whether the basic_alltype contains a complex type, i.e. one which has to be allocated (not bool, int, unsigned int, double, or interval).

Definition at line 1595 of file search_graph.cc.

10.24.3.81 bool coco::coco::basic_alltype::is_dense_matrix ( ) const [inline]

Check, whether the basic_alltype contains a dense matrix type.

Definition at line 1644 of file expression.h.

10.24.3.82 bool coco::coco::basic_alltype::is_dense_matrix ( ) const [inline]

Check, whether the basic_alltype contains a dense matrix type.

Definition at line 1644 of file search_graph.cc.

10.24.3.83 bool coco::coco::basic_alltype::is_dense_matrix ( ) const [inline]

Check, whether the basic_alltype contains a dense matrix type.

Definition at line 1644 of file search_graph.cc.

10.24.3.84 bool coco::coco::basic_alltype::is_dense_matrix ( ) const [inline]

Check, whether the basic_alltype contains a dense matrix type.

Definition at line 1644 of file search_graph.cc.

10.24.3.85 bool coco::coco::basic_alltype::is_dense_vector ( ) const [inline]

Check, whether the basic_alltype contains a dense vector type.

Definition at line 1607 of file expression.h.

10.24.3.86 bool coco::coco::basic_alltype::is_dense_vector ( ) const [inline]

Check, whether the basic_alltype contains a dense vector type.

Definition at line 1607 of file search_graph.cc.

10.24.3.87 bool coco::coco::basic_alltype::is_dense_vector ( ) const [inline]

Check, whether the basic_alltype contains a dense vector type.

Definition at line 1607 of file search_graph.cc.

10.24.3.88 bool coco::coco::basic_alltype::is_dense_vector ( ) const [inline]

Check, whether the basic_alltype contains a dense vector type.

Definition at line 1607 of file search_graph.cc.

10.24.3.89 bool coco::coco::basic_alltype::is_empty_matrix ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1648 of file expression.h.

10.24.3.90 bool coco::coco::basic_alltype::is_empty_matrix ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1648 of file search_graph.cc.

10.24.3.91 bool coco::coco::basic_alltype::is_empty_matrix ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1648 of file search_graph.cc.

10.24.3.92 bool coco::coco::basic_alltype::is_empty_matrix ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1648 of file search_graph.cc.

10.24.3.93 bool coco::coco::basic_alltype::is_empty_vector ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1613 of file expression.h.

10.24.3.94 bool coco::coco::basic_alltype::is_empty_vector ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1613 of file search_graph.cc.

10.24.3.95 bool coco::coco::basic_alltype::is_empty_vector ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1613 of file search_graph.cc.

10.24.3.96 bool coco::coco::basic_alltype::is_empty_vector ( ) const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 1613 of file search_graph.cc.

10.24.3.97 bool coco::coco::basic_alltype::is_matrix ( ) const [inline]

Check, whether the basic_alltype contains a matrix type.

Definition at line 1642 of file expression.h.

10.24.3.98 bool coco::coco::basic_alltype::is_matrix ( ) const [inline]

Check, whether the basic_alltype contains a matrix type.

Definition at line 1642 of file search_graph.cc.

10.24.3.99 bool coco::coco::basic_alltype::is_matrix ( ) const [inline]

Check, whether the basic_alltype contains a matrix type.

Definition at line 1642 of file search_graph.cc.

10.24.3.100 bool coco::coco::basic_alltype::is_matrix ( ) const [inline]

Check, whether the basic_alltype contains a matrix type.

Definition at line 1642 of file search_graph.cc.

10.24.3.101 bool coco::coco::basic_alltype::is_scalar ( ) const [inline]

Check, whether the basic_alltype contains a scalar type, i.e. neither a vector nor a matrix.

Definition at line 1598 of file expression.h.

10.24.3.102 bool coco::coco::basic_alltype::is_scalar ( ) const [inline]

Check, whether the basic_alltype contains a scalar type, i.e. neither a vector nor a matrix.

Definition at line 1598 of file search_graph.cc.

10.24.3.103 bool coco::coco::basic_alltype::is_scalar ( ) const [inline]

Check, whether the basic_alltype contains a scalar type, i.e. neither a vector nor a matrix.

Definition at line 1598 of file search_graph.cc.

10.24.3.104 bool coco::coco::basic_alltype::is_scalar ( ) const [inline]

Check, whether the basic_alltype contains a scalar type, i.e. neither a vector nor a matrix.

Definition at line 1598 of file search_graph.cc.

10.24.3.105 bool coco::coco::basic_alltype::is_sparse_matrix ( ) const [inline]

Check, whether the basic_alltype contains a sparse matrix type.

Definition at line 1646 of file search_graph.cc.

10.24.3.106 bool coco::coco::basic_alltype::is_sparse_matrix ( ) const [inline]

Check, whether the basic_alltype contains a sparse matrix type.

Definition at line 1646 of file search_graph.cc.

10.24.3.107 bool coco::coco::basic_alltype::is_sparse_matrix ( ) const [inline]

Check, whether the basic_alltype contains a sparse matrix type.

Definition at line 1646 of file expression.h.

10.24.3.108 bool coco::coco::basic_alltype::is_sparse_matrix ( ) const [inline]

Check, whether the basic_alltype contains a sparse matrix type.

Definition at line 1646 of file search_graph.cc.

10.24.3.109 bool coco::coco::basic_alltype::is_sparse_vector ( ) const [inline]

Check, whether the basic_alltype contains a sparse vector type.

Definition at line 1610 of file expression.h.

10.24.3.110 bool coco::coco::basic_alltype::is_sparse_vector ( ) const [inline]

Check, whether the basic_alltype contains a sparse vector type.

Definition at line 1610 of file search_graph.cc.

10.24.3.111 bool coco::coco::basic_alltype::is_sparse_vector ( ) const [inline]

Check, whether the basic_alltype contains a sparse vector type.

Definition at line 1610 of file search_graph.cc.

10.24.3.112 bool coco::coco::basic_alltype::is_sparse_vector ( ) const [inline]

Check, whether the basic_alltype contains a sparse vector type.

Definition at line 1610 of file search_graph.cc.

10.24.3.113 bool coco::coco::basic_alltype::is_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector type.

Definition at line 1601 of file expression.h.

10.24.3.114 bool coco::coco::basic_alltype::is_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector type.

Definition at line 1601 of file search_graph.cc.

10.24.3.115 bool coco::coco::basic_alltype::is_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector type.

Definition at line 1601 of file search_graph.cc.

10.24.3.116 bool coco::coco::basic_alltype::is_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector type.

Definition at line 1601 of file search_graph.cc.

10.24.3.117 bool coco::coco::basic_alltype::is_vector_of_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector of vector type.

Definition at line 1604 of file expression.h.

10.24.3.118 bool coco::coco::basic_alltype::is_vector_of_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector of vector type.

Definition at line 1604 of file search_graph.cc.

10.24.3.119 bool coco::coco::basic_alltype::is_vector_of_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector of vector type.

Definition at line 1604 of file search_graph.cc.

10.24.3.120 bool coco::coco::basic_alltype::is_vector_of_vector ( ) const [inline]

Check, whether the basic_alltype contains a vector of vector type.

Definition at line 1604 of file search_graph.cc.

10.24.3.121 const vmtl::sparse_matrix<interval>& coco::coco::basic_alltype::ism ( ) const [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1311 of file search_graph.cc.

10.24.3.122 const vmtl::sparse_matrix<interval>& coco::coco::basic_alltype::ism ( ) const [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1311 of file expression.h.

10.24.3.123 const vmtl::sparse_matrix<interval>& coco::coco::basic_alltype::ism ( ) const [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1311 of file search_graph.cc.

10.24.3.124 const vmtl::sparse_matrix<interval>& coco::coco::basic_alltype::ism ( ) const [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1311 of file search_graph.cc.

10.24.3.125 vmtl::sparse_matrix<interval>& coco::coco::basic_alltype::ism ( ) [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1582 of file expression.h.

10.24.3.126 vmtl::sparse_matrix<interval>& coco::coco::basic_alltype::ism ( ) [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1582 of file search_graph.cc.

10.24.3.127 vmtl::sparse_matrix<interval>& coco::coco::basic_alltype::ism ( ) [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1582 of file search_graph.cc.

10.24.3.128 vmtl::sparse_matrix<interval>& coco::coco::basic_alltype::ism ( ) [inline]

retrieve a sparse_matrix<interval> from the basic_alltype

Definition at line 1582 of file search_graph.cc.

10.24.3.129 bool coco::coco::basic_alltype::less_equal (const basic_alltype & *b*) const

Total order comparison operator

10.24.3.130 bool coco::coco::basic_alltype::less_equal (const basic_alltype & *b*) const

Total order comparison operator

10.24.3.131 bool coco::basic_alltype::less_equal (const basic_alltype & *b*) const

Total order comparison operator

Definition at line 439 of file basic_alltype.cc.

10.24.3.132 bool coco::coco::basic_alltype::less_equal (const basic_alltype & *b*) const

Total order comparison operator

10.24.3.133 bool coco::coco::basic_alltype::less_than (const basic_alltype & *b*) const

Total order comparison operator

10.24.3.134 bool coco::coco::basic_alltype::less_than (const basic_alltype & *b*) const

Total order comparison operator

10.24.3.135 bool coco::coco::basic_alltype::less_than (const basic_alltype & *b*) const

Total order comparison operator

10.24.3.136 bool coco::basic_alltype::less_than (const basic_alltype & *b*) const

Total order comparison operator

Definition at line 323 of file basic_alltype.cc.

10.24.3.137 const std::vector<int>& coco::coco::basic_alltype::n ( ) const [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1089 of file search_graph.cc.

10.24.3.138 const std::vector<int>& coco::coco::basic_alltype::n ( ) const [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1089 of file search_graph.cc.

10.24.3.139 const std::vector<int>& coco::coco::basic_alltype::n ( ) const [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1089 of file expression.h.

10.24.3.140 const std::vector<int>& coco::coco::basic_alltype::n ( ) const [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1089 of file search_graph.cc.

10.24.3.141 std::vector<int>& coco::coco::basic_alltype::n ( ) [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1361 of file search_graph.cc.

10.24.3.142 std::vector<int>& coco::coco::basic_alltype::n ( ) [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1361 of file search_graph.cc.

10.24.3.143 std::vector<int>& coco::coco::basic_alltype::n ( ) [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1361 of file expression.h.

10.24.3.144 std::vector<int>& coco::coco::basic_alltype::n ( ) [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 1361 of file search_graph.cc.

10.24.3.145 bool coco::coco::basic_alltype::nb ( ) const [inline]

retrieve a bool from the basic_alltype

Definition at line 1049 of file search_graph.cc.

10.24.3.146 bool coco::coco::basic_alltype::nb ( ) const [inline]

retrieve a bool from the basic_alltype

Definition at line 1049 of file search_graph.cc.

10.24.3.147 bool coco::coco::basic_alltype::nb ( ) const [inline]

retrieve a bool from the basic_alltype

Definition at line 1049 of file expression.h.

10.24.3.148 bool coco::coco::basic_alltype::nb ( ) const [inline]

retrieve a bool from the basic_alltype

Definition at line 1049 of file search_graph.cc.

10.24.3.149 bool& coco::coco::basic_alltype::nb ( ) [inline]

retrieve a bool from the basic_alltype

Definition at line 1321 of file search_graph.cc.

10.24.3.150 bool& coco::coco::basic_alltype::nb ( ) [inline]

retrieve a bool from the basic_alltype

Definition at line 1321 of file expression.h.

10.24.3.151 bool& coco::coco::basic_alltype::nb ( ) [inline]

retrieve a bool from the basic_alltype

Definition at line 1321 of file search_graph.cc.

10.24.3.152 bool& coco::coco::basic_alltype::nb ( ) [inline]

retrieve a bool from the basic_alltype

Definition at line 1321 of file search_graph.cc.

10.24.3.153 double coco::coco::basic_alltype::nd ( ) const [inline]

retrieve a double from the basic_alltype

Definition at line 1070 of file search_graph.cc.

10.24.3.154 double coco::coco::basic_alltype::nd ( ) const [inline]

retrieve a double from the basic_alltype

Definition at line 1070 of file search_graph.cc.

10.24.3.155 double coco::coco::basic_alltype::nd ( ) const [inline]

retrieve a double from the basic_alltype

Definition at line 1070 of file expression.h.

10.24.3.156 double coco::coco::basic_alltype::nd ( ) const [inline]

retrieve a double from the basic_alltype

Definition at line 1070 of file search_graph.cc.

10.24.3.157 double& coco::coco::basic_alltype::nd ( ) [inline]

retrieve a double from the basic_alltype

Definition at line 1342 of file search_graph.cc.

10.24.3.158 double& coco::coco::basic_alltype::nd ( ) [inline]

retrieve a double from the basic_alltype

Definition at line 1342 of file expression.h.

10.24.3.159 double& coco::coco::basic_alltype::nd ( ) [inline]

retrieve a double from the basic_alltype

Definition at line 1342 of file search_graph.cc.

10.24.3.160 double& coco::coco::basic_alltype::nd ( ) [inline]

retrieve a double from the basic_alltype

Definition at line 1342 of file search_graph.cc.

10.24.3.161 const vmtl::dense_matrix<int>& coco::coco::basic_alltype::ndm ( ) const [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1292 of file search_graph.cc.

10.24.3.162 const vmtl::dense_matrix<int>& coco::coco::basic_alltype::ndm ( ) const [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1292 of file expression.h.

10.24.3.163 const vmtl::dense_matrix<int>& coco::coco::basic_alltype::ndm ( ) const [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1292 of file search_graph.cc.

10.24.3.164 const vmtl::dense_matrix<int>& coco::coco::basic_alltype::ndm ( ) const [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1292 of file search_graph.cc.

10.24.3.165 vmtl::dense_matrix<int>& coco::coco::basic_alltype::ndm ( ) [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1563 of file search_graph.cc.

10.24.3.166 vmtl::dense_matrix<int>& coco::coco::basic_alltype::ndm ( ) [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1563 of file search_graph.cc.

10.24.3.167 vmtl::dense_matrix<int>& coco::coco::basic_alltype::ndm ( ) [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1563 of file search_graph.cc.

10.24.3.168 vmtl::dense_matrix<int>& coco::coco::basic_alltype::ndm ( ) [inline]

retrieve a dense_matrix<int> from the basic_alltype

Definition at line 1563 of file expression.h.

10.24.3.169 interval coco::coco::basic_alltype::ni ( ) const [inline]

retrieve an interval from the basic_alltype

Definition at line 1073 of file search_graph.cc.

10.24.3.170 interval coco::coco::basic_alltype::ni ( ) const [inline]

retrieve an interval from the basic_alltype

Definition at line 1073 of file search_graph.cc.

10.24.3.171 interval coco::coco::basic_alltype::ni ( ) const [inline]

retrieve an interval from the basic_alltype

Definition at line 1073 of file expression.h.

10.24.3.172 interval coco::coco::basic_alltype::ni ( ) const [inline]

retrieve an interval from the basic_alltype

Definition at line 1073 of file search_graph.cc.

10.24.3.173 interval_st& coco::coco::basic_alltype::ni ( ) [inline]

retrieve an interval from the basic_alltype

Definition at line 1345 of file search_graph.cc.

10.24.3.174 interval_st& coco::coco::basic_alltype::ni ( ) [inline]

retrieve an interval from the basic_alltype

Definition at line 1345 of file expression.h.

10.24.3.175 interval_st& coco::coco::basic_alltype::ni ( ) [inline]

retrieve an interval from the basic_alltype

Definition at line 1345 of file search_graph.cc.

10.24.3.176 interval_st& coco::coco::basic_alltype::ni ( ) [inline]

retrieve an interval from the basic_alltype

Definition at line 1345 of file search_graph.cc.

10.24.3.177 int coco::coco::basic_alltype::nn ( ) const [inline]

retrieve an int from the basic_alltype

Definition at line 1052 of file search_graph.cc.

10.24.3.178 int coco::coco::basic_alltype::nn ( ) const [inline]

retrieve an int from the basic_alltype

Definition at line 1052 of file expression.h.

10.24.3.179 int coco::coco::basic_alltype::nn ( ) const [inline]

retrieve an int from the basic_alltype

Definition at line 1052 of file search_graph.cc.

10.24.3.180 int coco::coco::basic_alltype::nn ( ) const [inline]

retrieve an int from the basic_alltype

Definition at line 1052 of file search_graph.cc.

10.24.3.181 int& coco::coco::basic_alltype::nn ( ) [inline]

retrieve an int from the basic_alltype

Definition at line 1324 of file search_graph.cc.

10.24.3.182 int& coco::coco::basic_alltype::nn ( ) [inline]

retrieve an int from the basic_alltype

Definition at line 1324 of file expression.h.

10.24.3.183 int& coco::coco::basic_alltype::nn ( ) [inline]

retrieve an int from the basic_alltype

Definition at line 1324 of file search_graph.cc.

10.24.3.184 int& coco::coco::basic_alltype::nn ( ) [inline]

retrieve an int from the basic_alltype

Definition at line 1324 of file search_graph.cc.

10.24.3.185 const vmtl::sparse_matrix<int>& coco::coco::basic_alltype::nsm ( ) const [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1308 of file search_graph.cc.

10.24.3.186 const vmtl::sparse_matrix<int>& coco::coco::basic_alltype::nsm ( ) const [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1308 of file expression.h.

10.24.3.187 const vmtl::sparse_matrix<int>& coco::coco::basic_alltype::nsm ( ) const [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1308 of file search_graph.cc.

10.24.3.188 const vmtl::sparse_matrix<int>& coco::coco::basic_alltype::nsm ( ) const [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1308 of file search_graph.cc.

10.24.3.189 vmtl::sparse_matrix<int>& coco::coco::basic_alltype::nsm ( ) [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1579 of file search_graph.cc.

10.24.3.190 vmtl::sparse_matrix<int>& coco::coco::basic_alltype::nsm ( ) [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1579 of file search_graph.cc.

10.24.3.191 vmtl::sparse_matrix<int>& coco::coco::basic_alltype::nsm ( ) [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1579 of file search_graph.cc.

10.24.3.192 vmtl::sparse_matrix<int>& coco::coco::basic_alltype::nsm ( ) [inline]

retrieve a sparse_matrix<int> from the basic_alltype

Definition at line 1579 of file expression.h.

10.24.3.193 unsigned int coco::coco::basic_alltype::nu ( ) const [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1061 of file search_graph.cc.

10.24.3.194 unsigned int coco::coco::basic_alltype::nu ( ) const [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1061 of file search_graph.cc.

10.24.3.195 unsigned int coco::coco::basic_alltype::nu ( ) const [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1061 of file search_graph.cc.

10.24.3.196 unsigned int coco::coco::basic_alltype::nu ( ) const [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1061 of file expression.h.

10.24.3.197 unsigned int& coco::coco::basic_alltype::nu ( ) [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1333 of file search_graph.cc.

10.24.3.198 unsigned int& coco::coco::basic_alltype::nu ( ) [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1333 of file expression.h.

10.24.3.199 unsigned int& coco::coco::basic_alltype::nu ( ) [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1333 of file search_graph.cc.

10.24.3.200 unsigned int& coco::coco::basic_alltype::nu ( ) [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 1333 of file search_graph.cc.

10.24.3.201 bool coco::coco::basic_alltype::operator!= (const basic_alltype & *b*) const

Standard comparison operator

10.24.3.202 bool coco::basic_alltype::operator!= (const basic_alltype & *b*) const

Standard comparison operator

Definition at line 318 of file basic_alltype.cc.

10.24.3.203 bool coco::coco::basic_alltype::operator!= (const basic_alltype & *b*) const

Standard comparison operator

10.24.3.204 bool coco::coco::basic_alltype::operator!= (const basic_alltype & *b*) const

Standard comparison operator

10.24.3.205 basic_alltype& coco::coco::basic_alltype::operator= (bool __x) [inline]

Assignment Operator, assigning the bool value __x

Definition at line 647 of file expression.h.

10.24.3.206 basic_alltype& coco::coco::basic_alltype::operator= (bool __x) [inline]

Assignment Operator, assigning the bool value __x

Definition at line 647 of file search_graph.cc.

10.24.3.207 basic_alltype& coco::coco::basic_alltype::operator= (bool __x) [inline]

Assignment Operator, assigning the bool value __x

Definition at line 647 of file search_graph.cc.

10.24.3.208 basic_alltype& coco::coco::basic_alltype::operator= (bool __x) [inline]

Assignment Operator, assigning the bool value __x

Definition at line 647 of file search_graph.cc.

10.24.3.209 basic_alltype& coco::coco::basic_alltype::operator= (int __x) [inline]

Assignment Operator, assigning the int value __x

Definition at line 653 of file search_graph.cc.

10.24.3.210 basic_alltype& coco::coco::basic_alltype::operator= (int __x) [inline]

Assignment Operator, assigning the int value __x

Definition at line 653 of file search_graph.cc.

10.24.3.211 basic_alltype& coco::coco::basic_alltype::operator= (int __x) [inline]

Assignment Operator, assigning the int value __x

Definition at line 653 of file expression.h.

10.24.3.212 basic_alltype& coco::coco::basic_alltype::operator= (int __x) [inline]

Assignment Operator, assigning the int value __x

Definition at line 653 of file search_graph.cc.

10.24.3.213 basic_alltype& coco::coco::basic_alltype::operator= (unsigned int __x) [inline]

Assignment Operator, assigning the unsigned int value __x

Definition at line 659 of file search_graph.cc.

10.24.3.214 basic_alltype& coco::coco::basic_alltype::operator= (unsigned int __x) [inline]

Assignment Operator, assigning the unsigned int value __x

Definition at line 659 of file search_graph.cc.

10.24.3.215 basic_alltype& coco::coco::basic_alltype::operator= (unsigned int __x) [inline]

Assignment Operator, assigning the unsigned int value __x

Definition at line 659 of file expression.h.

10.24.3.216 basic_alltype& coco::coco::basic_alltype::operator= (unsigned int __x) [inline]

Assignment Operator, assigning the unsigned int value __x

Definition at line 659 of file search_graph.cc.

10.24.3.217 basic_alltype& coco::coco::basic_alltype::operator= (double __x) [inline]

Assignment Operator, assigning the double value __x

Definition at line 665 of file expression.h.

10.24.3.218 basic_alltype& coco::coco::basic_alltype::operator= (double __x) [inline]

Assignment Operator, assigning the double value __x

Definition at line 665 of file search_graph.cc.

10.24.3.219 basic_alltype& coco::coco::basic_alltype::operator= (double __x) [inline]

Assignment Operator, assigning the double value __x

Definition at line 665 of file search_graph.cc.

10.24.3.220 basic_alltype& coco::coco::basic_alltype::operator= (double __x) [inline]

Assignment Operator, assigning the double value __x

Definition at line 665 of file search_graph.cc.

10.24.3.221 basic_alltype& coco::coco::basic_alltype::operator= (interval __x) [inline]

Assignment Operator, assigning the interval value __x

Definition at line 671 of file search_graph.cc.

10.24.3.222 basic_alltype& coco::coco::basic_alltype::operator= (interval __x) [inline]

Assignment Operator, assigning the interval value __x

Definition at line 671 of file search_graph.cc.

10.24.3.223 basic_alltype& coco::coco::basic_alltype::operator= (interval __x) [inline]

Assignment Operator, assigning the interval value ___x

Definition at line 671 of file expression.h.

10.24.3.224 basic_alltype& coco::coco::basic_alltype::operator= (interval __x) [inline]

Assignment Operator, assigning the interval value ___x

Definition at line 671 of file search_graph.cc.

10.24.3.225 basic_alltype& coco::coco::basic_alltype::operator= (void * __x) [inline]

Assignment Operator, assigning the void* value ___x

Definition at line 677 of file search_graph.cc.

10.24.3.226 basic_alltype& coco::coco::basic_alltype::operator= (void * __x) [inline]

Assignment Operator, assigning the void* value ___x

Definition at line 677 of file search_graph.cc.

10.24.3.227 basic_alltype& coco::coco::basic_alltype::operator= (void * __x) [inline]

Assignment Operator, assigning the void* value ___x

Definition at line 677 of file expression.h.

10.24.3.228 basic_alltype& coco::coco::basic_alltype::operator= (void * __x) [inline]

Assignment Operator, assigning the void* value ___x

Definition at line 677 of file search_graph.cc.

10.24.3.229 basic_alltype& coco::coco::basic_alltype::operator= (const std::string & __x) [inline]

Assignment Operator, assigning the string value ___x

Definition at line 683 of file search_graph.cc.

10.24.3.230 basic_alltype& coco::coco::basic_alltype::operator= (const std::string & __x) [inline]

Assignment Operator, assigning the string value ___x

Definition at line 683 of file search_graph.cc.

10.24.3.231 basic_alltype& coco::coco::basic_alltype::operator= (const std::string & __x) [inline]

Assignment Operator, assigning the string value ___x

Definition at line 683 of file search_graph.cc.

10.24.3.232 basic_alltype& coco::coco::basic_alltype::operator= (const std::string & __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 683 of file expression.h.

10.24.3.233 basic_alltype& coco::coco::basic_alltype::operator= (const char * __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 690 of file expression.h.

10.24.3.234 basic_alltype& coco::coco::basic_alltype::operator= (const char * __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 690 of file search_graph.cc.

10.24.3.235 basic_alltype& coco::coco::basic_alltype::operator= (const char * __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 690 of file search_graph.cc.

10.24.3.236 basic_alltype& coco::coco::basic_alltype::operator= (const char * __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 690 of file search_graph.cc.

10.24.3.237 basic_alltype& coco::coco::basic_alltype::operator= (const num::Number & __x) [inline]

Assignment Operator, assigning the Number value __x

Definition at line 697 of file search_graph.cc.

10.24.3.238 basic_alltype& coco::coco::basic_alltype::operator= (const num::Number & __x) [inline]

Assignment Operator, assigning the Number value __x

Definition at line 697 of file search_graph.cc.

10.24.3.239 basic_alltype& coco::coco::basic_alltype::operator= (const num::Number & __x) [inline]

Assignment Operator, assigning the Number value __x

Definition at line 697 of file expression.h.

10.24.3.240 basic_alltype& coco::coco::basic_alltype::operator= (const num::Number & __x) [inline]

Assignment Operator, assigning the Number value __x

Definition at line 697 of file search_graph.cc.

10.24.3.241 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< bool > & __x) [inline]

Assignment Operator, assigning the vector<bool> value __x

Definition at line 704 of file search_graph.cc.

10.24.3.242 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< bool > & __x) [inline]

Assignment Operator, assigning the vector<bool> value __x

Definition at line 704 of file search_graph.cc.

10.24.3.243 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< bool > & __x) [inline]

Assignment Operator, assigning the vector<bool> value __x

Definition at line 704 of file expression.h.

10.24.3.244 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< bool > & __x) [inline]

Assignment Operator, assigning the vector<bool> value __x

Definition at line 704 of file search_graph.cc.

10.24.3.245 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< int > & __x) [inline]

Assignment Operator, assigning the vector<int> value __x

Definition at line 711 of file search_graph.cc.

10.24.3.246 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< int > & __x) [inline]

Assignment Operator, assigning the vector<int> value __x

Definition at line 711 of file search_graph.cc.

10.24.3.247 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< int > & __x) [inline]

Assignment Operator, assigning the vector<int> value __x

Definition at line 711 of file expression.h.

10.24.3.248 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< int > & __x) [inline]

Assignment Operator, assigning the vector<int> value __x

Definition at line 711 of file search_graph.cc.

10.24.3.249 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the vector<unsigned int> value __x

Definition at line 718 of file search_graph.cc.

10.24.3.250 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the vector<unsigned int> value __x

Definition at line 718 of file search_graph.cc.

10.24.3.251 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the vector<unsigned int> value __x

Definition at line 718 of file expression.h.

10.24.3.252 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the vector<unsigned int> value __x

Definition at line 718 of file search_graph.cc.

10.24.3.253 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< double > & __x) [inline]

Assignment Operator, assigning the vector<double> value __x

Definition at line 725 of file search_graph.cc.

10.24.3.254 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< double > & __x) [inline]

Assignment Operator, assigning the vector<double> value __x

Definition at line 725 of file search_graph.cc.

10.24.3.255 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< double > & __x) [inline]

Assignment Operator, assigning the vector<double> value __x

Definition at line 725 of file expression.h.

10.24.3.256 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< double > & __x) [inline]

Assignment Operator, assigning the vector<double> value __x

Definition at line 725 of file search_graph.cc.

10.24.3.257 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< interval > & __x) [inline]

Assignment Operator, assigning the vector<interval> value __x

Definition at line 732 of file search_graph.cc.

10.24.3.258 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< interval > & __x) [inline]

Assignment Operator, assigning the vector<interval> value __x

Definition at line 732 of file search_graph.cc.

10.24.3.259 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< interval > & __x) [inline]

Assignment Operator, assigning the vector<interval> value __x

Definition at line 732 of file search_graph.cc.

10.24.3.260 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< interval > & __x) [inline]

Assignment Operator, assigning the vector<interval> value __x

Definition at line 732 of file expression.h.

10.24.3.261 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::string > & __x) [inline]

Assignment Operator, assigning the vector<string> value __x

Definition at line 739 of file search_graph.cc.

10.24.3.262 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::string > & __x) [inline]

Assignment Operator, assigning the vector<string> value __x

Definition at line 739 of file search_graph.cc.

10.24.3.263 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::string > & __x) [inline]

Assignment Operator, assigning the vector<string> value __x

Definition at line 739 of file search_graph.cc.

10.24.3.264 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::string > & __x) [inline]

Assignment Operator, assigning the vector<string> value __x

Definition at line 739 of file expression.h.

10.24.3.265 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< num::Number > & __x) [inline]

Assignment Operator, assigning the vector<Number> value __x

Definition at line 746 of file expression.h.

10.24.3.266 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< num::Number > & __x) [inline]

Assignment Operator, assigning the vector<Number> value __x

Definition at line 746 of file search_graph.cc.

10.24.3.267 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< num::Number > & __x) [inline]

Assignment Operator, assigning the vector<Number> value __x

Definition at line 746 of file search_graph.cc.

10.24.3.268 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< num::Number > & __x) [inline]

Assignment Operator, assigning the vector<Number> value __x

Definition at line 746 of file search_graph.cc.

10.24.3.269 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< bool > > & __x) [inline]

Assignment Operator, assigning the vector<vector<bool> > value __x

Definition at line 753 of file expression.h.

10.24.3.270 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< bool > > & __x) [inline]

Assignment Operator, assigning the vector<vector<bool> > value __x

Definition at line 753 of file search_graph.cc.

10.24.3.271 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< bool > > & __x) [inline]

Assignment Operator, assigning the vector<vector<bool> > value __x

Definition at line 753 of file search_graph.cc.

10.24.3.272 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< bool > > & __x) [inline]

Assignment Operator, assigning the vector<vector<bool> > value __x

Definition at line 753 of file search_graph.cc.

10.24.3.273 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<int> > value __x

Definition at line 760 of file expression.h.

10.24.3.274 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<int> > value __x

Definition at line 760 of file search_graph.cc.

10.24.3.275 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<int> > value __x

Definition at line 760 of file search_graph.cc.

10.24.3.276 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<int> > value __x

Definition at line 760 of file search_graph.cc.

10.24.3.277 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< unsigned int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<unsigned int> > value __x

Definition at line 767 of file expression.h.

10.24.3.278 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< unsigned int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<unsigned int> > value __x

Definition at line 767 of file search_graph.cc.

10.24.3.279 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< unsigned int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<unsigned int> > value __x

Definition at line 767 of file search_graph.cc.

10.24.3.280 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< unsigned int > > & __x) [inline]

Assignment Operator, assigning the vector<vector<unsigned int> > value __x

Definition at line 767 of file search_graph.cc.

10.24.3.281 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< double > > & __x) [inline]

Assignment Operator, assigning the vector<vector<double> > value __x

Definition at line 774 of file search_graph.cc.

10.24.3.282 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< double > > & __x) [inline]

Assignment Operator, assigning the vector<vector<double> > value __x

Definition at line 774 of file search_graph.cc.

10.24.3.283 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< double > > & __x) [inline]

Assignment Operator, assigning the vector<vector<double> > value __x

Definition at line 774 of file expression.h.

10.24.3.284 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< double > > & __x) [inline]

Assignment Operator, assigning the vector<vector<double> > value __x

Definition at line 774 of file search_graph.cc.

10.24.3.285 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< interval > > & __x) [inline]

Assignment Operator, assigning the vector<vector<interval> > value __x

Definition at line 781 of file search_graph.cc.

10.24.3.286 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< interval > > & __x) [inline]

Assignment Operator, assigning the vector<vector<interval> > value __x

Definition at line 781 of file expression.h.

10.24.3.287 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< interval > > & __x) [inline]

Assignment Operator, assigning the vector<vector<interval> > value __x

Definition at line 781 of file search_graph.cc.

10.24.3.288 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< interval > > & __x) [inline]

Assignment Operator, assigning the vector<vector<interval> > value __x

Definition at line 781 of file search_graph.cc.

10.24.3.289 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< std::string > > & __x) [inline]

Assignment Operator, assigning the vector<vector<string> > value __x

Definition at line 788 of file search_graph.cc.

10.24.3.290 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< std::string > > & __x) [inline]

Assignment Operator, assigning the vector<vector<string> > value __x

Definition at line 788 of file search_graph.cc.

10.24.3.291 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< std::string > > & __x) [inline]

Assignment Operator, assigning the vector<vector<string> > value __x

Definition at line 788 of file expression.h.

10.24.3.292 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< std::string > > & __x) [inline]

Assignment Operator, assigning the vector<vector<string> > value __x

Definition at line 788 of file search_graph.cc.

10.24.3.293 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< num::Number > > & __x) [inline]

Assignment Operator, assigning the vector<vector<Number> > value __x

Definition at line 795 of file search_graph.cc.

10.24.3.294 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< num::Number > > & __x) [inline]

Assignment Operator, assigning the vector<vector<Number> > value __x

Definition at line 795 of file search_graph.cc.

10.24.3.295 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< num::Number > > & __x) [inline]

Assignment Operator, assigning the vector<vector<Number> > value __x

Definition at line 795 of file expression.h.

10.24.3.296 basic_alltype& coco::coco::basic_alltype::operator= (const std::vector< std::vector< num::Number > > & __x) [inline]

Assignment Operator, assigning the vector<vector<Number> > value __x

Definition at line 795 of file search_graph.cc.

10.24.3.297 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< bool > & __x) [inline]

Assignment Operator, assigning the sparse_vector<bool> value __x

Definition at line 802 of file search_graph.cc.

10.24.3.298 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< bool > & __x) [inline]

Assignment Operator, assigning the sparse_vector<bool> value __x

Definition at line 802 of file search_graph.cc.

10.24.3.299 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< bool > & __x) [inline]

Assignment Operator, assigning the sparse_vector<bool> value __x

Definition at line 802 of file expression.h.

10.24.3.300 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< bool > & __x) [inline]

Assignment Operator, assigning the sparse_vector<bool> value __x

Definition at line 802 of file search_graph.cc.

10.24.3.301 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<int> value __x

Definition at line 809 of file search_graph.cc.

10.24.3.302 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<int> value __x

Definition at line 809 of file search_graph.cc.

10.24.3.303 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<int> value __x

Definition at line 809 of file expression.h.

10.24.3.304 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<int> value __x

Definition at line 809 of file search_graph.cc.

10.24.3.305 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<unsigned int> value __x

Definition at line 816 of file search_graph.cc.

10.24.3.306 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<unsigned int> value __x

Definition at line 816 of file search_graph.cc.

10.24.3.307 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<unsigned int> value __x

Definition at line 816 of file expression.h.

10.24.3.308 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< unsigned int > & __x) [inline]

Assignment Operator, assigning the sparse_vector<unsigned int> value __x

Definition at line 816 of file search_graph.cc.

10.24.3.309 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< double > & __x) [inline]

Assignment Operator, assigning the sparse_vector<double> value __x

Definition at line 823 of file search_graph.cc.

10.24.3.310 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< double > & __x) [inline]

Assignment Operator, assigning the sparse_vector<double> value __x

Definition at line 823 of file search_graph.cc.

10.24.3.311 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< double > & __x) [inline]

Assignment Operator, assigning the sparse_vector<double> value __x

Definition at line 823 of file expression.h.

10.24.3.312 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< double > & __x) [inline]

Assignment Operator, assigning the sparse_vector<double> value __x

Definition at line 823 of file search_graph.cc.

10.24.3.313 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< interval > & __x) [inline]

Assignment Operator, assigning the sparse_vector<interval> value __x

Definition at line 830 of file search_graph.cc.

10.24.3.314 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< interval > & __x) [inline]

Assignment Operator, assigning the sparse_vector<interval> value __x

Definition at line 830 of file search_graph.cc.

10.24.3.315 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< interval > & __x) [inline]

Assignment Operator, assigning the sparse_vector<interval> value __x

Definition at line 830 of file expression.h.

10.24.3.316 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< interval > & __x) [inline]

Assignment Operator, assigning the sparse_vector<interval> value __x

Definition at line 830 of file search_graph.cc.

10.24.3.317 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_vector<string> value __x

Definition at line 837 of file search_graph.cc.

10.24.3.318 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_vector<string> value __x

Definition at line 837 of file search_graph.cc.

10.24.3.319 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_vector<string> value __x

Definition at line 837 of file expression.h.

10.24.3.320 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_vector<string> value __x

Definition at line 837 of file search_graph.cc.

10.24.3.321 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_vector<Number> value __x

Definition at line 844 of file search_graph.cc.

10.24.3.322 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_vector<Number> value __x

Definition at line 844 of file search_graph.cc.

10.24.3.323 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_vector<Number> value __x

Definition at line 844 of file search_graph.cc.

10.24.3.324 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_vector< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_vector<Number> value __x

Definition at line 844 of file expression.h.

10.24.3.325 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< double > & __x) [inline]

Assignment Operator, assigning the dense_matrix<double> value __x

Definition at line 851 of file search_graph.cc.

10.24.3.326 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< double > & __x) [inline]

Assignment Operator, assigning the dense_matrix<double> value __x

Definition at line 851 of file search_graph.cc.

10.24.3.327 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< double > & __x) [inline]

Assignment Operator, assigning the dense_matrix<double> value __x

Definition at line 851 of file search_graph.cc.

10.24.3.328 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< double > & __x) [inline]

Assignment Operator, assigning the dense_matrix<double> value __x

Definition at line 851 of file expression.h.

10.24.3.329 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< int > & __x) [inline]

Assignment Operator, assigning the dense_matrix<int> value __x

Definition at line 858 of file expression.h.

10.24.3.330 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< int > & __x) [inline]

Assignment Operator, assigning the dense_matrix<int> value __x

Definition at line 858 of file search_graph.cc.

10.24.3.331 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< int > & __x) [inline]

Assignment Operator, assigning the dense_matrix<int> value __x

Definition at line 858 of file search_graph.cc.

10.24.3.332 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< int > & __x) [inline]

Assignment Operator, assigning the dense_matrix<int> value __x

Definition at line 858 of file search_graph.cc.

10.24.3.333 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< interval > & __x) [inline]

Assignment Operator, assigning the dense_matrix<interval> value __x

Definition at line 865 of file expression.h.

10.24.3.334 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< interval > & __x) [inline]

Assignment Operator, assigning the dense_matrix<interval> value __x

Definition at line 865 of file search_graph.cc.

10.24.3.335 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< interval > & __x) [inline]

Assignment Operator, assigning the dense_matrix<interval> value __x

Definition at line 865 of file search_graph.cc.

10.24.3.336 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< interval > & __x) [inline]

Assignment Operator, assigning the dense_matrix<interval> value __x

Definition at line 865 of file search_graph.cc.

10.24.3.337 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the dense_matrix<string> value __x

Definition at line 872 of file expression.h.

10.24.3.338 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the dense_matrix<string> value __x

Definition at line 872 of file search_graph.cc.

10.24.3.339 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the dense_matrix<string> value __x

Definition at line 872 of file search_graph.cc.

10.24.3.340 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the dense_matrix<string> value __x

Definition at line 872 of file search_graph.cc.

10.24.3.341 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the dense_matrix<Number> value __x

Definition at line 879 of file expression.h.

10.24.3.342 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the dense_matrix<Number> value __x

Definition at line 879 of file search_graph.cc.

10.24.3.343 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the dense_matrix<Number> value __x

Definition at line 879 of file search_graph.cc.

10.24.3.344 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::dense_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the dense_matrix<Number> value __x

Definition at line 879 of file search_graph.cc.

10.24.3.345 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< double > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<double> value __x

Definition at line 886 of file search_graph.cc.

10.24.3.346 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< double > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<double> value __x

Definition at line 886 of file expression.h.

10.24.3.347 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< double > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<double> value __x

Definition at line 886 of file search_graph.cc.

10.24.3.348 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< double > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<double> value __x

Definition at line 886 of file search_graph.cc.

10.24.3.349 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< int > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<int> value __x

Definition at line 893 of file expression.h.

10.24.3.350 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< int > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<int> value __x

Definition at line 893 of file search_graph.cc.

10.24.3.351 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< int > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<int> value __x

Definition at line 893 of file search_graph.cc.

10.24.3.352 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< int > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<int> value __x

Definition at line 893 of file search_graph.cc.

10.24.3.353 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< interval > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<interval> value __x

Definition at line 900 of file search_graph.cc.

10.24.3.354 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< interval > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<interval> value __x

Definition at line 900 of file expression.h.

10.24.3.355 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< interval > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<interval> value __x

Definition at line 900 of file search_graph.cc.

10.24.3.356 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< interval > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<interval> value __x

Definition at line 900 of file search_graph.cc.

10.24.3.357 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<string> value __x

Definition at line 907 of file search_graph.cc.

10.24.3.358 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<string> value __x

Definition at line 907 of file expression.h.

10.24.3.359 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<string> value __x

Definition at line 907 of file search_graph.cc.

10.24.3.360 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< std::string > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<string> value __x

Definition at line 907 of file search_graph.cc.

10.24.3.361 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<Number> value __x

Definition at line 914 of file search_graph.cc.

10.24.3.362 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<Number> value __x

Definition at line 914 of file search_graph.cc.

10.24.3.363 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<Number> value __x

Definition at line 914 of file search_graph.cc.

10.24.3.364 basic_alltype& coco::coco::basic_alltype::operator= (const vmtl::sparse_matrix< num::Number > & __x) [inline]

Assignment Operator, assigning the sparse_matrix<Number> value __x

Definition at line 914 of file expression.h.

10.24.3.365 basic_alltype& coco::coco::basic_alltype::operator= (const basic_alltype & __a) [inline]

Standard Assignment Operator

Definition at line 921 of file search_graph.cc.

10.24.3.366 basic_alltype& coco::coco::basic_alltype::operator= (const basic_alltype & __a) [inline]

Standard Assignment Operator

Definition at line 921 of file search_graph.cc.

10.24.3.367 basic_alltype& coco::coco::basic_alltype::operator= (const basic_alltype & __a) [inline]

Standard Assignment Operator

Definition at line 921 of file expression.h.

10.24.3.368 basic_alltype& coco::coco::basic_alltype::operator= (const basic_alltype & __a) [inline]

Standard Assignment Operator

Definition at line 921 of file search_graph.cc.

10.24.3.369 bool coco::coco::basic_alltype::operator== (const basic_alltype & *b*) const

Standard comparison operator

10.24.3.370 bool coco::basic_alltype::operator== (const basic_alltype & *b*) const

Standard comparison operator

Definition at line 182 of file basic_alltype.cc.

10.24.3.371 bool coco::coco::basic_alltype::operator== (const basic_alltype & *b*) const

Standard comparison operator

10.24.3.372 bool coco::coco::basic_alltype::operator== (const basic_alltype & *b*) const

Standard comparison operator

10.24.3.373 const void* coco::coco::basic_alltype::p ( ) const [inline]

retrieve a void* from the basic_alltype

Definition at line 1076 of file search_graph.cc.

10.24.3.374 const void* coco::coco::basic_alltype::p ( ) const [inline]

retrieve a void* from the basic_alltype

Definition at line 1076 of file expression.h.

10.24.3.375 const void* coco::coco::basic_alltype::p ( ) const [inline]

retrieve a void* from the basic_alltype

Definition at line 1076 of file search_graph.cc.

10.24.3.376 const void* coco::coco::basic_alltype::p ( ) const [inline]

retrieve a void* from the basic_alltype

Definition at line 1076 of file search_graph.cc.

10.24.3.377 void*& coco::coco::basic_alltype::p ( ) [inline]

retrieve a void* reference from the basic_alltype

Definition at line 1348 of file search_graph.cc.

10.24.3.378 void*& coco::coco::basic_alltype::p ( ) [inline]

retrieve a void* reference from the basic_alltype

Definition at line 1348 of file expression.h.

10.24.3.379 void*& coco::coco::basic_alltype::p ( ) [inline]

retrieve a void* reference from the basic_alltype

Definition at line 1348 of file search_graph.cc.

10.24.3.380 void*& coco::coco::basic_alltype::p ( ) [inline]

retrieve a void* reference from the basic_alltype

Definition at line 1348 of file search_graph.cc.

10.24.3.381 const std::string& coco::coco::basic_alltype::s ( ) const [inline]

retrieve a string from the basic_alltype

Definition at line 1079 of file expression.h.

10.24.3.382 const std::string& coco::coco::basic_alltype::s ( ) const [inline]

retrieve a string from the basic_alltype

Definition at line 1079 of file search_graph.cc.

10.24.3.383 const std::string& coco::coco::basic_alltype::s ( ) const [inline]

retrieve a string from the basic_alltype

Definition at line 1079 of file search_graph.cc.

10.24.3.384 const std::string& coco::coco::basic_alltype::s ( ) const [inline]

retrieve a string from the basic_alltype

Definition at line 1079 of file search_graph.cc.

10.24.3.385 std::string& coco::coco::basic_alltype::s ( ) [inline]

retrieve a string from the basic_alltype

Definition at line 1351 of file search_graph.cc.

10.24.3.386 std::string& coco::coco::basic_alltype::s ( ) [inline]

retrieve a string from the basic_alltype

Definition at line 1351 of file expression.h.

10.24.3.387 std::string& coco::coco::basic_alltype::s ( ) [inline]

retrieve a string from the basic_alltype

Definition at line 1351 of file search_graph.cc.

10.24.3.388 std::string& coco::coco::basic_alltype::s ( ) [inline]

retrieve a string from the basic_alltype

Definition at line 1351 of file search_graph.cc.

10.24.3.389 const vmtl::sparse_vector<bool>& coco::coco::basic_alltype::sb ( ) const [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1224 of file search_graph.cc.

10.24.3.390 const vmtl::sparse_vector<bool>& coco::coco::basic_alltype::sb ( ) const [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1224 of file expression.h.

10.24.3.391 const vmtl::sparse_vector<bool>& coco::coco::basic_alltype::sb ( ) const [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1224 of file search_graph.cc.

10.24.3.392 const vmtl::sparse_vector<bool>& coco::coco::basic_alltype::sb ( ) const [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1224 of file search_graph.cc.

10.24.3.393 vmtl::sparse_vector<bool>& coco::coco::basic_alltype::sb ( ) [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1495 of file search_graph.cc.

10.24.3.394 vmtl::sparse_vector<bool>& coco::coco::basic_alltype::sb ( ) [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1495 of file search_graph.cc.

10.24.3.395 vmtl::sparse_vector<bool>& coco::coco::basic_alltype::sb ( ) [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1495 of file expression.h.

10.24.3.396 vmtl::sparse_vector<bool>& coco::coco::basic_alltype::sb ( ) [inline]

retrieve a sparse_vector<bool> from the basic_alltype

Definition at line 1495 of file search_graph.cc.

10.24.3.397 const vmtl::sparse_vector<std::string>& coco::coco::basic_alltype::sc ( ) const [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1280 of file search_graph.cc.

10.24.3.398 const vmtl::sparse_vector<std::string>& coco::coco::basic_alltype::sc ( ) const [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1280 of file expression.h.

10.24.3.399 const vmtl::sparse_vector<std::string>& coco::coco::basic_alltype::sc ( ) const [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1280 of file search_graph.cc.

10.24.3.400 const vmtl::sparse_vector<std::string>& coco::coco::basic_alltype::sc ( ) const [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1280 of file search_graph.cc.

10.24.3.401 vmtl::sparse_vector<std::string>& coco::coco::basic_alltype::sc ( ) [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1551 of file search_graph.cc.

10.24.3.402 vmtl::sparse_vector<std::string>& coco::coco::basic_alltype::sc ( ) [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1551 of file search_graph.cc.

10.24.3.403 vmtl::sparse_vector<std::string>& coco::coco::basic_alltype::sc ( ) [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1551 of file search_graph.cc.

10.24.3.404 vmtl::sparse_vector<std::string>& coco::coco::basic_alltype::sc ( ) [inline]

retrieve a sparse_vector<string> from the basic_alltype

Definition at line 1551 of file expression.h.

10.24.3.405 const vmtl::sparse_vector<double>& coco::coco::basic_alltype::sd ( ) const [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1272 of file search_graph.cc.

10.24.3.406 const vmtl::sparse_vector<double>& coco::coco::basic_alltype::sd ( ) const [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1272 of file expression.h.

10.24.3.407 const vmtl::sparse_vector<double>& coco::coco::basic_alltype::sd ( ) const [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1272 of file search_graph.cc.

10.24.3.408 const vmtl::sparse_vector<double>& coco::coco::basic_alltype::sd ( ) const [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1272 of file search_graph.cc.

10.24.3.409 vmtl::sparse_vector<double>& coco::coco::basic_alltype::sd ( ) [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1543 of file search_graph.cc.

10.24.3.410 vmtl::sparse_vector<double>& coco::coco::basic_alltype::sd ( ) [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1543 of file expression.h.

10.24.3.411 vmtl::sparse_vector<double>& coco::coco::basic_alltype::sd ( ) [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1543 of file search_graph.cc.

10.24.3.412 vmtl::sparse_vector<double>& coco::coco::basic_alltype::sd ( ) [inline]

retrieve a sparse_vector<double> from the basic_alltype

Definition at line 1543 of file search_graph.cc.

10.24.3.413 basic_alltype& coco::coco::basic_alltype::set_cdm (vmtl::dense_matrix< std::string > * *__m*) [inline]

This method assigns a dense_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<string>.

Definition at line 973 of file search_graph.cc.

10.24.3.414 basic_alltype& coco::coco::basic_alltype::set_cdm (vmtl::dense_matrix< std::string > * *__m*) [inline]

This method assigns a dense_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<string>.

Definition at line 973 of file search_graph.cc.

10.24.3.415 basic_alltype& coco::coco::basic_alltype::set_cdm (vmtl::dense_matrix< std::string > * __m) [inline]

This method assigns a dense_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<string>.

Definition at line 973 of file expression.h.

10.24.3.416 basic_alltype& coco::coco::basic_alltype::set_cdm (vmtl::dense_matrix< std::string > * __m) [inline]

This method assigns a dense_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<string>.

Definition at line 973 of file search_graph.cc.

10.24.3.417 basic_alltype& coco::coco::basic_alltype::set_csm (vmtl::sparse_matrix< std::string > * __m) [inline]

This method assigns a sparse_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<string>.

Definition at line 1028 of file search_graph.cc.

10.24.3.418 basic_alltype& coco::coco::basic_alltype::set_csm (vmtl::sparse_matrix< std::string > * __m) [inline]

This method assigns a sparse_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<string>.

Definition at line 1028 of file search_graph.cc.

10.24.3.419 basic_alltype& coco::coco::basic_alltype::set_csm (vmtl::sparse_matrix< std::string > * __m) [inline]

This method assigns a sparse_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<string>.

Definition at line 1028 of file expression.h.

10.24.3.420 basic_alltype& coco::coco::basic_alltype::set_csm (vmtl::sparse_matrix< std::string > * __m) [inline]

This method assigns a sparse_matrix<string>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<string>.

Definition at line 1028 of file search_graph.cc.

10.24.3.421 basic_alltype& coco::coco::basic_alltype::set_dm (vmtl::dense_matrix< double > * __m) [inline]

This method assigns a dense_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<double>.

Definition at line 940 of file search_graph.cc.

10.24.3.422 basic_alltype& coco::coco::basic_alltype::set_dm (vmtl::dense_matrix< double > * __m)

[inline]

This method assigns a dense_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<double>.

Definition at line 940 of file expression.h.

10.24.3.423 basic_alltype& coco::coco::basic_alltype::set_dm (vmtl::dense_matrix< double > * __m)

[inline]

This method assigns a dense_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<double>.

Definition at line 940 of file search_graph.cc.

10.24.3.424 basic_alltype& coco::coco::basic_alltype::set_dm (vmtl::dense_matrix< double > * __m)

[inline]

This method assigns a dense_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<double>.

Definition at line 940 of file search_graph.cc.

10.24.3.425 basic_alltype& coco::coco::basic_alltype::set_idm (vmtl::dense_matrix< interval > * __m)

[inline]

This method assigns a dense_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<interval>.

Definition at line 962 of file expression.h.

10.24.3.426 basic_alltype& coco::coco::basic_alltype::set_idm (vmtl::dense_matrix< interval > * __m)

[inline]

This method assigns a dense_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<interval>.

Definition at line 962 of file search_graph.cc.

10.24.3.427 basic_alltype& coco::coco::basic_alltype::set_idm (vmtl::dense_matrix< interval > * __m)

[inline]

This method assigns a dense_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<interval>.

Definition at line 962 of file search_graph.cc.

10.24.3.428 basic_alltype& coco::coco::basic_alltype::set_idm (vmtl::dense_matrix< interval > * __m)

[inline]

This method assigns a dense_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<interval>.

Definition at line 962 of file search_graph.cc.

10.24.3.429 basic_alltype& coco::coco::basic_alltype::set_ism (vmtl::sparse_matrix< interval > * __m)

[inline]

This method assigns a sparse_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<interval>.

Definition at line 1017 of file search_graph.cc.

10.24.3.430 basic_alltype& coco::coco::basic_alltype::set_ism (vmtl::sparse_matrix< interval > * __m)

[inline]

This method assigns a sparse_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<interval>.

Definition at line 1017 of file search_graph.cc.

10.24.3.431 basic_alltype& coco::coco::basic_alltype::set_ism (vmtl::sparse_matrix< interval > * __m)

[inline]

This method assigns a sparse_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<interval>.

Definition at line 1017 of file expression.h.

10.24.3.432 basic_alltype& coco::coco::basic_alltype::set_ism (vmtl::sparse_matrix< interval > * __m)

[inline]

This method assigns a sparse_matrix<interval>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<interval>.

Definition at line 1017 of file search_graph.cc.

10.24.3.433 basic_alltype& coco::coco::basic_alltype::set_ndm (vmtl::dense_matrix< int > * __m)

[inline]

This method assigns a dense_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<int>.

Definition at line 951 of file search_graph.cc.

10.24.3.434 basic_alltype& coco::coco::basic_alltype::set_ndm (vmtl::dense_matrix< int > * __m)

[inline]

This method assigns a dense_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<int>.

Definition at line 951 of file expression.h.

10.24.3.435 basic_alltype& coco::coco::basic_alltype::set_ndm (vmtl::dense_matrix< int > * __m)

[inline]

This method assigns a dense_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<int>.

Definition at line 951 of file search_graph.cc.

10.24.3.436 basic_alltype& coco::coco::basic_alltype::set_ndm (vmtl::dense_matrix< int > * __m)

[inline]

This method assigns a dense_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<int>.

Definition at line 951 of file search_graph.cc.

10.24.3.437 basic_alltype& coco::coco::basic_alltype::set_nsm (vmtl::sparse_matrix< int > * __m)

[inline]

This method assigns a sparse_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<int>.

Definition at line 1006 of file search_graph.cc.

10.24.3.438 basic_alltype& coco::coco::basic_alltype::set_nsm (vmtl::sparse_matrix< int > * __m)

[inline]

This method assigns a sparse_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<int>.

Definition at line 1006 of file expression.h.

10.24.3.439 basic_alltype& coco::coco::basic_alltype::set_nsm (vmtl::sparse_matrix< int > * __m)

[inline]

This method assigns a sparse_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<int>.

Definition at line 1006 of file search_graph.cc.

10.24.3.440 basic_alltype& coco::coco::basic_alltype::set_nsm (vmtl::sparse_matrix< int > * __m)

[inline]

This method assigns a sparse_matrix<int>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<int>.

Definition at line 1006 of file search_graph.cc.

10.24.3.441 basic_alltype& coco::coco::basic_alltype::set_sm (vmtl::sparse_matrix< double > * __m)

[inline]

This method assigns a sparse_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<double>.

Definition at line 995 of file search_graph.cc.

10.24.3.442 basic_alltype& coco::coco::basic_alltype::set_sm (vmtl::sparse_matrix< double > * __m)

[inline]

This method assigns a sparse_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<double>.

Definition at line 995 of file expression.h.

10.24.3.443 basic_alltype& coco::coco::basic_alltype::set_sm (vmtl::sparse_matrix< double > * __m) [inline]

This method assigns a sparse_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<double>.

Definition at line 995 of file search_graph.cc.

10.24.3.444 basic_alltype& coco::coco::basic_alltype::set_sm (vmtl::sparse_matrix< double > * __m) [inline]

This method assigns a sparse_matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<double>.

Definition at line 995 of file search_graph.cc.

10.24.3.445 basic_alltype& coco::coco::basic_alltype::set_xdm (vmtl::dense_matrix< num::Number > * __m) [inline]

This method assigns a dense_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<Number>.

Definition at line 984 of file search_graph.cc.

10.24.3.446 basic_alltype& coco::coco::basic_alltype::set_xdm (vmtl::dense_matrix< num::Number > * __m) [inline]

This method assigns a dense_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<Number>.

Definition at line 984 of file expression.h.

10.24.3.447 basic_alltype& coco::coco::basic_alltype::set_xdm (vmtl::dense_matrix< num::Number > * __m) [inline]

This method assigns a dense_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<Number>.

Definition at line 984 of file search_graph.cc.

10.24.3.448 basic_alltype& coco::coco::basic_alltype::set_xdm (vmtl::dense_matrix< num::Number > * __m) [inline]

This method assigns a dense_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new dense_matrix<Number>.

Definition at line 984 of file search_graph.cc.

10.24.3.449 basic_alltype& coco::coco::basic_alltype::set_xsm (vmtl::sparse_matrix< num::Number > * __m) [inline]

This method assigns a sparse_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<Number>.

Definition at line 1039 of file search_graph.cc.

10.24.3.450 basic_alltype& coco::coco::basic_alltype::set_xsm (vmtl::sparse_matrix< num::Number > * __m) [inline]

This method assigns a sparse_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<Number>.

Definition at line 1039 of file expression.h.

10.24.3.451 basic_alltype& coco::coco::basic_alltype::set_xsm (vmtl::sparse_matrix< num::Number > * __m) [inline]

This method assigns a sparse_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<Number>.

Definition at line 1039 of file search_graph.cc.

10.24.3.452 basic_alltype& coco::coco::basic_alltype::set_xsm (vmtl::sparse_matrix< num::Number > * __m) [inline]

This method assigns a sparse_matrix<Number>* to the basic alltype. This method must not be used unless __m was allocated by new sparse_matrix<Number>.

Definition at line 1039 of file search_graph.cc.

10.24.3.453 const vmtl::sparse_vector<interval>& coco::coco::basic_alltype::si ( ) const [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1276 of file search_graph.cc.

10.24.3.454 const vmtl::sparse_vector<interval>& coco::coco::basic_alltype::si ( ) const [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1276 of file expression.h.

10.24.3.455 const vmtl::sparse_vector<interval>& coco::coco::basic_alltype::si ( ) const [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1276 of file search_graph.cc.

10.24.3.456 const vmtl::sparse_vector<interval>& coco::coco::basic_alltype::si ( ) const [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1276 of file search_graph.cc.

10.24.3.457 vmtl::sparse_vector<interval>& coco::coco::basic_alltype::si ( ) [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1547 of file search_graph.cc.

10.24.3.458 vmtl::sparse_vector<interval>& coco::coco::basic_alltype::si ( ) [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1547 of file search_graph.cc.

10.24.3.459 vmtl::sparse_vector<interval>& coco::coco::basic_alltype::si ( ) [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1547 of file search_graph.cc.

10.24.3.460 vmtl::sparse_vector<interval>& coco::coco::basic_alltype::si ( ) [inline]

retrieve a sparse_vector<interval> from the basic_alltype

Definition at line 1547 of file expression.h.

10.24.3.461 const vmtl::sparse_matrix<double>& coco::coco::basic_alltype::sm ( ) const [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1305 of file search_graph.cc.

10.24.3.462 const vmtl::sparse_matrix<double>& coco::coco::basic_alltype::sm ( ) const [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1305 of file expression.h.

10.24.3.463 const vmtl::sparse_matrix<double>& coco::coco::basic_alltype::sm ( ) const [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1305 of file search_graph.cc.

10.24.3.464 const vmtl::sparse_matrix<double>& coco::coco::basic_alltype::sm ( ) const [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1305 of file search_graph.cc.

10.24.3.465 vmtl::sparse_matrix<double>& coco::coco::basic_alltype::sm ( ) [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1576 of file search_graph.cc.

10.24.3.466 vmtl::sparse_matrix<double>& coco::coco::basic_alltype::sm ( ) [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1576 of file search_graph.cc.

10.24.3.467 vmtl::sparse_matrix<double>& coco::coco::basic_alltype::sm ( ) [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1576 of file search_graph.cc.

10.24.3.468 vmtl::sparse_matrix<double>& coco::coco::basic_alltype::sm ( ) [inline]

retrieve a sparse_matrix<double> from the basic_alltype

Definition at line 1576 of file expression.h.

10.24.3.469 const vmtl::sparse_vector<int>& coco::coco::basic_alltype::sn ( ) const [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1228 of file search_graph.cc.

10.24.3.470 const vmtl::sparse_vector<int>& coco::coco::basic_alltype::sn ( ) const [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1228 of file search_graph.cc.

10.24.3.471 const vmtl::sparse_vector<int>& coco::coco::basic_alltype::sn ( ) const [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1228 of file expression.h.

10.24.3.472 const vmtl::sparse_vector<int>& coco::coco::basic_alltype::sn ( ) const [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1228 of file search_graph.cc.

10.24.3.473 vmtl::sparse_vector<int>& coco::coco::basic_alltype::sn ( ) [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1499 of file search_graph.cc.

10.24.3.474 vmtl::sparse_vector<int>& coco::coco::basic_alltype::sn ( ) [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1499 of file search_graph.cc.

10.24.3.475 vmtl::sparse_vector<int>& coco::coco::basic_alltype::sn ( ) [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1499 of file expression.h.

10.24.3.476 vmtl::sparse_vector<int>& coco::coco::basic_alltype::sn ( ) [inline]

retrieve a sparse_vector<int> from the basic_alltype

Definition at line 1499 of file search_graph.cc.

10.24.3.477 const vmtl::sparse_vector<unsigned int>& coco::coco::basic_alltype::su ( ) const [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1250 of file search_graph.cc.

10.24.3.478 const vmtl::sparse_vector<unsigned int>& coco::coco::basic_alltype::su ( ) const [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1250 of file expression.h.

10.24.3.479 const vmtl::sparse_vector<unsigned int>& coco::coco::basic_alltype::su ( ) const [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1250 of file search_graph.cc.

10.24.3.480 const vmtl::sparse_vector<unsigned int>& coco::coco::basic_alltype::su ( ) const [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1250 of file search_graph.cc.

10.24.3.481 vmtl::sparse_vector<unsigned int>& coco::coco::basic_alltype::su ( ) [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1521 of file search_graph.cc.

10.24.3.482 vmtl::sparse_vector<unsigned int>& coco::coco::basic_alltype::su ( ) [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1521 of file search_graph.cc.

10.24.3.483 vmtl::sparse_vector<unsigned int>& coco::coco::basic_alltype::su ( ) [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1521 of file search_graph.cc.

10.24.3.484 vmtl::sparse_vector<unsigned int>& coco::coco::basic_alltype::su ( ) [inline]

retrieve a sparse_vector<unsigned int> from the basic_alltype

Definition at line 1521 of file expression.h.

10.24.3.485 const vmtl::sparse_vector<num::Number>& coco::coco::basic_alltype::sx ( ) const [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1284 of file expression.h.

10.24.3.486 const vmtl::sparse_vector<num::Number>& coco::coco::basic_alltype::sx ( ) const [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1284 of file search_graph.cc.

10.24.3.487 const vmtl::sparse_vector<num::Number>& coco::coco::basic_alltype::sx ( ) const [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1284 of file search_graph.cc.

10.24.3.488 const vmtl::sparse_vector<num::Number>& coco::coco::basic_alltype::sx ( ) const [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1284 of file search_graph.cc.

10.24.3.489 vmtl::sparse_vector<num::Number>& coco::coco::basic_alltype::sx ( ) [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1555 of file expression.h.

10.24.3.490 vmtl::sparse_vector<num::Number>& coco::coco::basic_alltype::sx ( ) [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1555 of file search_graph.cc.

10.24.3.491 vmtl::sparse_vector<num::Number>& coco::coco::basic_alltype::sx ( ) [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1555 of file search_graph.cc.

10.24.3.492 vmtl::sparse_vector<num::Number>& coco::coco::basic_alltype::sx ( ) [inline]

retrieve a sparse_vector<Number> from the basic_alltype

Definition at line 1555 of file search_graph.cc.

10.24.3.493 const char* coco::coco::basic_alltype::type_cstr ( ) const [inline]

Return the type of the value contained in the storage as a C-string

Definition at line 1674 of file search_graph.cc.

10.24.3.494 const char* coco::coco::basic_alltype::type_cstr ( ) const [inline]

Return the type of the value contained in the storage as a C-string

Definition at line 1674 of file search_graph.cc.

10.24.3.495 const char* coco::coco::basic_alltype::type_cstr ( ) const [inline]

Return the type of the value contained in the storage as a C-string

Definition at line 1674 of file search_graph.cc.

10.24.3.496 const char* coco::coco::basic_alltype::type_cstr ( ) const [inline]

Return the type of the value contained in the storage as a C-string

Definition at line 1674 of file expression.h.

10.24.3.497 std::string coco::coco::basic_alltype::type_name ( ) const [inline]

Return the type of the value contained in the storage as a C++-string

Definition at line 1671 of file search_graph.cc.

10.24.3.498 std::string coco::coco::basic_alltype::type_name ( ) const [inline]

Return the type of the value contained in the storage as a C++-string

Definition at line 1671 of file expression.h.

10.24.3.499 std::string coco::coco::basic_alltype::type_name ( ) const [inline]

Return the type of the value contained in the storage as a C++-string

Definition at line 1671 of file search_graph.cc.

10.24.3.500 std::string coco::coco::basic_alltype::type_name ( ) const [inline]

Return the type of the value contained in the storage as a C++-string

Definition at line 1671 of file search_graph.cc.

10.24.3.501 const std::vector<unsigned int>& coco::coco::basic_alltype::u ( ) const [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1111 of file search_graph.cc.

10.24.3.502 const std::vector<unsigned int>& coco::coco::basic_alltype::u ( ) const [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1111 of file search_graph.cc.

10.24.3.503 const std::vector<unsigned int>& coco::coco::basic_alltype::u ( ) const [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1111 of file search_graph.cc.

10.24.3.504 const std::vector<unsigned int>& coco::coco::basic_alltype::u ( ) const [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1111 of file expression.h.

10.24.3.505 std::vector<unsigned int>& coco::coco::basic_alltype::u ( ) [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1383 of file search_graph.cc.

10.24.3.506 std::vector<unsigned int>& coco::coco::basic_alltype::u ( ) [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1383 of file expression.h.

10.24.3.507 std::vector<unsigned int>& coco::coco::basic_alltype::u ( ) [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1383 of file search_graph.cc.

10.24.3.508 std::vector<unsigned int>& coco::coco::basic_alltype::u ( ) [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 1383 of file search_graph.cc.

10.24.3.509 const std::vector<std::vector<bool> >& coco::coco::basic_alltype::vb ( ) const [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1150 of file search_graph.cc.

10.24.3.510 const std::vector<std::vector<bool> >& coco::coco::basic_alltype::vb ( ) const [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1150 of file search_graph.cc.

10.24.3.511 const std::vector<std::vector<bool> >& coco::coco::basic_alltype::vb ( ) const [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1150 of file expression.h.

10.24.3.512 const std::vector<std::vector<bool> >& coco::coco::basic_alltype::vb ( ) const [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1150 of file search_graph.cc.

10.24.3.513 std::vector<std::vector<bool> >& coco::coco::basic_alltype::vb ( ) [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1422 of file search_graph.cc.

10.24.3.514 std::vector<std::vector<bool> >& coco::coco::basic_alltype::vb ( ) [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1422 of file expression.h.

10.24.3.515 std::vector<std::vector<bool> >& coco::coco::basic_alltype::vb ( ) [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1422 of file search_graph.cc.

10.24.3.516 std::vector<std::vector<bool> >& coco::coco::basic_alltype::vb ( ) [inline]

retrieve a vector<vector<bool> > from the basic_alltype

Definition at line 1422 of file search_graph.cc.

10.24.3.517 const std::vector<std::vector<std::string> >& coco::coco::basic_alltype::vc ( ) const [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1214 of file search_graph.cc.

10.24.3.518 const std::vector<std::vector<std::string> >& coco::coco::basic_alltype::vc ( ) const [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1214 of file search_graph.cc.

10.24.3.519 const std::vector<std::vector<std::string> >& coco::coco::basic_alltype::vc ( ) const [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1214 of file search_graph.cc.

10.24.3.520 const std::vector<std::vector<std::string> >& coco::coco::basic_alltype::vc ( ) const [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1214 of file expression.h.

10.24.3.521 std::vector<std::vector<std::string> >& coco::coco::basic_alltype::vc ( ) [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1486 of file search_graph.cc.

10.24.3.522 std::vector<std::vector<std::string> >& coco::coco::basic_alltype::vc ( ) [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1486 of file search_graph.cc.

10.24.3.523 std::vector<std::vector<std::string> >& coco::coco::basic_alltype::vc ( ) [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1486 of file expression.h.

10.24.3.524 std::vector<std::vector<std::string> >& coco::coco::basic_alltype::vc ( ) [inline]

retrieve a vector<vector<string> > from the basic_alltype

Definition at line 1486 of file search_graph.cc.

10.24.3.525 const std::vector<std::vector<double> >& coco::coco::basic_alltype::vd ( ) const [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1206 of file expression.h.

10.24.3.526 const std::vector<std::vector<double> >& coco::coco::basic_alltype::vd ( ) const [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1206 of file search_graph.cc.

10.24.3.527 const std::vector<std::vector<double> >& coco::coco::basic_alltype::vd ( ) const [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1206 of file search_graph.cc.

10.24.3.528 const std::vector<std::vector<double> >& coco::coco::basic_alltype::vd ( ) const [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1206 of file search_graph.cc.

10.24.3.529 std::vector<std::vector<double> >& coco::coco::basic_alltype::vd ( ) [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1478 of file expression.h.

10.24.3.530 std::vector<std::vector<double> >& coco::coco::basic_alltype::vd ( ) [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1478 of file search_graph.cc.

10.24.3.531 std::vector<std::vector<double> >& coco::coco::basic_alltype::vd ( ) [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1478 of file search_graph.cc.

10.24.3.532 std::vector<std::vector<double> >& coco::coco::basic_alltype::vd ( ) [inline]

retrieve a vector<vector<double> > from the basic_alltype

Definition at line 1478 of file search_graph.cc.

10.24.3.533 const std::vector<std::vector<interval> >& coco::coco::basic_alltype::vi ( ) const [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1210 of file search_graph.cc.

10.24.3.534 const std::vector<std::vector<interval> >& coco::coco::basic_alltype::vi ( ) const [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1210 of file search_graph.cc.

10.24.3.535 const std::vector<std::vector<interval> >& coco::coco::basic_alltype::vi ( ) const [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1210 of file expression.h.

10.24.3.536 const std::vector<std::vector<interval> >& coco::coco::basic_alltype::vi ( ) const [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1210 of file search_graph.cc.

10.24.3.537 std::vector<std::vector<interval> >& coco::coco::basic_alltype::vi ( ) [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1482 of file search_graph.cc.

10.24.3.538 std::vector<std::vector<interval> >& coco::coco::basic_alltype::vi ( ) [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1482 of file search_graph.cc.

10.24.3.539 std::vector<std::vector<interval> >& coco::coco::basic_alltype::vi ( ) [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1482 of file expression.h.

10.24.3.540 std::vector<std::vector<interval> >& coco::coco::basic_alltype::vi ( ) [inline]

retrieve a vector<vector<interval> > from the basic_alltype

Definition at line 1482 of file search_graph.cc.

10.24.3.541 const std::vector<std::vector<int> >& coco::coco::basic_alltype::vn ( ) const [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1154 of file search_graph.cc.

10.24.3.542 const std::vector<std::vector<int> >& coco::coco::basic_alltype::vn ( ) const [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1154 of file expression.h.

10.24.3.543 const std::vector<std::vector<int> >& coco::coco::basic_alltype::vn ( ) const [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1154 of file search_graph.cc.

10.24.3.544 const std::vector<std::vector<int> >& coco::coco::basic_alltype::vn ( ) const [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1154 of file search_graph.cc.

10.24.3.545 std::vector<std::vector<int> >& coco::coco::basic_alltype::vn ( ) [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1426 of file search_graph.cc.

10.24.3.546 std::vector<std::vector<int> >& coco::coco::basic_alltype::vn ( ) [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1426 of file search_graph.cc.

10.24.3.547 std::vector<std::vector<int> >& coco::coco::basic_alltype::vn ( ) [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1426 of file expression.h.

10.24.3.548 std::vector<std::vector<int> >& coco::coco::basic_alltype::vn ( ) [inline]

retrieve a vector<vector<int> > from the basic_alltype

Definition at line 1426 of file search_graph.cc.

10.24.3.549 const std::vector<std::vector<unsigned int> >& coco::coco::basic_alltype::vu ( ) const [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1180 of file search_graph.cc.

10.24.3.550 const std::vector<std::vector<unsigned int> >& coco::coco::basic_alltype::vu ( ) const [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1180 of file expression.h.

10.24.3.551 const std::vector<std::vector<unsigned int> >& coco::coco::basic_alltype::vu ( ) const [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1180 of file search_graph.cc.

10.24.3.552 const std::vector<std::vector<unsigned int> >& coco::coco::basic_alltype::vu ( ) const [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1180 of file search_graph.cc.

10.24.3.553 std::vector<std::vector<unsigned int> >& coco::coco::basic_alltype::vu ( ) [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1452 of file search_graph.cc.

10.24.3.554 std::vector<std::vector<unsigned int> >& coco::coco::basic_alltype::vu ( ) [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1452 of file search_graph.cc.

10.24.3.555 std::vector<std::vector<unsigned int> >& coco::coco::basic_alltype::vu ( ) [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1452 of file expression.h.

10.24.3.556 std::vector<std::vector<unsigned int> >& coco::coco::basic_alltype::vu ( ) [inline]

retrieve a vector<vector<unsigned int> > from the basic_alltype

Definition at line 1452 of file search_graph.cc.

10.24.3.557 const std::vector<std::vector<num::Number> >& coco::coco::basic_alltype::vx ( ) const [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1218 of file expression.h.

10.24.3.558 const std::vector<std::vector<num::Number> >& coco::coco::basic_alltype::vx ( ) const [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1218 of file search_graph.cc.

10.24.3.559 const std::vector<std::vector<num::Number> >& coco::coco::basic_alltype::vx ( ) const [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1218 of file search_graph.cc.

10.24.3.560 const std::vector<std::vector<num::Number> >& coco::coco::basic_alltype::vx ( ) const [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1218 of file search_graph.cc.

10.24.3.561 std::vector<std::vector<num::Number> >& coco::coco::basic_alltype::vx ( ) [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1490 of file expression.h.

10.24.3.562 std::vector<std::vector<num::Number> >& coco::coco::basic_alltype::vx ( ) [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1490 of file search_graph.cc.

10.24.3.563 std::vector<std::vector<num::Number> >& coco::coco::basic_alltype::vx ( ) [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1490 of file search_graph.cc.

10.24.3.564 std::vector<std::vector<num::Number> >& coco::coco::basic_alltype::vx ( ) [inline]

retrieve a vector<vector<Number> > from the basic_alltype

Definition at line 1490 of file search_graph.cc.

10.24.3.565 const std::vector<num::Number>& coco::coco::basic_alltype::x ( ) const [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1145 of file search_graph.cc.

10.24.3.566 const std::vector<num::Number>& coco::coco::basic_alltype::x ( ) const [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1145 of file search_graph.cc.

10.24.3.567 const std::vector<num::Number>& coco::coco::basic_alltype::x ( ) const [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1145 of file expression.h.

10.24.3.568 const std::vector<num::Number>& coco::coco::basic_alltype::x ( ) const [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1145 of file search_graph.cc.

10.24.3.569 std::vector<num::Number>& coco::coco::basic_alltype::x ( ) [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1417 of file search_graph.cc.

10.24.3.570 std::vector<num::Number>& coco::coco::basic_alltype::x ( ) [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1417 of file search_graph.cc.

10.24.3.571 std::vector<num::Number>& coco::coco::basic_alltype::x ( ) [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1417 of file search_graph.cc.

10.24.3.572 std::vector<num::Number>& coco::coco::basic_alltype::x ( ) [inline]

retrieve a vector<Number> from the basic_alltype

Definition at line 1417 of file expression.h.

10.24.3.573 const vmtl::dense_matrix<num::Number>& coco::coco::basic_alltype::xdm ( ) const [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1301 of file search_graph.cc.

10.24.3.574 const vmtl::dense_matrix<num::Number>& coco::coco::basic_alltype::xdm ( ) const [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1301 of file search_graph.cc.

10.24.3.575 const vmtl::dense_matrix<num::Number>& coco::coco::basic_alltype::xdm ( ) const [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1301 of file search_graph.cc.

10.24.3.576 const vmtl::dense_matrix<num::Number>& coco::coco::basic_alltype::xdm ( ) const [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1301 of file expression.h.

10.24.3.577 vmtl::dense_matrix<num::Number>& coco::coco::basic_alltype::xdm ( ) [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1572 of file search_graph.cc.

10.24.3.578 vmtl::dense_matrix<num::Number>& coco::coco::basic_alltype::xdm ( ) [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1572 of file expression.h.

10.24.3.579 vmtl::dense_matrix<num::Number>& coco::coco::basic_alltype::xdm ( ) [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1572 of file search_graph.cc.

10.24.3.580 vmtl::dense_matrix<num::Number>& coco::coco::basic_alltype::xdm ( ) [inline]

retrieve a dense_matrix<Number> from the basic_alltype

Definition at line 1572 of file search_graph.cc.

10.24.3.581 const vmtl::sparse_matrix<num::Number>& coco::coco::basic_alltype::xsm ( ) const [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1317 of file expression.h.

10.24.3.582 const vmtl::sparse_matrix<num::Number>& coco::coco::basic_alltype::xsm ( ) const [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1317 of file search_graph.cc.

10.24.3.583 const vmtl::sparse_matrix<num::Number>& coco::coco::basic_alltype::xsm ( ) const [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1317 of file search_graph.cc.

10.24.3.584 const vmtl::sparse_matrix<num::Number>& coco::coco::basic_alltype::xsm ( ) const [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1317 of file search_graph.cc.

10.24.3.585 vmtl::sparse_matrix<num::Number>& coco::coco::basic_alltype::xsm ( ) [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1588 of file expression.h.

10.24.3.586 vmtl::sparse_matrix<num::Number>& coco::coco::basic_alltype::xsm ( ) [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1588 of file search_graph.cc.

10.24.3.587 vmtl::sparse_matrix<num::Number>& coco::coco::basic_alltype::xsm ( ) [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1588 of file search_graph.cc.

10.24.3.588 vmtl::sparse_matrix<num::Number>& coco::coco::basic_alltype::xsm ( ) [inline]

retrieve a sparse_matrix<Number> from the basic_alltype

Definition at line 1588 of file search_graph.cc.

10.24.3.589 const num::Number& coco::coco::basic_alltype::y ( ) const [inline]

retrieve a Number from the basic_alltype

Definition at line 1082 of file search_graph.cc.

10.24.3.590 const num::Number& coco::coco::basic_alltype::y ( ) const [inline]

retrieve a Number from the basic_alltype

Definition at line 1082 of file search_graph.cc.

10.24.3.591 const num::Number& coco::coco::basic_alltype::y ( ) const [inline]

retrieve a Number from the basic_alltype

Definition at line 1082 of file expression.h.

10.24.3.592 const num::Number& coco::coco::basic_alltype::y ( ) const [inline]

retrieve a Number from the basic_alltype

Definition at line 1082 of file search_graph.cc.

10.24.3.593 num::Number& coco::coco::basic_alltype::y ( ) [inline]

retrieve a Number from the basic_alltype

Definition at line 1354 of file search_graph.cc.

10.24.3.594 num::Number& coco::coco::basic_alltype::y ( ) [inline]

retrieve a Number from the basic_alltype

Definition at line 1354 of file search_graph.cc.

10.24.3.595 num::Number& coco::coco::basic_alltype::y ( ) [inline]

retrieve a Number from the basic_alltype

Definition at line 1354 of file search_graph.cc.

10.24.3.596 num::Number& coco::coco::basic_alltype::y ( ) [inline]

retrieve a Number from the basic_alltype

Definition at line 1354 of file expression.h.

10.24.4 Member Data Documentation

10.24.4.1 std::vector<bool>* coco::coco::basic_alltype::b

Definition at line 191 of file expression.h.

10.24.4.2 std::vector<std::string>* coco::coco::basic_alltype::c

Definition at line 196 of file expression.h.

10.24.4.3 vmtl::dense_matrix<std::string>* coco::coco::basic_alltype::cdm

Definition at line 215 of file expression.h.

10.24.4.4 vmtl::sparse_matrix<std::string>* coco::coco::basic_alltype::csm

Definition at line 220 of file expression.h.

10.24.4.5 std::vector<double>* coco::coco::basic_alltype::d

Definition at line 194 of file expression.h.

10.24.4.6 vmtl::dense_matrix<double>* coco::coco::basic_alltype::dm

Definition at line 212 of file expression.h.

10.24.4.7 std::vector<interval>* coco::coco::basic_alltype::i

Definition at line 195 of file expression.h.

10.24.4.8 vmtl::dense_matrix<interval>* coco::coco::basic_alltype::idm

Definition at line 214 of file expression.h.

10.24.4.9 vmtl::sparse_matrix<interval>* coco::coco::basic_alltype::ism

Definition at line 219 of file expression.h.

10.24.4.10 std::vector<int>* coco::coco::basic_alltype::n

Definition at line 192 of file expression.h.

10.24.4.11 bool coco::coco::basic_alltype::nb

Definition at line 183 of file expression.h.

10.24.4.12 double coco::coco::basic_alltype::nd

Definition at line 186 of file expression.h.

10.24.4.13 vmtl::dense_matrix<int>* coco::coco::basic_alltype::ndm

Definition at line 213 of file expression.h.

10.24.4.14 interval_st coco::coco::basic_alltype::ni

Definition at line 187 of file expression.h.

10.24.4.15 int coco::coco::basic_alltype::nn

Definition at line 184 of file expression.h.

10.24.4.16 vmtl::sparse_matrix<int>* coco::coco::basic_alltype::nsm

Definition at line 218 of file expression.h.

10.24.4.17 unsigned int coco::coco::basic_alltype::nu

Definition at line 185 of file expression.h.

10.24.4.18 void* coco::coco::basic_alltype::p

Definition at line 188 of file expression.h.

10.24.4.19 std::string* coco::coco::basic_alltype::s

Definition at line 189 of file expression.h.

10.24.4.20 vmtl::sparse_vector<bool>* coco::coco::basic_alltype::sb

Definition at line 205 of file expression.h.

10.24.4.21 vmtl::sparse_vector<std::string>* coco::coco::basic_alltype::sc

Definition at line 210 of file expression.h.

10.24.4.22 vmtl::sparse_vector<double>* coco::coco::basic_alltype::sd

Definition at line 208 of file expression.h.

10.24.4.23 vmtl::sparse_vector<interval>* coco::coco::basic_alltype::si

Definition at line 209 of file expression.h.

10.24.4.24 vmtl::sparse_matrix<double>* coco::coco::basic_alltype::sm

Definition at line 217 of file expression.h.

10.24.4.25 vmtl::sparse_vector<int>* coco::coco::basic_alltype::sn

Definition at line 206 of file expression.h.

10.24.4.26 vmtl::sparse_vector<unsigned int>* coco::coco::basic_alltype::su

Definition at line 207 of file expression.h.

10.24.4.27 vmtl::sparse_vector<num::Number>* coco::coco::basic_alltype::sx

Definition at line 211 of file expression.h.

10.24.4.28 std::vector<unsigned int>* coco::coco::basic_alltype::u

Definition at line 193 of file expression.h.

10.24.4.29 std::vector<std::vector<bool> >* coco::coco::basic_alltype::vb

Definition at line 198 of file expression.h.

10.24.4.30 std::vector<std::vector<std::string> >* coco::coco::basic_alltype::vc

Definition at line 203 of file expression.h.

10.24.4.31 std::vector<std::vector<double> >* coco::coco::basic_alltype::vd

Definition at line 201 of file expression.h.

10.24.4.32 std::vector<std::vector<interval> >* coco::coco::basic_alltype::vi

Definition at line 202 of file expression.h.

10.24.4.33 std::vector<std::vector<int> >* coco::coco::basic_alltype::vn

Definition at line 199 of file expression.h.

10.24.4.34 std::vector<std::vector<unsigned int> >* coco::coco::basic_alltype::vu

Definition at line 200 of file expression.h.

10.24.4.35 std::vector<std::vector<num::Number> >* coco::coco::basic_alltype::vx

Definition at line 204 of file expression.h.

10.24.4.36 std::vector<num::Number>* coco::coco::basic_alltype::x

Definition at line 197 of file expression.h.

10.24.4.37 vmtl::dense_matrix<num::Number>* coco::coco::basic_alltype::xdm

Definition at line 216 of file expression.h.

10.24.4.38 vmtl::sparse_matrix<num::Number>* coco::coco::basic_alltype::xsm

Definition at line 221 of file expression.h.

10.24.4.39 num::Number* coco::coco::basic_alltype::y

Definition at line 190 of file expression.h.

The documentation for this class was generated from the following files:

- basic_alltype.h
- basic_alltype.cc

10.25 coco::bbfunc Class Reference

The bbfunc class (black box function)

`#include <bbfunc.h>`

Public Member Functions

- bbfunc ()
- virtual ~bbfunc ()
- virtual const char * func_name () const =0
- virtual uint64_t thash ()=0
- virtual uint64_t chash ()=0
- virtual std::string dag_out () const
- virtual std::string print_params () const
- virtual double eval (const std::vector< double > &x) const =0
- virtual interval eval (const std::vector< interval > &x) const =0
- virtual interval eval_slp (const std::vector< interval > &z, const std::vector< interval > &x, unsigned int i) const =0
- virtual interval eval_slp2 (const std::vector< interval > &z, const std::vector< interval > &y, const std::vector< interval > &x, unsigned int i, unsigned int j) const =0
- virtual interval eval_slp2 (const std::vector< interval > &z, const std::vector< interval > &x, unsigned int i, unsigned int j) const =0
- virtual interval_set intersect_inv (const std::vector< interval > &x, const interval &r, unsigned int i) const =0
- virtual interval_set intersect_inv (const std::vector< interval_set > &x, const interval_set &r, unsigned int i) const
- virtual std::pair< double, double > evald (const std::vector< double > &x, unsigned int i) const =0
- virtual std::triple< double, double, double > evalt (const std::vector< double > &x, unsigned int i, unsigned int j) const =0
- virtual void evalf (const std::vector< double > &x, double *r, unsigned int i, unsigned int j, unsigned int k) const =0
- virtual std::pair< interval, interval > evald (const std::vector< interval > &x, unsigned int i) const =0
- virtual std::triple< interval, interval, interval > evalt (const std::vector< interval > &x, unsigned int i, unsigned int j) const =0
- virtual void evalf (const std::vector< interval > &x, interval *r, unsigned int i, unsigned int j, unsigned int k) const =0
- virtual convex_info convexity (const std::vector< interval > &x) const =0
- virtual int differentiability (const std::vector< interval > &x) const =0
- virtual unsigned int supported_eval_order () const =0
- virtual bool operator== (const bbfunc &b) const
- virtual bool operator!= (const bbfunc &b) const
- virtual bbfunc * differentiate () const =0

10.25.1 Detailed Description

The bbfunc class is the base class for black box functions. It is an abstract class for various types of black box functions.

10.25.2 Constructor & Destructor Documentation

10.25.2.1 coco::bbfunc::bbfunc ( ) [inline]

Default Constructor

Definition at line 57 of file bbfunc.h.

10.25.2.2 virtual coco::bbfunc::~bbfunc ( ) [inline, virtual]

Standard Destructor

Definition at line 60 of file bbfunc.h.

10.25.3 Member Function Documentation

10.25.3.1 virtual uint64_t coco::bbfunc::chash ( ) [pure virtual]

the const hash value for hash calculation in the DAG

10.25.3.2 virtual convex_info coco::bbfunc::convexity (const std::vector< interval > & *x*) const [pure virtual]

return convexity information on `i`.

10.25.3.3 virtual std::string coco::bbfunc::dag_out ( ) const [inline, virtual]

the parameters written in DAG format (it must contain necessary ':'s)

Definition at line 73 of file bbfunc.h.

10.25.3.4 virtual int coco::bbfunc::differentiability (const std::vector< interval > & *x*) const [pure virtual]

return differentiability information on `i`.

10.25.3.5 virtual bbfunc* coco::bbfunc::differentiate ( ) const [pure virtual]

differentiate symbolically

10.25.3.6 virtual double coco::bbfunc::eval (const std::vector< double > & *x*) const [pure virtual]

the point evaluation

10.25.3.7 virtual interval coco::bbfunc::eval (const std::vector< interval > & *x*) const [pure virtual]

the range evaluation

10.25.3.8 virtual interval coco::bbfunc::eval_slp (const std::vector< interval > & *z*, const std::vector< interval > & *x*, unsigned int *i*) const [pure virtual]

the interval first order slope evaluation at center z

10.25.3.9 virtual interval coco::bbfunc::eval_slp2 (const std::vector< interval > & *z*, const std::vector< interval > & *y*, const std::vector< interval > & *x*, unsigned int *i*, unsigned int *j*) const [pure virtual]

the interval second order slope evaluation at centers z and y

10.25.3.10 virtual interval coco::bbfunc::eval_slp2 (const std::vector< interval > & *z*, const std::vector< interval > & *x*, unsigned int *i*, unsigned int *j*) const [pure virtual]

the interval second order slope evaluation at double center z

10.25.3.11 virtual std::pair<double,double> coco::bbfunc::evald (const std::vector< double > & *x*, unsigned int *i*) const [pure virtual]

the point and derivative evaluation

10.25.3.12 virtual std::pair<interval,interval> coco::bbfunc::evald (const std::vector< interval > & *x*, unsigned int *i*) const [pure virtual]

the point and derivative evaluation

10.25.3.13 virtual void coco::bbfunc::evalf (const std::vector< double > & *x*, double * *r*, unsigned int *i*, unsigned int *j*, unsigned int *k*) const [pure virtual]

the point, first to third derivative evaluation

10.25.3.14 virtual void coco::bbfunc::evalf (const std::vector< interval > & *x*, interval * *r*, unsigned int *i*, unsigned int *j*, unsigned int *k*) const [pure virtual]

the point, first to third derivative evaluation

10.25.3.15 virtual std::triple<double,double,double> coco::bbfunc::evalt (const std::vector< double > & *x*, unsigned int *i*, unsigned int *j*) const [pure virtual]

the point and first and second derivative evaluation

10.25.3.16 virtual std::triple<interval,interval,interval> coco::bbfunc::evalt (const std::vector< interval > & *x*, unsigned int *i*, unsigned int *j*) const [pure virtual]

the point and first and second derivative evaluation

10.25.3.17 virtual const char* coco::bbfunc::func_name ( ) const [pure virtual]

the name of the bbfunc.

10.25.3.18 virtual interval_set coco::bbfunc::intersect_inv (const std::vector< interval > & *x*, const interval & *r*, unsigned int *i*) const [pure virtual]

back propagation

10.25.3.19 virtual interval_set coco::bbfunc::intersect_inv (const std::vector< interval_set > & *x*, const interval_set & *r*, unsigned int *i*) const [virtual]

back propagation

10.25.3.20 virtual bool coco::bbfunc::operator!= (const bbfunc & *b*) const [inline, virtual]

the other comparison operator

Definition at line 134 of file bbfunc.h.

10.25.3.21 virtual bool coco::bbfunc::operator== (const bbfunc & *b*) const [inline, virtual]

the comparison operator

Definition at line 131 of file bbfunc.h.

10.25.3.22 virtual std::string coco::bbfunc::print_params ( ) const [inline, virtual]

the parameters written in printable form

Definition at line 76 of file bbfunc.h.

10.25.3.23 virtual unsigned int coco::bbfunc::supported_eval_order ( ) const [pure virtual]

return the maximal evaluation order supported.

10.25.3.24 virtual uint64_t coco::bbfunc::thash ( ) [pure virtual]

the pure hash value for hash calculation in the DAG

The documentation for this class was generated from the following file:

- bbfunc.h

10.26 coco::bbfunc_expr Class Reference

The bbfunc_expr class (black box function expression)

`#include <bbfunc_expr.h>`

Public Member Functions

- std::string func_name () const
- std::string dag_out () const
- std::string print_params () const

- bbfunc_expr (bbfunc *_d)
- ~bbfunc_expr ()
- double eval (std::vector< double > &x) const
- std::pair< double, double > evald (std::vector< double > &x, unsigned int i) const
- std::triple< double, double, double > evalt (std::vector< double > &x, unsigned int i, unsigned int j) const
- void evalf (std::vector< double > &x, double *r, unsigned int i, unsigned int j, unsigned int k) const
- interval eval (const std::vector< interval > &x) const
- std::pair< interval, interval > evald (std::vector< interval > &x, unsigned int i) const
- std::triple< interval, interval, interval > evalt (std::vector< interval > &x, unsigned int i, unsigned int j) const
- void evalf (std::vector< interval > &x, interval *r, unsigned int i, unsigned int j, unsigned int k) const
- interval_set eval (const std::vector< interval_set > &x) const
- interval eval_slp (const std::vector< interval > &z, const std::vector< interval > &x, unsigned int i) const
- interval eval_slp2 (const std::vector< interval > &z, const std::vector< interval > &y, const std::vector< interval > &x, unsigned int i, unsigned int j) const
- interval eval_slp2 (const std::vector< interval > &z, const std::vector< interval > &x, unsigned int i, unsigned int j) const
- interval_set intersect_inv (const std::vector< interval > &x, const interval &r, unsigned int i) const
- interval_set intersect_inv (const std::vector< interval_set > &x, const interval_set &r, unsigned int i) const
- unsigned int supported_eval_order () const
- bool operator== (const bbfunc_expr &b) const
- bool operator!= (const bbfunc_expr &b) const
- bbfunc_expr * differentiate (unsigned int order) const
- convex_e convex_update (const std::vector< convex_e > &c, const std::vector< interval > &x) const
- int differentiability (const std::vector< int > &diff, const std::vector< interval > &x) const

10.26.1 Detailed Description

The bbfunc_expr class is the class for black box function expressions. It wraps the bbfunc class with an additional parameter for the order of the derivative.

10.26.2 Constructor & Destructor Documentation

10.26.2.1 coco::bbfunc_expr::bbfunc_expr (bbfunc * _d) [inline]

constructor: note the _d should be allocated with new!

Definition at line 66 of file bbfunc_expr.h.

10.26.2.2 coco::bbfunc_expr::~bbfunc_expr ( ) [inline]

the destructor

Definition at line 69 of file bbfunc_expr.h.

10.26.3 Member Function Documentation

10.26.3.1 convex_e coco::bbfunc_expr::convex_update (const std::vector< convex_e > & *c*, const std::vector< interval > & *x*) const

update the convexity information for in argument convexity c on box x.

Definition at line 36 of file bbfunc_expr.cc.

10.26.3.2 std::string coco::bbfunc_expr::dag_out ( ) const [inline]

the parameters written in DAG format (it must contain necessary ':'s)

Definition at line 59 of file bbfunc_expr.h.

10.26.3.3 int coco::bbfunc_expr::differentiability (const std::vector< int > & *v_diff*, const std::vector< interval > & *x*) const

return differentiability information on box x.

Definition at line 93 of file bbfunc_expr.cc.

10.26.3.4 bbfunc_expr* coco::bbfunc_expr::differentiate (unsigned int *order*) const [inline]

differentiate a bbfunc_expr

Definition at line 137 of file bbfunc_expr.h.

10.26.3.5 double coco::bbfunc_expr::eval (std::vector< double > & *x*) const [inline]

the point evaluation

Definition at line 73 of file bbfunc_expr.h.

10.26.3.6 interval coco::bbfunc_expr::eval (const std::vector< interval > & *x*) const [inline]

the range evaluation

Definition at line 87 of file bbfunc_expr.h.

10.26.3.7 interval_set coco::bbfunc_expr::eval (const std::vector< interval_set > & *x*) const

the range evaluation

Definition at line 120 of file bbfunc_expr.cc.

10.26.3.8 interval coco::bbfunc_expr::eval_slp (const std::vector< interval > & *z*, const std::vector< interval > & *x*, unsigned int *i*) const [inline]

the interval first order slope evaluation at center z

Definition at line 104 of file bbfunc_expr.h.

10.26.3.9 interval coco::bbfunc_expr::eval_slp2 (const std::vector< interval > & z, const std::vector< interval > & y, const std::vector< interval > & x, unsigned int i, unsigned int j) const [inline]

the interval second order slope evaluation at centers z and y

Definition at line 108 of file bbfunc_expr.h.

10.26.3.10 interval coco::bbfunc_expr::eval_slp2 (const std::vector< interval > & z, const std::vector< interval > & x, unsigned int i, unsigned int j) const [inline]

the interval second order slope evaluation at double center z

Definition at line 113 of file bbfunc_expr.h.

10.26.3.11 std::pair<double,double> coco::bbfunc_expr::evald (std::vector< double > & x, unsigned int i) const [inline]

the point and first derivative evaluation

Definition at line 76 of file bbfunc_expr.h.

10.26.3.12 std::pair<interval,interval> coco::bbfunc_expr::evald (std::vector< interval > & x, unsigned int i) const [inline]

the point and first derivative evaluation

Definition at line 90 of file bbfunc_expr.h.

10.26.3.13 void coco::bbfunc_expr::evalf (std::vector< double > & x, double * r, unsigned int i, unsigned int j, unsigned int k) const [inline]

the point, and first to third derivative evaluation

Definition at line 83 of file bbfunc_expr.h.

10.26.3.14 void coco::bbfunc_expr::evalf (std::vector< interval > & x, interval * r, unsigned int i, unsigned int j, unsigned int k) const [inline]

the point, and first to third derivative evaluation

Definition at line 98 of file bbfunc_expr.h.

10.26.3.15 std::triple<double,double,double> coco::bbfunc_expr::evalt (std::vector< double > & x, unsigned int i, unsigned int j) const [inline]

the point, first and second derivative evaluation

Definition at line 79 of file bbfunc_expr.h.

10.26.3.16 std::triple<interval,interval,interval> coco::bbfunc_expr::evalt (std::vector< interval > & x, unsigned int i, unsigned int j) const [inline]

the point, first and second derivative evaluation

Definition at line 94 of file bbfunc_expr.h.

10.26.3.17 std::string coco::bbfunc_expr::func_name ( ) const [inline]

the name of the bbfunc.

Definition at line 56 of file bbfunc_expr.h.

10.26.3.18 interval_set coco::bbfunc_expr::intersect_inv (const std::vector< interval > & *x*, const interval & *r*, unsigned int *i*) const [inline]

Definition at line 116 of file bbfunc_expr.h.

10.26.3.19 interval_set coco::bbfunc_expr::intersect_inv (const std::vector< interval_set > & *x*, const interval_set & *r*, unsigned int *i*) const [inline]

back propagation

Definition at line 120 of file bbfunc_expr.h.

10.26.3.20 bool coco::bbfunc_expr::operator!= (const bbfunc_expr & *b*) const [inline]

the other comparison operator

Definition at line 133 of file bbfunc_expr.h.

10.26.3.21 bool coco::bbfunc_expr::operator== (const bbfunc_expr & *b*) const [inline]

the comparison operator

Definition at line 129 of file bbfunc_expr.h.

10.26.3.22 std::string coco::bbfunc_expr::print_params ( ) const [inline]

the parameters written in printable form

Definition at line 62 of file bbfunc_expr.h.

10.26.3.23 unsigned int coco::bbfunc_expr::supported_eval_order ( ) const [inline]

the maximal order of derivatives which can be evaluated.

Definition at line 125 of file bbfunc_expr.h.

The documentation for this class was generated from the following files:

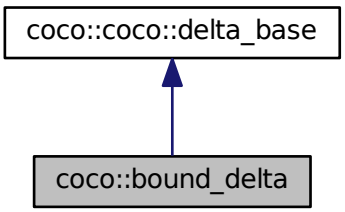
- bbfunc_expr.h
- bbfunc_expr.cc

10.27 coco::bound_delta Class Reference

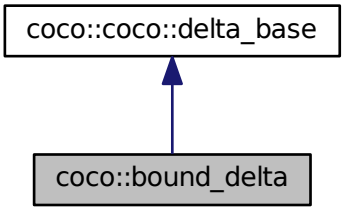
The bound delta class for changing the node bounds within a model.

`#include <bound_delta.h>`

Inheritance diagram for coco::bound_delta:



Collaboration diagram for coco::bound_delta:



Public Member Functions

- bound_delta (const std::vector< unsigned int > &__i, const std::vector< interval > &__b)
- bound_delta (unsigned int __i, interval __b)
- bound_delta (const bound_delta &__d)
- bound_delta * new_copy () const
- void destroy_copy (delta_base *__d) const
- bool apply (work_node &_x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const bound_delta &_c) const
- bool operator!= (const bound_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)

- const std::string & get_action () const
- const std::string & get_action () const
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual void unkeep ()
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Public Attributes

- std::vector< unsigned int > indices
- std::vector< interval > new_f_bounds

Protected Attributes

- std::string _action

10.27.1 Detailed Description

This class is used to specify information for changing the node bounds within a model.

10.27.2 Constructor & Destructor Documentation

10.27.2.1 coco::bound_delta::bound_delta (const std::vector< unsigned int > & __i, const std::vector< interval > & __b) [inline]

Constructor which explicitly assigns the indices ___i and the new bounds ___b.

Definition at line 109 of file bound_delta.h.

10.27.2.2 coco::bound_delta::bound_delta (unsigned int __i, interval __b) [inline]

Constructor which explicitly assigns one index ___i and the corresponding bound ___b.

Definition at line 116 of file bound_delta.h.

10.27.2.3 coco::bound_delta::bound_delta (const bound_delta & __d) [inline]

Standard Copy Constructor

Definition at line 121 of file bound_delta.h.

10.27.3 Member Function Documentation

10.27.3.1 bool coco::bound_delta::apply (work_node & _x, undelta_base *& _u, const delta_id & _did, size_t & delta_size) const

Apply the delta with delta_id _d to work node _x, hereby changing the bounds as requested.

Definition at line 34 of file bound_delta.cc.

10.27.3.2 virtual bool coco::coco::delta_base::apply (work_node & _x, undelta_base *& _u, const delta_id & _di, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.27.3.3 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.27.3.4 bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.27.3.5 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.27.3.6 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.27.3.7 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.27.3.8 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.27.3.9 void coco::bound_delta::destroy_copy (delta_base * __d) const [inline]

Clone Destructor

Definition at line 132 of file bound_delta.h.

10.27.3.10 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.27.3.11 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.27.3.12 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.27.3.13 delta coco::coco::delta_base::make_delta (const std::string & a) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.27.3.14 delta coco::coco::delta_base::make_delta (const std::string & a) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.27.3.15 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.27.3.16 bound_delta* coco::bound_delta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::delta_base.

Definition at line 130 of file bound_delta.h.

10.27.3.17 bool coco::bound_delta::operator!= (const delta_base & *_c*) const [inline]

Definition at line 143 of file bound_delta.h.

10.27.3.18 bool coco::bound_delta::operator!= (const bound_delta & *_c*) const [inline]

Definition at line 151 of file bound_delta.h.

10.27.3.19 bool coco::bound_delta::operator== (const delta_base & *_c*) const [inline]

Comparison operators

Definition at line 140 of file bound_delta.h.

10.27.3.20 bool coco::bound_delta::operator== (const bound_delta & *_c*) const [inline]

Comparison operators

Definition at line 148 of file bound_delta.h.

10.27.3.21 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.27.3.22 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.27.3.23 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.27.4 Member Data Documentation

10.27.4.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

10.27.4.2 std::vector<unsigned int> coco::bound_delta::indices

This variable holds the indices of the changed nodes. If **all** variables are being updated this vector is empty.

Definition at line 103 of file bound_delta.h.

10.27.4.3 std::vector<interval> coco::bound_delta::new_f_bounds

This variable holds the new bounds of the changed nodes.

Definition at line 105 of file bound_delta.h.

The documentation for this class was generated from the following files:

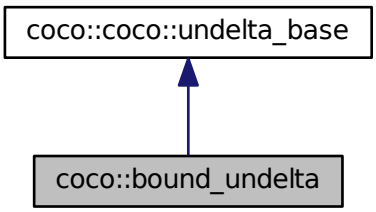
- bound_delta.h
- bound_delta.cc

10.28 coco::bound_undelta Class Reference

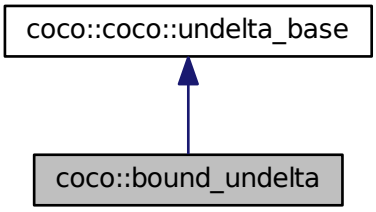
The bound undelta class for undoing changes to the node bounds in a model.

`#include <bound_delta.h>`

Inheritance diagram for coco::bound_undelta:



Collaboration diagram for coco::bound_undelta:



Public Member Functions

- bound_undelta (const std::vector< unsigned int > &__i, const std::vector< interval > &__b, double _og, double _olv)
- bound_undelta (const std::vector< unsigned int > &__i, double _og, double _olv)
- bound_undelta (const bound_undelta &__d)
- bound_undelta * new_copy () const
- void destroy_copy (undelta_base *__d) const
- bool unapply (work_node &_x, const delta_id &_did) const
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

Public Attributes

- std::vector< unsigned int > indices
- std::vector< interval > old_f_bounds
- double old_gain
- double old_log_vol

Friends

- class bound_delta

10.28.1 Detailed Description

This class is used to specify undo information for changes to the node bounds within a model. This class undoes the apply of a bound_delta.

10.28.2 Constructor & Destructor Documentation

10.28.2.1 coco::bound_undelta::bound_undelta (const std::vector< unsigned int > & __i, const std::vector< interval > & __b, double _og, double _olv) [inline]

Constructor, setting `indices` to `__i`, the `old_f_bounds` to `__b`, the `old_gain` to `_og`, and the `old_log_vol` to `_olv`.

Definition at line 56 of file bound_delta.h.

10.28.2.2 coco::bound_undelta::bound_undelta (const std::vector< unsigned int > & __i, double _og, double _olv) [inline]

Constructor, setting `indices` to `__i`, the `old_gain` to `_og`, the `old_log_vol` to `_olv`, and leaving `old_f_bounds` empty.

Definition at line 64 of file bound_delta.h.

10.28.2.3 coco::bound_undelta::bound_undelta (const bound_undelta & __d) [inline]

Standard Copy Constructor

Definition at line 70 of file bound_delta.h.

10.28.3 Member Function Documentation

10.28.3.1 void coco::bound_undelta::destroy_copy (undelta_base * __d) const [inline]

Clone Destructor

Definition at line 84 of file bound_delta.h.

10.28.3.2 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.28.3.3 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.28.3.4 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.28.3.5 bound_undelta* coco::bound_undelta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::undelta_base.

Definition at line 82 of file bound_delta.h.

10.28.3.6 bool coco::bound_undelta::unapply (work_node & _x, const delta_id & _did) const [virtual]

Undo the bound_delta with delta_id _i in work node _x

Reimplemented from coco::coco::undelta_base.

Definition at line 78 of file bound_delta.cc.

10.28.3.7 virtual bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.28.3.8 virtual bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.28.3.9 bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [inline, virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

Definition at line 94 of file api_delta.h.

10.28.4 Friends And Related Function Documentation

10.28.4.1 friend class bound_delta [friend]

Definition at line 89 of file bound_delta.h.

10.28.5 Member Data Documentation

10.28.5.1 std::vector<unsigned int> coco::bound_undelta::indices

This variable holds the indices of the changed nodes. If **all** variables are being updated this vector is empty.

Definition at line 46 of file bound_delta.h.

10.28.5.2 std::vector<interval> coco::bound_undelta::old_f_bounds

This variable holds the old bounds of the changed nodes.

Definition at line 49 of file bound_delta.h.

10.28.5.3 double coco::bound_undelta::old_gain

the gain in the work_node before the application

Definition at line 51 of file bound_delta.h.

10.28.5.4 double coco::bound_undelta::old_log_vol

the old log-Volume of the work_node

Definition at line 52 of file bound_delta.h.

The documentation for this class was generated from the following files:

- bound_delta.h
- bound_delta.cc

10.29 coco::box_check_intersection Class Reference

Stored procedure checking whether a box intersects the work_node's box.

`#include <dbtools.h>`

Public Types

- typedef work_node_context context
- typedef bool return_type

Public Member Functions

- box_check_intersection (vdbl::colid _x)
- box_check_intersection (const box_check_intersection &_i)
- virtual ~box_check_intersection ()
- bool operator() () const
- bool def () const
- void setcontext (const context *c, const vdbl::row *r)

10.29.1 Detailed Description

This class is used as a stored procedure (vdbl::method) for checking whether a box stored in the "boxes" table is disjunct with the search box in the evaluation context of a given work_node.

10.29.2 Member Typedef Documentation

10.29.2.1 typedef work_node_context coco::box_check_intersection::context

The evaluation context type

Definition at line 99 of file dbtools.h.

10.29.2.2 typedef bool coco::box_check_intersection::return_type

The return type of the operator(), i.e. the function class.

Definition at line 113 of file dbtools.h.

10.29.3 Constructor & Destructor Documentation

10.29.3.1 coco::box_check_intersection::box_check_intersection (vdbl::colid _x) [inline]

Constructor setting the relevant column id.

Definition at line 116 of file dbtools.h.

10.29.3.2 coco::box_check_intersection::box_check_intersection (const box_check_intersection & _i) [inline]

Standard Copy Constructor

Definition at line 119 of file dbtools.h.

10.29.3.3 virtual coco::box_check_intersection::~box_check_intersection ( ) [inline, virtual]

Standard Destructor

Definition at line 122 of file dbtools.h.

10.29.4 Member Function Documentation

10.29.4.1 bool coco::box_check_intersection::def ( ) const [inline]

Method, which computes the default value of the stored procedure

Definition at line 127 of file dbtools.h.

10.29.4.2 bool coco::box_check_intersection::operator() ( ) const

Evaluation operator, which computes the value of the stored procedure

10.29.4.3 void coco::box_check_intersection::setcontext (const context * c, const vdbl::row * r) [inline]

This method initializes the evaluation context and the row, preparing for the evaluation.

Definition at line 130 of file dbtools.h.

The documentation for this class was generated from the following file:

- dbtools.h

10.30 coco::coco::box_check_intersection Class Reference

Stored procedure checking whether a box intersects the work_node's box.

Public Types

- typedef work_node_context context
- typedef bool return_type

Public Member Functions

- box_check_intersection (vdbl::colid _x)
- box_check_intersection (const box_check_intersection &_i)
- virtual ~box_check_intersection ()
- bool operator() () const
- bool def () const
- void setcontext (const context *c, const vdbl::row *r)

10.30.1 Detailed Description

This class is used as a stored procedure (vdbl::method) for checking whether a box stored in the "boxes" table is disjunct with the search box in the evaluation context of a given work_node.

10.30.2 Member Typedef Documentation

10.30.2.1 typedef work_node_context coco::coco::box_check_intersection::context

The evaluation context type

Definition at line 99 of file search_graph.cc.

10.30.2.2 typedef bool coco::coco::box_check_intersection::return_type

The return type of the operator(), i.e. the function class.

Definition at line 113 of file search_graph.cc.

10.30.3 Constructor & Destructor Documentation

10.30.3.1 coco::coco::box_check_intersection::box_check_intersection (vdbl::colid _x) [inline]

Constructor setting the relevant column id.

Definition at line 116 of file search_graph.cc.

10.30.3.2 coco::coco::box_check_intersection::box_check_intersection (const box_check_intersection & _i) [inline]

Standard Copy Constructor

Definition at line 119 of file search_graph.cc.

10.30.3.3 virtual coco::coco::box_check_intersection::~box_check_intersection ( ) [inline, virtual]

Standard Destructor

Definition at line 122 of file search_graph.cc.

10.30.4 Member Function Documentation

10.30.4.1 bool coco::coco::box_check_intersection::def ( ) const [inline]

Method, which computes the default value of the stored procedure

Definition at line 127 of file search_graph.cc.

10.30.4.2 bool coco::box_check_intersection::operator() ( ) const

Evaluation operator, which computes the value of the stored procedure

Definition at line 76 of file dbtools.cc.

10.30.4.3 void coco::coco::box_check_intersection::setcontext (const context * c, const vdbl::row * r) [inline]

This method initializes the evaluation context and the row, preparing for the evaluation.

Definition at line 130 of file search_graph.cc.

The documentation for this class was generated from the following files:

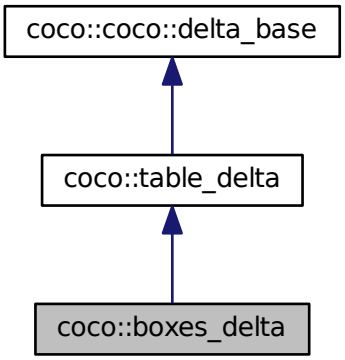
- dbtools.h
- dbtools.cc

10.31 coco::boxes_delta Class Reference

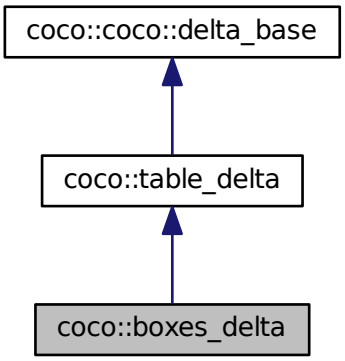
A delta class which adds new boxes to the search database.

`#include <boxes_delta.h>`

Inheritance diagram for coco::boxes_delta:



Collaboration diagram for coco::boxes_delta:



Public Types

- typedef std::pair< std::string, dbt_row > t_line
- typedef std::vector< t_line > t_ctr

Public Member Functions

- boxes_delta (bool _add=true)
- boxes_delta (const dbt_row &_b, bool _add=true)
- boxes_delta (const std::vector< dbt_row > &_b, bool _add=true)
- boxes_delta (const boxes_delta &__d)
- boxes_delta * new_copy () const
- void destroy_copy (delta_base *__d) const
- void create_table (work_node &_x, vdbl::standard_table *&ptb, const std::string &__t) const
- bool apply (work_node &_x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const boxes_delta &_c) const
- bool operator!= (const boxes_delta &_c) const
- void add (const t_line &_tl)
- void add (const std::string &_tn, const dbt_row &_r)
- void add (const std::vector< t_line > &_tlv)
- void rm (const annotation &_tr)
- void rm (const std::vector< annotation > &_trv)
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- void convert (work_node &_x, delta_base *&_u)
- virtual void convert (work_node &_x, delta_base *&_d)
- bool operator== (const table_delta &_c) const
- bool operator!= (const table_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- const std::string & get_action () const
- const std::string & get_action () const
- virtual void unkeep ()
- virtual void unkeep ()
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Protected Attributes

- std::string _action

10.31.1 Detailed Description

A specialized delta class which adds new boxes, e.g. exclusion boxes, to the search database. It is a subclass of table_delta and is, after storing the information in the search database, converted to an annotation_delta. Therefore, no `boxes_undelta` class is needed.

10.31.2 Member Typedef Documentation

10.31.2.1 typedef std::vector<t_line> coco::table_delta::t_ctr [inherited]

A variable of this type holds the new table entries.

Definition at line 49 of file table_delta.h.

10.31.2.2 typedef std::pair<std::string,dbt_row> coco::table_delta::t_line [inherited]

This type specifies one row in one table.

Definition at line 47 of file table_delta.h.

10.31.3 Constructor & Destructor Documentation

10.31.3.1 coco::boxes_delta::boxes_delta (bool _add = true) [inline]

Standard Constructor: if _add is false, the box is interpreted as an exclusion box. Otherwise it is stored in the table of boxes.

Definition at line 53 of file boxes_delta.h.

10.31.3.2 coco::boxes_delta::boxes_delta (const dbt_row & _b, bool _add = true) [inline]

Constructor: the box is described, including all additional information as dbt_row in the database format, if _add is false, the box is interpreted as an exclusion box. It is stored in the table box of boxes. The columns are

Name	Type	R/O	
x	vector<interval>	required	the box
exclusion box	bool	optional	default: false
contains optimum	bool	optional	default: false
eval region	bool	optional	default: false

Definition at line 67 of file boxes_delta.h.

10.31.3.3 coco::boxes_delta::boxes_delta (const std::vector< dbt_row > & _b, bool _add = true) [inline]

Constructor: the boxes are described, including all additional information as dbt_row in the database format. The parameter _b is a vector of boxes to be added. If _add is false, the boxes are interpreted as an exclusion boxes. They are stored in the table box of boxes. The columns are

Name	Type	R/O	Description	Default
x	vector<interval>	required	the box	
exclusion box	bool	optional	an exclusion box?	false
contains optimum	bool	optional	contains the optimum?	false
eval region	bool	optional	an eval region?	false

Definition at line 84 of file boxes_delta.h.

10.31.3.4 coco::boxes_delta::boxes_delta (const boxes_delta & __d) [inline]

Standard Copy Constructor

Definition at line 89 of file boxes_delta.h.

10.31.4 Member Function Documentation

10.31.4.1 void coco::table_delta::add (const t_line & _tl) [inline, inherited]

This method adds one table row.

Definition at line 95 of file table_delta.h.

10.31.4.2 void coco::table_delta::add (const std::string & _tn, const dbt_row & _r) [inline, inherited]

This method adds one row _r to the table _tn.

Definition at line 97 of file table_delta.h.

10.31.4.3 void coco::table_delta::add (const std::vector< t_line > & _tlv) [inline, inherited]

This method adds a list of table rows.

Definition at line 100 of file table_delta.h.

10.31.4.4 bool coco::boxes_delta::apply (work_node & _x, undelta_base *& _u, const delta_id & _did, size_t & delta_size) const [virtual]

Apply the delta with delta_id _d to work node _x. This will never be used, because the boxes_delta is converted to annotation_delta before the apply is performed.

Reimplemented from coco::table_delta.

Definition at line 35 of file boxes_delta.cc.

10.31.4.5 virtual bool coco::coco::delta_base::apply (work_node & _x, undelta_base *& _u, const delta_id & _di, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.31.4.6 bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.31.4.7 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.31.4.8 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.31.4.9 void coco::table_delta::convert (work_node & _x, delta_base *& _u) [inherited]

This method converts the table delta to an annotation_delta after the information is stored in the search database. The conversion is based on work_node _x, and the generated annotation_delta is returned via _u.

Definition at line 43 of file table_delta.cc.

10.31.4.10 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.31.4.11 void coco::boxes_delta::create_table (work_node & _x, vdbl::standard_table *& ptb, const std::string & __t) const [virtual]

This method creates the box table in the search database for the work_node _x. A pointer ptb to the created table is set. The parameter ___t for the table name is ignored.

Reimplemented from coco::table_delta.

Definition at line 44 of file boxes_delta.cc.

10.31.4.12 void coco::boxes_delta::destroy_copy (delta_base * __d) const [inline, virtual]

Clone Destructor

Reimplemented from coco::table_delta.

Definition at line 95 of file boxes_delta.h.

10.31.4.13 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.31.4.14 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.31.4.15 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.31.4.16 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.31.4.17 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.31.4.18 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.31.4.19 boxes_delta* coco::boxes_delta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::table_delta.

Definition at line 92 of file boxes_delta.h.

10.31.4.20 bool coco::boxes_delta::operator!= (const delta_base & *_c*) const [inline]

Reimplemented from coco::table_delta.

Definition at line 113 of file boxes_delta.h.

10.31.4.21 bool coco::boxes_delta::operator!= (const boxes_delta & *_c*) const [inline]

Definition at line 120 of file boxes_delta.h.

10.31.4.22 bool coco::table_delta::operator!= (const table_delta & _c) const [inline, inherited]

Definition at line 144 of file table_delta.h.

10.31.4.23 bool coco::boxes_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Reimplemented from coco::table_delta.

Definition at line 110 of file boxes_delta.h.

10.31.4.24 bool coco::boxes_delta::operator== (const boxes_delta & _c) const [inline]

Comparison operators

Definition at line 118 of file boxes_delta.h.

10.31.4.25 bool coco::table_delta::operator== (const table_delta & _c) const [inline, inherited]

Comparison operators

Definition at line 143 of file table_delta.h.

10.31.4.26 void coco::table_delta::rm (const annotation & _tr) [inline, inherited]

This method adds the annotation `_tr` to the list of removed annotations.

Definition at line 104 of file table_delta.h.

10.31.4.27 void coco::table_delta::rm (const std::vector< annotation > & _trv) [inline, inherited]

This method adds the annotations in `_trv` to the list of removed annotations.

Definition at line 107 of file table_delta.h.

10.31.4.28 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.31.4.29 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.31.4.30 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.31.5 Member Data Documentation

10.31.5.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

The documentation for this class was generated from the following files:

- boxes_delta.h
- boxes_delta.cc

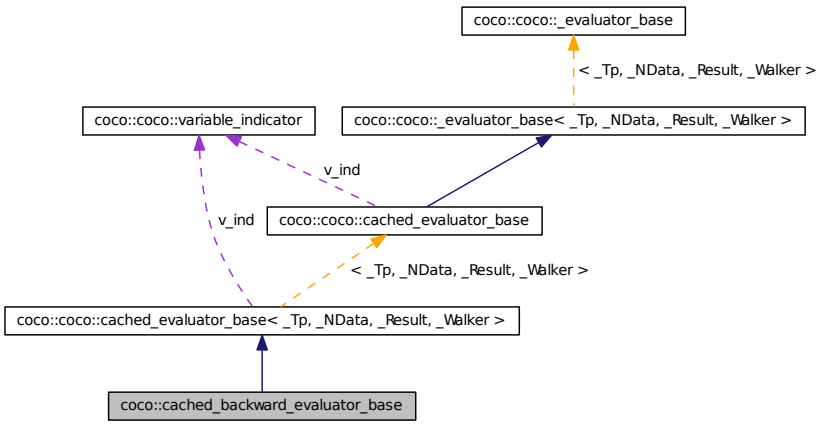
10.32 coco::cached_backward_evaluator_base Class Reference

Base class of all caching backward evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::cached_backward_evaluator_base:

Collaboration diagram for coco::cached_backward_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual void initialize ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual return_value calculate_value (bool eval_all)
- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind

10.32.1 Detailed Description

This class is the base class of all caching backward evaluators. Basically, it is a visitor to expression_node nodes in a DAG, based on evaluator_base.

10.32.2 Member Typedef Documentation

10.32.2.1 typedef _Base::const_walker coco::cached_backward_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 853 of file evaluator.h.

10.32.2.2 typedef _Base::node_data_type coco::cached_backward_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 849 of file evaluator.h.

10.32.2.3 typedef _Base::return_value coco::cached_backward_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 851 of file evaluator.h.

10.32.3 Member Function Documentation

10.32.3.1 virtual int coco::cached_backward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual]

This method is called right before all children of a node are visited. The `__data` parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 921 of file evaluator.h.

10.32.3.2 virtual return_value coco::cached_backward_evaluator_base::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Definition at line 956 of file evaluator.h.

10.32.3.3 virtual void coco::cached_backward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 925 of file evaluator.h.

10.32.3.4 int coco::cached_backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 879 of file evaluator.h.

10.32.3.5 virtual void coco::cached_backward_evaluator_base::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Definition at line 910 of file evaluator.h.

10.32.3.6 virtual bool coco::cached_backward_evaluator_base::is_cached (const node_data_type & __data) [inline, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 905 of file evaluator.h.

10.32.3.7 void coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 874 of file evaluator.h.

10.32.3.8 virtual int coco::coco::cached_evaluator_base::preorder (const node_data_type & __data) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 447 of file search_graph.cc.

10.32.3.9 int coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 862 of file evaluator.h.

10.32.3.10 virtual void coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 929 of file evaluator.h.

10.32.3.11 virtual const_walker coco::coco::cached_evaluator_base::short_cut_to (const node_data_type & __data) [inline, virtual, inherited]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 451 of file search_graph.cc.

10.32.3.12 virtual int coco::cached_backward_evaluator_base::update (const return_value & __rval) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 940 of file evaluator.h.

10.32.3.13 virtual int coco::cached_backward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 951 of file evaluator.h.

10.32.3.14 return_value coco::cached_backward_evaluator_base::value ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 891 of file evaluator.h.

10.32.3.15 int coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 885 of file evaluator.h.

10.32.3.16 void coco::cached_backward_evaluator_base::vinit ( ) [inline]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 900 of file evaluator.h.

10.32.3.17 return_value coco::cached_backward_evaluator_base::vvalue ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 896 of file evaluator.h.

10.32.4 Member Data Documentation

10.32.4.1 const variable_indicator* coco::coco::cached_evaluator_base::v_ind [protected, inherited]

For caching a variable indicator is needed to check which nodes need not be re-evaluated.

Definition at line 428 of file search_graph.cc.

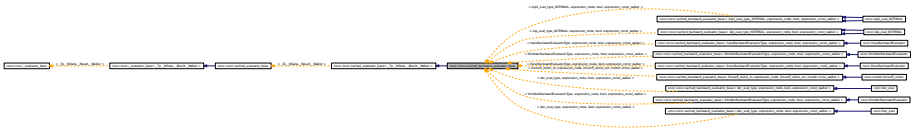
The documentation for this class was generated from the following file:

- evaluator.h

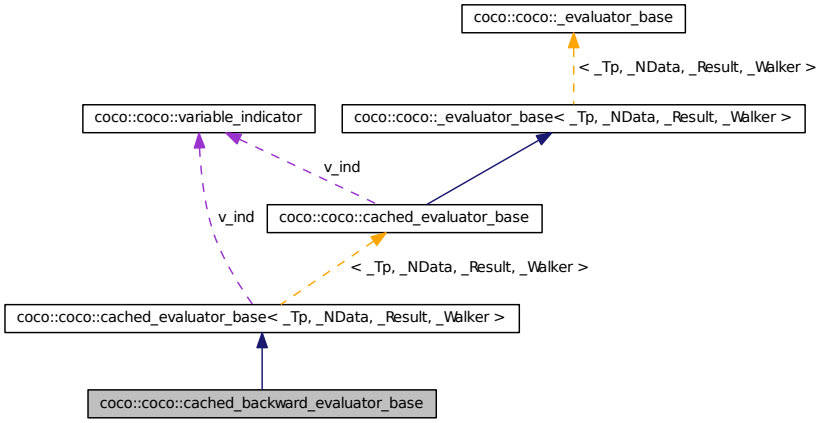
10.33 coco::coco::cached_backward_evaluator_base Class Reference

Base class of all caching backward evaluators.

Inheritance diagram for coco::coco::cached_backward_evaluator_base:



Collaboration diagram for coco::coco::cached_backward_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual void initialize ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)

- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual return_value calculate_value (bool eval_all)
- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind

10.33.1 Detailed Description

This class is the base class of all caching backward evaluators. Basically, it is a visitor to expression_node nodes in a DAG, based on evaluator_base.

10.33.2 Member Typedef Documentation

10.33.2.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 854 of file search_graph.cc.

10.33.2.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 850 of file search_graph.cc.

10.33.2.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 852 of file search_graph.cc.

10.33.3 Member Function Documentation

10.33.3.1 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.33.3.2 virtual return_value coco::coco::cached_backward_evaluator_base::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented in coco::model::lincoeff_visitor, coco::ithirdderBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, coco::thirdderBackwardEvaluator, coco::hessBackwardEvaluator, coco::der_eval, coco::islp2_eval_INTERNAL, coco::islp2_eval_INTERNAL, coco::islp_eval_INTERNAL, and coco::islp_eval_INTERNAL.

Definition at line 957 of file search_graph.cc.

10.33.3.3 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.33.3.4 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.33.3.5 virtual void coco::coco::cached_backward_evaluator_base::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented in coco::model::lincoeff_visitor, coco::ithirdderBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, coco::thirdderBackwardEvaluator, coco::hessBackwardEvaluator, coco::der_eval, coco::islp2_eval_INTERNAL, coco::islp2_eval_INTERNAL, coco::islp_eval_INTERNAL, and coco::islp_eval_INTERNAL.

Definition at line 911 of file search_graph.cc.

10.33.3.6 virtual bool coco::coco::cached_backward_evaluator_base::is_cached (const node_data_type & *__data*) [inline, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented in coco::ithirdderBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, coco::thirdderBackwardEvaluator, coco::hessBackwardEvaluator, coco::der_eval, coco::islp2_eval_INTERNAL, coco::islp2_eval_INTERNAL, coco::islp_eval_INTERNAL, and coco::islp_eval_INTERNAL.

Definition at line 906 of file search_graph.cc.

10.33.3.7 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & *__data*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.33.3.8 virtual int coco::coco::cached_evaluator_base::preorder (const node_data_type & *__data*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 447 of file search_graph.cc.

10.33.3.9 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & *__data*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.33.3.10 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.33.3.11 virtual const_walker coco::coco::cached_evaluator_base::short_cut_to (const node_data_type & *__data*) [inline, virtual, inherited]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 451 of file search_graph.cc.

10.33.3.12 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.33.3.13 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.33.3.14 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter false.

Definition at line 892 of file search_graph.cc.

10.33.3.15 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.33.3.16 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.33.3.17 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter

`true`.

Definition at line 897 of file search_graph.cc.

10.33.4 Member Data Documentation

10.33.4.1 const variable_indicator* coco::coco::cached_evaluator_base::v_ind [protected, inherited]

For caching a variable indicator is needed to check which nodes need not be re-evaluated.

Definition at line 428 of file search_graph.cc.

The documentation for this class was generated from the following file:

- evaluator.h

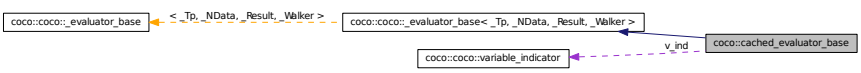
10.34 coco::cached_evaluator_base Class Reference

Base class of all caching evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::cached_evaluator_base:

Collaboration diagram for coco::cached_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker
- typedef _Tp data_type

Public Member Functions

- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)
- cached_evaluator_base ()
- cached_evaluator_base (const _Tp &__x, const variable_indicator &__v)
- cached_evaluator_base (const _Self &__x)
- virtual ~cached_evaluator_base ()
- virtual return_value vvalue ()
- virtual return_value value ()
- virtual int vcollect (const return_value &__cresult)
- virtual int collect (const node_data_type &__data, const return_value &__cresult)
- virtual void postorder (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- _Tp eval_data

10.34.1 Detailed Description

This class is the base class of all caching evaluators. Basically, it is a visitor to expression_node nodes in a DAG, based on _evaluator_base.

10.34.2 Member Typedef Documentation

10.34.2.1 typedef _Base::const_walker coco::cached_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 435 of file evaluator.h.

10.34.2.2 typedef _Tp coco::coco::_evaluator_base::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 306 of file search_graph.cc.

10.34.2.3 typedef _Base::node_data_type coco::cached_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 431 of file evaluator.h.

10.34.2.4 typedef _Base::return_value coco::cached_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 433 of file evaluator.h.

10.34.3 Constructor & Destructor Documentation

10.34.3.1 coco::cached_evaluator_base::cached_evaluator_base ( ) [inline]

Standard Constructor

Definition at line 455 of file evaluator.h.

10.34.3.2 coco::cached_evaluator_base::cached_evaluator_base (const _Tp & *__x*, const variable_indicator & *__v*) [inline]

Constructor, which initializes the internal data and the variable indicator

Definition at line 458 of file evaluator.h.

10.34.3.3 coco::cached_evaluator_base::cached_evaluator_base (const _Self & *__x*) [inline]

Standard Copy Constructor

Definition at line 461 of file evaluator.h.

10.34.3.4 virtual coco::cached_evaluator_base::~cached_evaluator_base ( ) [inline, virtual]

Standard Destructor

Definition at line 464 of file evaluator.h.

10.34.4 Member Function Documentation

10.34.4.1 virtual int coco::coco::_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__cresult*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 360 of file search_graph.cc.

10.34.4.2 virtual void coco::coco::_evaluator_base::postorder (const node_data_type & __data) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited.

Definition at line 365 of file search_graph.cc.

10.34.4.3 virtual int coco::cached_evaluator_base::preorder (const node_data_type & __data) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 446 of file evaluator.h.

10.34.4.4 virtual const_walker coco::cached_evaluator_base::short_cut_to (const node_data_type & __data) [inline, virtual]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 450 of file evaluator.h.

10.34.4.5 virtual return_value coco::coco::_evaluator_base::value ( ) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value.

Definition at line 337 of file search_graph.cc.

10.34.4.6 virtual int coco::coco::_evaluator_base::vcollect (const return_value & __cresult) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 349 of file search_graph.cc.

10.34.4.7 virtual return_value coco::coco::_evaluator_base::vvalue ( ) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value.

Definition at line 333 of file search_graph.cc.

10.34.5 Member Data Documentation

10.34.5.1 _Tp coco::coco::_evaluator_base::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 317 of file search_graph.cc.

10.34.5.2 const variable_indicator* coco::cached_evaluator_base::v_ind [protected]

For caching a variable indicator is needed to check which nodes need not be re-evaluated.

Definition at line 427 of file evaluator.h.

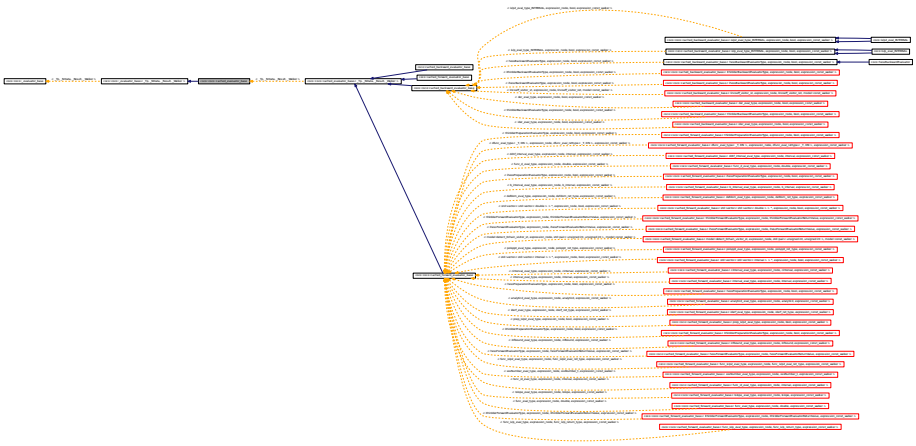
The documentation for this class was generated from the following file:

- evaluator.h

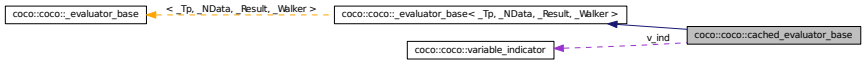
10.35 coco::coco::cached_evaluator_base Class Reference

Base class of all caching evaluators.

Inheritance diagram for coco::coco::cached_evaluator_base:



Collaboration diagram for coco::coco::cached_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker
- typedef _Tp data_type

Public Member Functions

- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)
- cached_evaluator_base ()
- cached_evaluator_base (const _Tp &__x, const variable_indicator &__v)
- cached_evaluator_base (const _Self &__x)
- virtual ~cached_evaluator_base ()
- virtual return_value vvalue ()
- virtual return_value value ()
- virtual int vcollect (const return_value &__cresult)
- virtual int collect (const node_data_type &__data, const return_value &__cresult)
- virtual void postorder (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- _Tp eval_data

10.35.1 Detailed Description

This class is the base class of all caching evaluators. Basically, it is a visitor to expression_node nodes in a DAG, based on _evaluator_base.

10.35.2 Member Typedef Documentation

10.35.2.1 typedef _Base::const_walker coco::coco::cached_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::coco::cached_backward_evaluator_base, coco::cached_backward_evaluator_base, coco::coco::cached_forward_evaluator_base, and coco::cached_forward_evaluator_base.

Definition at line 436 of file search_graph.cc.

10.35.2.2 typedef _Tp coco::coco::_evaluator_base::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 306 of file search_graph.cc.

10.35.2.3 typedef _Base::node_data_type coco::coco::cached_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::coco::cached_backward_evaluator_base, coco::cached_backward_evaluator_base, coco::coco::cached_forward_evaluator_base, and coco::cached_forward_evaluator_base.

Definition at line 432 of file search_graph.cc.

10.35.2.4 typedef _Base::return_value coco::coco::cached_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::coco::cached_backward_evaluator_base, coco::cached_backward_evaluator_base, coco::coco::cached_forward_evaluator_base, and coco::cached_forward_evaluator_base.

Definition at line 434 of file search_graph.cc.

10.35.3 Constructor & Destructor Documentation

10.35.3.1 coco::coco::cached_evaluator_base::cached_evaluator_base ( ) [inline]

Standard Constructor

Definition at line 456 of file search_graph.cc.

10.35.3.2 coco::coco::cached_evaluator_base::cached_evaluator_base (const _Tp & __x, const variable_indicator & __v) [inline]

Constructor, which initializes the internal data and the variable indicator

Definition at line 459 of file search_graph.cc.

10.35.3.3 coco::coco::cached_evaluator_base::cached_evaluator_base (const _Self & __x) [inline]

Standard Copy Constructor

Definition at line 462 of file search_graph.cc.

10.35.3.4 virtual coco::coco::cached_evaluator_base::~cached_evaluator_base ( ) [inline, virtual]

Standard Destructor

Definition at line 465 of file search_graph.cc.

10.35.4 Member Function Documentation

10.35.4.1 virtual int coco::coco::_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__cresult*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 360 of file search_graph.cc.

10.35.4.2 virtual void coco::coco::_evaluator_base::postorder (const node_data_type & *__data*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited.

Definition at line 365 of file search_graph.cc.

10.35.4.3 virtual int coco::coco::cached_evaluator_base::preorder (const node_data_type & *__data*) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 447 of file search_graph.cc.

10.35.4.4 virtual const_walker coco::coco::cached_evaluator_base::short_cut_to (const node_data_type & *__data*) [inline, virtual]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 451 of file search_graph.cc.

10.35.4.5 virtual return_value coco::coco::_evaluator_base::value ( ) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value.

Definition at line 337 of file search_graph.cc.

10.35.4.6 virtual int coco::coco::_evaluator_base::vcollect (const return_value & *__cresult*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 349 of file search_graph.cc.

10.35.4.7 virtual return_value coco::coco::_evaluator_base::vvalue ( ) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value.

Definition at line 333 of file search_graph.cc.

10.35.5 Member Data Documentation

10.35.5.1 _Tp coco::coco::_evaluator_base::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 317 of file search_graph.cc.

10.35.5.2 const variable_indicator* coco::coco::cached_evaluator_base::v_ind [protected]

For caching a variable indicator is needed to check which nodes need not be re-evaluated.

Definition at line 428 of file search_graph.cc.

The documentation for this class was generated from the following file:

- evaluator.h

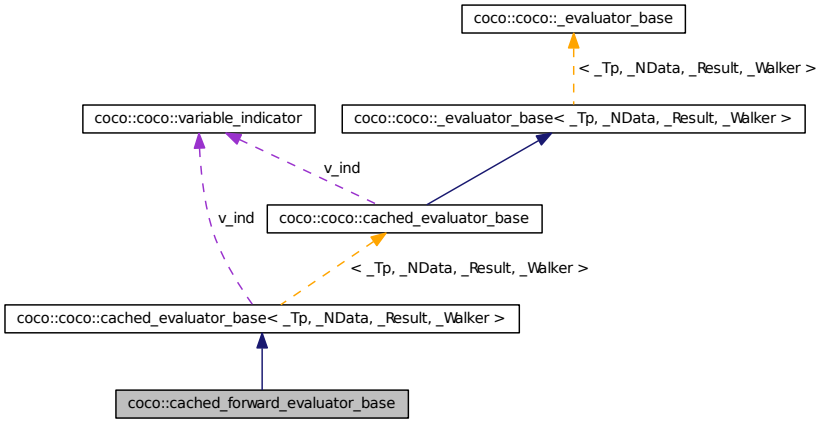
10.36 coco::cached_forward_evaluator_base Class Reference

Base class of all (non-caching) forward evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::cached_forward_evaluator_base:

Collaboration diagram for coco::cached_forward_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual void initialize ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)
- virtual return_value calculate_value (bool eval_all)
- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind

10.36.1 Detailed Description

This class is the base class of all non-caching forward evaluators. Basically, it is a visitor to expression_node nodes in a DAG, based on evaluator_base.

10.36.2 Member Typedef Documentation

10.36.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 723 of file evaluator.h.

10.36.2.2 typedef _Base::node_data_type coco::cached_forward_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 719 of file evaluator.h.

10.36.2.3 typedef _Base::return_value coco::cached_forward_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 721 of file evaluator.h.

10.36.3 Member Function Documentation

10.36.3.1 virtual void coco::cached_forward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual]

This method is called right after all children of a node have been visited. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 796 of file evaluator.h.

10.36.3.2 virtual return_value coco::cached_forward_evaluator_base::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Definition at line 830 of file evaluator.h.

10.36.3.3 virtual void coco::cached_forward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 804 of file evaluator.h.

10.36.3.4 int coco::cached_forward_evaluator_base::collect (const node_data_type & *__data,* const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 750 of file evaluator.h.

10.36.3.5 virtual void coco::cached_forward_evaluator_base::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Definition at line 781 of file evaluator.h.

10.36.3.6 virtual int coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 792 of file evaluator.h.

10.36.3.7 virtual bool coco::cached_forward_evaluator_base::is_cached (const node_data_type & *__data*) [inline, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 776 of file evaluator.h.

10.36.3.8 void coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 744 of file evaluator.h.

10.36.3.9 virtual int coco::coco::cached_evaluator_base::preorder (const node_data_type & __data) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 447 of file search_graph.cc.

10.36.3.10 int coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 732 of file evaluator.h.

10.36.3.11 virtual void coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 800 of file evaluator.h.

10.36.3.12 virtual const_walker coco::coco::cached_evaluator_base::short_cut_to (const node_data_type & __data) [inline, virtual, inherited]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 451 of file search_graph.cc.

10.36.3.13 virtual int coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 814 of file evaluator.h.

10.36.3.14 virtual int coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 826 of file evaluator.h.

10.36.3.15 return_value coco::cached_forward_evaluator_base::value ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter false.

Definition at line 762 of file evaluator.h.

10.36.3.16 int coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 756 of file evaluator.h.

10.36.3.17 void coco::cached_forward_evaluator_base::vinit ( ) [inline]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 771 of file evaluator.h.

10.36.3.18 return_value coco::cached_forward_evaluator_base::vvalue ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter true.

Definition at line 767 of file evaluator.h.

10.36.4 Member Data Documentation

10.36.4.1 const variable_indicator* coco::coco::cached_evaluator_base::v_ind [protected, inherited]

For caching a variable indicator is needed to check which nodes need not be re-evaluated.

Definition at line 428 of file search_graph.cc.

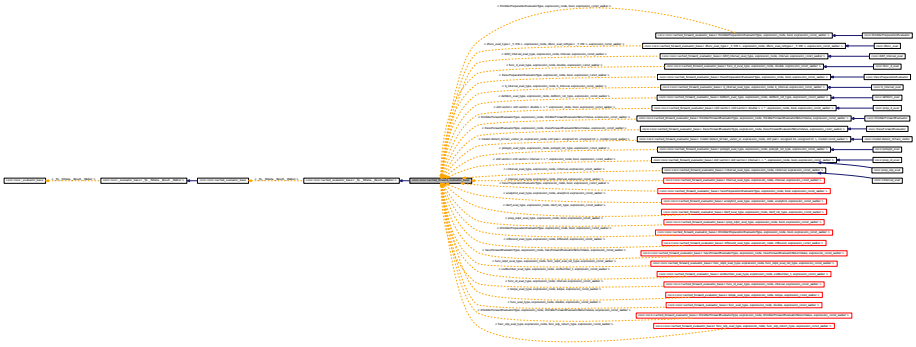
The documentation for this class was generated from the following file:

- evaluator.h

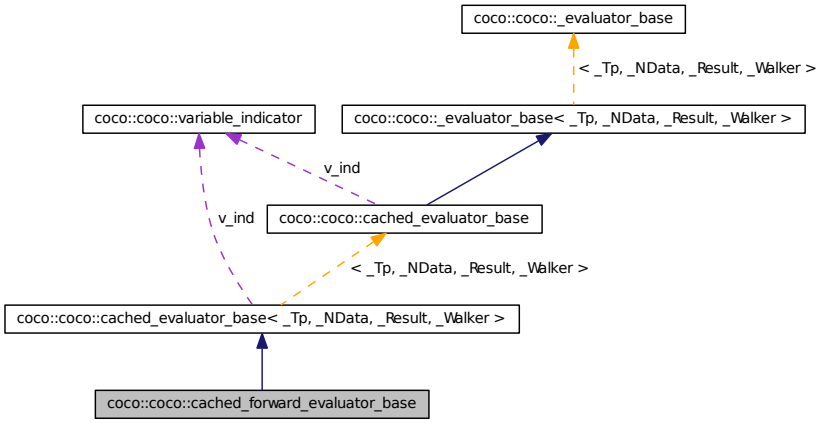
10.37 coco::coco::cached_forward_evaluator_base Class Reference

Base class of all (non-caching) forward evaluators.

Inheritance diagram for coco::coco::cached_forward_evaluator_base:



Collaboration diagram for coco::coco::cached_forward_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual void initialize ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)
- virtual return_value calculate_value (bool eval_all)
- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind

10.37.1 Detailed Description

This class is the base class of all non-caching forward evaluators. Basically, it is a visitor to expression_-node nodes in a DAG, based on evaluator_base.

10.37.2 Member Typedef Documentation

10.37.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 724 of file search_graph.cc.

10.37.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data-_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 720 of file search_graph.cc.

10.37.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 722 of file search_graph.cc.

10.37.3 Member Function Documentation

10.37.3.1 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.37.3.2 virtual return_value coco::coco::cached_forward_evaluator_base::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented in coco::model::detect_0chain_visitor, coco::polsppt_eval, coco::ithirdderForwardEvaluator, coco::func_id_eval, coco::func_islp_eval, coco::func_islp2_eval, coco::ihessForwardEvaluator, coco::thirdderForwardEvaluator, coco::hessForwardEvaluator, coco::func_d_eval, coco::defdom_eval, coco::iderf_eval, coco::xxxNumber_eval, coco::dd1f_interval_eval, coco::interval_eval, coco::func_eval, coco::infbound_eval, coco::analyticd_eval, coco::b_interval_eval, coco::Islope_eval, coco::cinterval_eval, coco::dfunc_eval, coco::ithirdderPreparationEvaluator, coco::thirdderPreparationEvaluator, coco::prep_islp2_eval, coco::ihessPreparationEvaluator, coco::hessPreparationEvaluator, coco::prep_islp_eval, coco::prep_id_eval, and coco::prep_d_eval.

Definition at line 831 of file search_graph.cc.

10.37.3.3 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.37.3.4 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.37.3.5 virtual void coco::coco::cached_forward_evaluator_base::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented in coco::model::detect_0chain_visitor, coco::ihessForwardEvaluator, coco::ithirdderForwardEvaluator, coco::func_islp2_eval, coco::hessForwardEvaluator, coco::func_id_eval, coco::xxxNumber_eval, coco::thirdderForwardEvaluator, coco::func_d_eval, coco::func_islp_eval, coco::defdom_eval, coco::polsppt_eval, coco::iderf_eval, coco::Islope_eval, coco::dd1f_interval_eval, coco::interval_eval, coco::func_eval, coco::dfunc_eval, coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::infbound_eval, coco::prep_islp_eval, coco::ithirdderPreparationEvaluator, coco::thirdderPreparationEvaluator, coco::prep_id_eval, coco::prep_islp2_eval, coco::prep_d_eval, coco::ihessPreparationEvaluator, and coco::hessPreparationEvaluator.

Definition at line 782 of file search_graph.cc.

10.37.3.6 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.37.3.7 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & __data) [inline, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented in coco::ithirdderForwardEvaluator, coco::func_islp2_eval, coco::thirdderForwardEvaluator, coco::ihessForwardEvaluator, coco::hessForwardEvaluator, coco::func_id_eval, coco::func_d_eval, coco::defdom_eval, coco::xxxNumber_eval, coco::polsppt_eval, coco::iderf_eval, coco::dd1f_interval_eval, coco::Islope_eval, coco::interval_eval, coco::func_eval, coco::dfunc_eval, coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, and coco::infbound_eval.

Definition at line 777 of file search_graph.cc.

10.37.3.8 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.37.3.9 virtual int coco::coco::cached_evaluator_base::preorder (const node_data_type & __data) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 447 of file search_graph.cc.

10.37.3.10 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.37.3.11 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.37.3.12 virtual const_walker coco::coco::cached_evaluator_base::short_cut_to (const node_data_type & *__data*) [inline, virtual, inherited]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 451 of file search_graph.cc.

10.37.3.13 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.37.3.14 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.37.3.15 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.37.3.16 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.37.3.17 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.37.3.18 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

10.37.4 Member Data Documentation

10.37.4.1 const variable_indicator* coco::coco::cached_evaluator_base::v_ind [protected, inherited]

For caching a variable indicator is needed to check which nodes need not be re-evaluated.

Definition at line 428 of file search_graph.cc.

The documentation for this class was generated from the following file:

- evaluator.h

10.38 coco::calc_pf_star Class Reference

Stored procedure calculating the pf∗ value of a box.

`#include <pfstar.h>`

Public Types

- typedef search_graph_context context
- typedef double return_type

Public Member Functions

- calc_pf_star (vdbl::colid _l, vdbl::colid _w)
- calc_pf_star (const calc_pf_star &_i)
- virtual ∼calc_pf_star ()
- double operator() () const
- double def () const
- void setcontext (const context ∗c, const vdbl::row ∗r)

10.38.1 Detailed Description

This class is used as a stored procedure (vdbl::method) for calculating the pf∗ value of a subproblem.

10.38.2 Member Typedef Documentation

10.38.2.1 typedef search_graph_context coco::calc_pf_star::context

The evaluation context type

Definition at line 47 of file pfstar.h.

10.38.2.2 typedef double coco::calc_pf_star::return_type

The return type of the operator(), i.e. the function class.

Definition at line 63 of file pfstar.h.

10.38.3 Constructor & Destructor Documentation

10.38.3.1 coco::calc_pf_star::calc_pf_star (vdbl::colid _l, vdbl::colid _w) [inline]

Constructor setting the relevant column id.

Definition at line 66 of file pfstar.h.

10.38.3.2 coco::calc_pf_star::calc_pf_star (const calc_pf_star & *_i*) [inline]

Standard Copy Constructor

Definition at line 69 of file pfstar.h.

10.38.3.3 virtual coco::calc_pf_star::~calc_pf_star ( ) [inline, virtual]

Standard Destructor

Definition at line 73 of file pfstar.h.

10.38.4 Member Function Documentation

10.38.4.1 double coco::calc_pf_star::def ( ) const [inline]

Method, which computes the default value of the stored procedure

Definition at line 78 of file pfstar.h.

10.38.4.2 double coco::calc_pf_star::operator() ( ) const

Evaluation operator, which computes the value of the stored procedure

Definition at line 36 of file pfstar.cc.

10.38.4.3 void coco::calc_pf_star::setcontext (const context * *c*, const vdbl::row * *r*) [inline]

This method initializes the evaluation context and the row, preparing for the evaluation.

Definition at line 81 of file pfstar.h.

The documentation for this class was generated from the following files:

- pfstar.h
- pfstar.cc

10.39 coco::certificate Class Reference

The certificate class (certifies deltas for rigorous mode operation)

`#include <api_certbase.h>`

Public Member Functions

- certificate ()
- certificate (const certificate_base &__c)
- certificate (const certificate &__c)
- ~certificate ()
- std::string get_contents () const
- const certificate_base * get_base () const

- bool verify (const work_node &_x) const
- certificate & operator= (const certificate &_c)
- bool operator== (const certificate &_c) const
- bool operator!= (const certificate &_c) const

Friends

- class certificate_base
- class ie_return_type
- std::ostream & operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.

10.39.1 Detailed Description

The certificate class is used to certify deltas in rigorous mode. It is, like the delta class, designed as a wrapper class in order to overcome the problem that copy constructors cannot be overloaded. Certificates are stored alongside the corresponding deltas in the table `deltas`.

10.39.2 Constructor & Destructor Documentation

10.39.2.1 coco::certificate::certificate ( ) [inline]

Standard Constructor

Definition at line 62 of file api_certbase.h.

10.39.2.2 coco::certificate::certificate (const certificate_base & *__c*)

Constructor, which constructs a certificate from a certificate_base. The clone operation for the certificate_base will be called in that process.

10.39.2.3 coco::certificate::certificate (const certificate & *__c*)

Copy Constructor, which constructs a new certificate from an existing certificate. The clone operation for the certificate_base will be called in that process, effectively overloading the copy constructor for the certificate_base.

10.39.2.4 coco::certificate::~certificate ( )

Destructor, which frees the `_c` using `delete`

10.39.3 Member Function Documentation

10.39.3.1 const certificate_base* coco::certificate::get_base ( ) const

Return the certificate_base stored in this wrapper

10.39.3.2 std::string coco::certificate::get_contents ( ) const

Retrieve the contents information (the certificate type) for this certificate.

10.39.3.3 bool coco::certificate::operator!= (const certificate & *_c*) const

10.39.3.4 certificate& coco::certificate::operator= (const certificate & *_c*)

Assignment operator, which uses the clone operation for the certificate_base, effectively overloading the assignment operator for the certificate_base.

10.39.3.5 bool coco::certificate::operator== (const certificate & *_c*) const

Comparison operators

10.39.3.6 bool coco::certificate::verify (const work_node & *_x*) const

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x.

10.39.4 Friends And Related Function Documentation

10.39.4.1 friend class certificate_base [friend]

Definition at line 102 of file api_certbase.h.

10.39.4.2 friend class ie_return_type [friend]

Definition at line 103 of file api_certbase.h.

10.39.4.3 std::ostream& operator<< (std::ostream & *o*, const certificate & *t*) [friend]

Stream output operator for certificates, which calls the get_contents method of the certificate.

Definition at line 112 of file api_certbase.h.

The documentation for this class was generated from the following file:

- api_certbase.h

10.40 coco::coco::certificate Class Reference

The certificate class (certifies deltas for rigorous mode operation)

Public Member Functions

- certificate ()
- certificate (const certificate_base &__c)
- certificate (const certificate &__c)
- ~certificate ()

- std::string get_contents () const
- const certificate_base ∗ get_base () const
- bool verify (const work_node &_x) const
- certificate & operator= (const certificate &_c)
- bool operator== (const certificate &_c) const
- bool operator!= (const certificate &_c) const
- certificate ()
- certificate (const certificate_base &__c)
- certificate (const certificate &__c)
- ∼certificate ()
- std::string get_contents () const
- const certificate_base ∗ get_base () const
- bool verify (const work_node &_x) const
- certificate & operator= (const certificate &_c)
- bool operator== (const certificate &_c) const
- bool operator!= (const certificate &_c) const
- certificate ()
- certificate (const certificate_base &__c)
- certificate (const certificate &__c)
- ∼certificate ()
- std::string get_contents () const
- const certificate_base ∗ get_base () const
- bool verify (const work_node &_x) const
- certificate & operator= (const certificate &_c)
- bool operator== (const certificate &_c) const
- bool operator!= (const certificate &_c) const
- certificate ()
- certificate (const certificate_base &__c)
- certificate (const certificate &__c)
- ∼certificate ()
- std::string get_contents () const
- const certificate_base ∗ get_base () const
- bool verify (const work_node &_x) const
- certificate & operator= (const certificate &_c)
- bool operator== (const certificate &_c) const
- bool operator!= (const certificate &_c) const

Friends

- class certificate_base
- class ie_return_type
- std::ostream & operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.
- std::ostream & operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.
- std::ostream & operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.
- std::ostream & operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.

10.40.1 Detailed Description

The certificate class is used to certify deltas in rigorous mode. It is, like the delta class, designed as a wrapper class in order to overcome the problem that copy constructors cannot be overloaded. Certificates are stored alongside the corresponding deltas in the table `deltas`.

Definition at line 54 of file search_graph.cc.

10.40.2 Constructor & Destructor Documentation

10.40.2.1 coco::coco::certificate::certificate ( ) [inline]

Standard Constructor

Definition at line 62 of file search_graph.cc.

10.40.2.2 coco::certificate::certificate (const certificate_base & *__c*) [inline]

Constructor, which constructs a certificate from a certificate_base. The clone operation for the certificate_base will be called in that process.

Definition at line 41 of file api_cert.h.

10.40.2.3 coco::certificate::certificate (const certificate & *__c*) [inline]

Copy Constructor, which constructs a new certificate from an existing certificate. The clone operation for the certificate_base will be called in that process, effectively overloading the copy constructor for the certificate_base.

Definition at line 43 of file api_cert.h.

10.40.2.4 coco::certificate::~certificate ( ) [inline]

Destructor, which frees the `_c` using `delete`

Definition at line 49 of file api_cert.h.

10.40.2.5 coco::coco::certificate::certificate ( ) [inline]

Standard Constructor

Definition at line 62 of file search_graph.cc.

10.40.2.6 coco::coco::certificate::certificate (const certificate_base & *__c*)

Constructor, which constructs a certificate from a certificate_base. The clone operation for the certificate_base will be called in that process.

10.40.2.7 coco::coco::certificate::certificate (const certificate & *__c*)

Copy Constructor, which constructs a new certificate from an existing certificate. The clone operation for the certificate_base will be called in that process, effectively overloading the copy constructor for the certificate_base.

10.40.2.8 coco::coco::certificate::~certificate ( )

Destructor, which frees the _c using `delete`

10.40.2.9 coco::coco::certificate::certificate ( ) [inline]

Standard Constructor

Definition at line 62 of file search_graph.cc.

10.40.2.10 coco::coco::certificate::certificate (const certificate_base & *__c*)

Constructor, which constructs a certificate from a certificate_base. The clone operation for the certificate_base will be called in that process.

10.40.2.11 coco::coco::certificate::certificate (const certificate & *__c*)

Copy Constructor, which constructs a new certificate from an existing certificate. The clone operation for the certificate_base will be called in that process, effectively overloading the copy constructor for the certificate_base.

10.40.2.12 coco::coco::certificate::~certificate ( )

Destructor, which frees the _c using `delete`

10.40.2.13 coco::coco::certificate::certificate ( ) [inline]

Standard Constructor

Definition at line 62 of file search_graph.cc.

10.40.2.14 coco::coco::certificate::certificate (const certificate_base & *__c*)

Constructor, which constructs a certificate from a certificate_base. The clone operation for the certificate_base will be called in that process.

10.40.2.15 coco::coco::certificate::certificate (const certificate & *__c*)

Copy Constructor, which constructs a new certificate from an existing certificate. The clone operation for the certificate_base will be called in that process, effectively overloading the copy constructor for the certificate_base.

10.40.2.16 coco::coco::certificate::~certificate ( )

Destructor, which frees the _c using `delete`

10.40.3 Member Function Documentation

10.40.3.1 const certificate_base * coco::certificate::get_base ( ) const [inline]

Return the certificate_base stored in this wrapper

Definition at line 67 of file api_cert.h.

10.40.3.2 const certificate_base* coco::coco::certificate::get_base ( ) const

Return the certificate_base stored in this wrapper

10.40.3.3 const certificate_base* coco::coco::certificate::get_base ( ) const

Return the certificate_base stored in this wrapper

10.40.3.4 const certificate_base* coco::coco::certificate::get_base ( ) const

Return the certificate_base stored in this wrapper

10.40.3.5 std::string coco::certificate::get_contents ( ) const [inline]

Retrieve the contents information (the certificate type) for this certificate.

Definition at line 65 of file api_cert.h.

10.40.3.6 std::string coco::coco::certificate::get_contents ( ) const

Retrieve the contents information (the certificate type) for this certificate.

10.40.3.7 std::string coco::coco::certificate::get_contents ( ) const

Retrieve the contents information (the certificate type) for this certificate.

10.40.3.8 std::string coco::coco::certificate::get_contents ( ) const

Retrieve the contents information (the certificate type) for this certificate.

10.40.3.9 bool coco::coco::certificate::operator!= (const certificate & _c) const

10.40.3.10 bool coco::coco::certificate::operator!= (const certificate & _c) const

10.40.3.11 bool coco::coco::certificate::operator!= (const certificate & _c) const

10.40.3.12 bool coco::certificate::operator!= (const certificate & _c) const [inline]

Definition at line 74 of file api_cert.h.

10.40.3.13 certificate & coco::certificate::operator= (const certificate & _c) [inline]

Assignment operator, which uses the clone operation for the certificate_base, effectively overloading the assignment operator for the certificate_base.

Definition at line 55 of file api_cert.h.

10.40.3.14 certificate& coco::coco::certificate::operator= (const certificate & _c)

Assignment operator, which uses the clone operation for the certificate_base, effectively overloading the assignment operator for the certificate_base.

10.40.3.15 certificate& coco::coco::certificate::operator= (const certificate & _c)

Assignment operator, which uses the clone operation for the certificate_base, effectively overloading the assignment operator for the certificate_base.

10.40.3.16 certificate& coco::coco::certificate::operator= (const certificate & _c)

Assignment operator, which uses the clone operation for the certificate_base, effectively overloading the assignment operator for the certificate_base.

10.40.3.17 bool coco::coco::certificate::operator== (const certificate & _c) const

Comparison operators

10.40.3.18 bool coco::coco::certificate::operator== (const certificate & _c) const

Comparison operators

10.40.3.19 bool coco::certificate::operator== (const certificate & _c) const [inline]

Comparison operators

Definition at line 71 of file api_cert.h.

10.40.3.20 bool coco::coco::certificate::operator== (const certificate & _c) const

Comparison operators

10.40.3.21 bool coco::coco::certificate::verify (const work_node & _x) const

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x.

10.40.3.22 bool coco::certificate::verify (const work_node & _x) const [inline]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x.

Definition at line 69 of file api_cert.h.

10.40.3.23 bool coco::coco::certificate::verify (const work_node & _x) const

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x.

10.40.3.24 bool coco::coco::certificate::verify (const work_node & _x) const

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x.

10.40.4 Friends And Related Function Documentation

10.40.4.1 certificate_base [friend]

Definition at line 102 of file search_graph.cc.

10.40.4.2 ie_return_type [friend]

Definition at line 103 of file search_graph.cc.

10.40.4.3 std::ostream& operator<< (std::ostream & *o*, const certificate & *t*) [friend]

Stream output operator for certificates, which calls the get_contents method of the certificate.

Definition at line 112 of file search_graph.cc.

10.40.4.4 std::ostream& operator<< (std::ostream & *o*, const certificate & *t*) [friend]

Stream output operator for certificates, which calls the get_contents method of the certificate.

Definition at line 112 of file search_graph.cc.

10.40.4.5 std::ostream& operator<< (std::ostream & *o*, const certificate & *t*) [friend]

Stream output operator for certificates, which calls the get_contents method of the certificate.

Definition at line 112 of file search_graph.cc.

10.40.4.6 std::ostream& operator<< (std::ostream & *o*, const certificate & *t*) [friend]

Stream output operator for certificates, which calls the get_contents method of the certificate.

Definition at line 112 of file search_graph.cc.

The documentation for this class was generated from the following files:

- api_certbase.h
- api_cert.h

10.41 coco::certificate_base Class Reference

Base class for the certificates.

`#include <api_certbase.h>`

Public Member Functions

- certificate_base ()
- certificate_base (const std::string &c)
- certificate_base (const char *c)
- certificate_base (const certificate_base &__c)
- virtual certificate_base * new_copy () const PURE_VIRTUALvirtual void destroy_copy(certificate_base *__c) const PURE_VIRTUALvirtual~certificate_base()
- certificate make_certificate (const std::string &c)
- virtual std::string get_contents () const
- virtual bool verify (const work_node &_x) const

Protected Attributes

- std::string _contents

10.41.1 Detailed Description

Base class for the certificates, which is wrapped by the certificate class since copy constructors cannot directly be overloaded.

10.41.2 Constructor & Destructor Documentation

10.41.2.1 coco::certificate_base::certificate_base ( ) [inline]

Standard Constructor

Definition at line 132 of file api_certbase.h.

10.41.2.2 coco::certificate_base::certificate_base (const std::string & *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 134 of file api_certbase.h.

10.41.2.3 coco::certificate_base::certificate_base (const char * *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 136 of file api_certbase.h.

10.41.2.4 coco::certificate_base::certificate_base (const certificate_base & *__c*) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 138 of file api_certbase.h.

10.41.3 Member Function Documentation

10.41.3.1 virtual std::string coco::certificate_base::get_contents ( ) const [inline, virtual]

Retrieve the contents information (the certificate type) for this certificate.

Definition at line 165 of file api_certbase.h.

10.41.3.2 certificate coco::certificate_base::make_certificate (const std::string & *c*) [inline]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file api_certbase.h.

10.41.3.3 virtual certificate_base* coco::certificate_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Definition at line 146 of file api_certbase.h.

10.41.3.4 virtual bool coco::certificate_base::verify (const work_node & _x) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. Will be overloaded.

Definition at line 169 of file api_certbase.h.

10.41.4 Member Data Documentation

10.41.4.1 std::string coco::certificate_base::_contents [protected]

The contents (descriptive string, type) of this certificate

Definition at line 128 of file api_certbase.h.

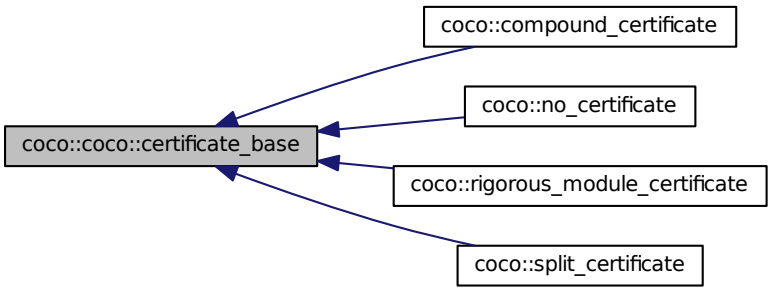
The documentation for this class was generated from the following file:

- api_certbase.h

10.42 coco::coco::certificate_base Class Reference

Base class for the certificates.

Inheritance diagram for coco::coco::certificate_base:



Public Member Functions

- certificate_base ()

- certificate_base (const std::string &c)
- certificate_base (const char ∗c)
- certificate_base (const certificate_base &__c)
- virtual certificate_base ∗ new_copy () const PURE_VIRTUALvirtual void destroy_copy(certificate_base ∗__c) const PURE_VIRTUALvirtual∼certificate_base()
- certificate make_certificate (const std::string &c)
- virtual std::string get_contents () const
- virtual bool verify (const work_node &_x) const
- certificate_base ()
- certificate_base (const std::string &c)
- certificate_base (const char ∗c)
- certificate_base (const certificate_base &__c)
- virtual certificate_base ∗ new_copy () const PURE_VIRTUALvirtual void destroy_copy(certificate_base ∗__c) const PURE_VIRTUALvirtual∼certificate_base()
- certificate make_certificate (const std::string &c)
- virtual std::string get_contents () const
- virtual bool verify (const work_node &_x) const
- certificate_base ()
- certificate_base (const std::string &c)
- certificate_base (const char ∗c)
- certificate_base (const certificate_base &__c)
- virtual certificate_base ∗ new_copy () const PURE_VIRTUALvirtual void destroy_copy(certificate_base ∗__c) const PURE_VIRTUALvirtual∼certificate_base()
- certificate make_certificate (const std::string &c)
- virtual std::string get_contents () const
- virtual bool verify (const work_node &_x) const
- certificate_base ()
- certificate_base (const std::string &c)
- certificate_base (const char ∗c)
- certificate_base (const certificate_base &__c)
- virtual certificate_base ∗ new_copy () const PURE_VIRTUALvirtual void destroy_copy(certificate_base ∗__c) const PURE_VIRTUALvirtual∼certificate_base()
- certificate make_certificate (const std::string &c)
- virtual std::string get_contents () const
- virtual bool verify (const work_node &_x) const

Protected Attributes

- std::string _contents

10.42.1 Detailed Description

Base class for the certificates, which is wrapped by the certificate class since copy constructors cannot directly be overloaded.

Definition at line 124 of file search_graph.cc.

10.42.2 Constructor & Destructor Documentation

10.42.2.1 coco::coco::certificate_base::certificate_base ( ) [inline]

Standard Constructor

Definition at line 132 of file search_graph.cc.

10.42.2.2 coco::coco::certificate_base::certificate_base (const std::string & *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 134 of file search_graph.cc.

10.42.2.3 coco::coco::certificate_base::certificate_base (const char * *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 136 of file search_graph.cc.

10.42.2.4 coco::coco::certificate_base::certificate_base (const certificate_base & *__c*) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 138 of file search_graph.cc.

10.42.2.5 coco::coco::certificate_base::certificate_base ( ) [inline]

Standard Constructor

Definition at line 132 of file search_graph.cc.

10.42.2.6 coco::coco::certificate_base::certificate_base (const std::string & *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 134 of file search_graph.cc.

10.42.2.7 coco::coco::certificate_base::certificate_base (const char * *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 136 of file search_graph.cc.

10.42.2.8 coco::coco::certificate_base::certificate_base (const certificate_base & *__c*) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 138 of file search_graph.cc.

10.42.2.9 coco::coco::certificate_base::certificate_base ( ) [inline]

Standard Constructor

Definition at line 132 of file search_graph.cc.

10.42.2.10 coco::coco::certificate_base::certificate_base (const std::string & *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 134 of file search_graph.cc.

10.42.2.11 coco::coco::certificate_base::certificate_base (const char * *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 136 of file search_graph.cc.

10.42.2.12 coco::coco::certificate_base::certificate_base (const certificate_base & *__c*) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 138 of file search_graph.cc.

10.42.2.13 coco::coco::certificate_base::certificate_base ( ) [inline]

Standard Constructor

Definition at line 132 of file search_graph.cc.

10.42.2.14 coco::coco::certificate_base::certificate_base (const std::string & *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 134 of file search_graph.cc.

10.42.2.15 coco::coco::certificate_base::certificate_base (const char * *c*) [inline]

Constructor, explicitly setting the contents to c

Definition at line 136 of file search_graph.cc.

10.42.2.16 coco::coco::certificate_base::certificate_base (const certificate_base & *__c*) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 138 of file search_graph.cc.

10.42.3 Member Function Documentation

10.42.3.1 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.42.3.2 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.42.3.3 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.42.3.4 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.42.3.5 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.42.3.6 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.42.3.7 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.42.3.8 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.42.3.9 virtual certificate_base* coco::coco::certificate_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::rigorous_module_certificate, coco::compound_certificate, coco::split_certificate, and coco::no_certificate.

Definition at line 146 of file search_graph.cc.

10.42.3.10 virtual certificate_base* coco::coco::certificate_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::rigorous_module_certificate, coco::compound_certificate, coco::split_certificate, and coco::no_certificate.

Definition at line 146 of file search_graph.cc.

10.42.3.11 virtual certificate_base* coco::coco::certificate_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::rigorous_module_certificate, coco::compound_certificate, coco::split_certificate, and coco::no_certificate.

Definition at line 146 of file search_graph.cc.

10.42.3.12 virtual certificate_base* coco::coco::certificate_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::rigorous_module_certificate, coco::compound_certificate, coco::split_certificate, and coco::no_certificate.

Definition at line 146 of file search_graph.cc.

10.42.3.13 virtual bool coco::coco::certificate_base::verify (const work_node & _x) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. Will be overloaded.

Reimplemented in coco::rigorous_module_certificate, coco::compound_certificate, coco::split_certificate, and coco::no_certificate.

Definition at line 169 of file search_graph.cc.

10.42.3.14 virtual bool coco::coco::certificate_base::verify (const work_node & _x) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. Will be overloaded.

Reimplemented in coco::rigorous_module_certificate, coco::compound_certificate, coco::split_certificate, and coco::no_certificate.

Definition at line 169 of file search_graph.cc.

10.42.3.15 virtual bool coco::coco::certificate_base::verify (const work_node & _x) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. Will be overloaded.

Reimplemented in coco::rigorous_module_certificate, coco::compound_certificate, coco::split_certificate, and coco::no_certificate.

Definition at line 169 of file search_graph.cc.

10.42.3.16 virtual bool coco::coco::certificate_base::verify (const work_node & _x) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. Will be overloaded.

Reimplemented in coco::rigorous_module_certificate, coco::compound_certificate, coco::split_certificate, and coco::no_certificate.

Definition at line 169 of file search_graph.cc.

10.42.4 Member Data Documentation

10.42.4.1 std::string coco::coco::certificate_base::_contents [protected]

The contents (descriptive string, type) of this certificate

Definition at line 128 of file search_graph.cc.

The documentation for this class was generated from the following file:

- api_certbase.h

10.43 coco::checking_my Struct Reference

`#include <interval_boost.h>`

Static Public Member Functions

- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)

10.43.1 Member Function Documentation

10.43.1.1 static T coco::checking_my::empty_inf ( ) [inline, static]

Definition at line 68 of file interval_boost.h.

10.43.1.2 static T coco::checking_my::empty_lower ( ) [inline, static]

Definition at line 78 of file interval_boost.h.

10.43.1.3 static T coco::checking_my::empty_upper ( ) [inline, static]

Definition at line 73 of file interval_boost.h.

10.43.1.4 static T coco::checking_my::inf ( ) [inline, static]

Definition at line 53 of file interval_boost.h.

10.43.1.5 static bool coco::checking_my::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 83 of file interval_boost.h.

10.43.1.6 static bool coco::checking_my::is_nan (const T & *x*) [inline, static]

Definition at line 64 of file interval_boost.h.

10.43.1.7 static T coco::checking_my::nan ( ) [inline, static]

Definition at line 59 of file interval_boost.h.

10.43.1.8 static T coco::checking_my::neg_inf ( ) [inline, static]

Definition at line 52 of file interval_boost.h.

10.43.1.9 static T coco::checking_my::pos_inf ( ) [inline, static]

Definition at line 51 of file interval_boost.h.

The documentation for this struct was generated from the following file:

- interval_boost.h

10.44 coco::coco::checking_my< T > Struct Template Reference

`#include <expression.h>`

Static Public Member Functions

- static T pos_inf ()
- static T neg_inf ()

- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()

- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()

- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)

10.44.1 Detailed Description

template<class T>struct coco::coco::checking_my< T >

Definition at line 50 of file expression.h.

10.44.2 Member Function Documentation

10.44.2.1 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file expression.h.

10.44.2.2 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.3 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.4 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.5 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.6 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.7 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file expression.h.

10.44.2.8 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.9 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.10 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.11 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.12 template<class T > static T coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.44.2.13 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file expression.h.

10.44.2.14 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file expression.h.

10.44.2.15 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.16 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.17 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.18 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.19 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.20 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.21 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.22 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.23 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.24 template<class T > static T coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.44.2.25 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file expression.h.

10.44.2.26 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.27 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.28 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.29 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.30 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.31 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.32 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.33 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.34 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.35 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.44.2.36 template<class T > static T coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file expression.h.

10.44.2.37 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.38 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.39 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.40 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.41 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.42 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.43 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.44 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.45 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file expression.h.

10.44.2.46 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.47 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.44.2.48 template<class T > static T coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file expression.h.

10.44.2.49 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file expression.h.

10.44.2.50 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.51 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.52 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.53 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.54 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file expression.h.

10.44.2.55 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.56 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.57 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.58 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.59 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.60 template<class T > static bool coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.44.2.61 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.62 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.63 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file expression.h.

10.44.2.64 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.65 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.66 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.67 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.68 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.69 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.70 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file expression.h.

10.44.2.71 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.72 template<class T > static bool coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.44.2.73 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.74 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.75 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file expression.h.

10.44.2.76 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.77 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.78 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.79 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.80 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.81 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file expression.h.

10.44.2.82 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.83 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.84 template<class T > static T coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.44.2.85 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.86 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.87 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.88 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file expression.h.

10.44.2.89 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.90 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.91 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.92 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file expression.h.

10.44.2.93 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.94 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.95 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.96 template<class T > static T coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.44.2.97 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.98 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file expression.h.

10.44.2.99 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file expression.h.

10.44.2.100 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.101 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.102 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.103 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.104 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.105 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.106 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.107 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.44.2.108 template<class T > static T coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

The documentation for this struct was generated from the following file:

- interval_boost.h

10.45 coco::coco::coco::checking_my< T > Struct Template Reference

Static Public Member Functions

- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)
- static T pos_inf ()
- static T neg_inf ()
- static T inf ()
- static T nan ()
- static bool is_nan (const T &x)
- static T empty_inf ()
- static T empty_upper ()
- static T empty_lower ()
- static bool is_empty (const T &l, const T &u)

10.45.1 Detailed Description

template<class T>struct coco::coco::coco::checking_my< T >

Definition at line 50 of file search_graph.cc.

10.45.2 Member Function Documentation

10.45.2.1 template<class T > static T coco::coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.45.2.2 template<class T > static T coco::coco::coco::checking_my< T >::empty_inf ( ) [inline, static]

Definition at line 69 of file search_graph.cc.

10.45.2.3 template<class T > static T coco::coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.45.2.4 template<class T > static T coco::coco::coco::checking_my< T >::empty_lower ( ) [inline, static]

Definition at line 79 of file search_graph.cc.

10.45.2.5 template<class T > static T coco::coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.45.2.6 template<class T > static T coco::coco::coco::checking_my< T >::empty_upper ( ) [inline, static]

Definition at line 74 of file search_graph.cc.

10.45.2.7 template<class T > static T coco::coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.45.2.8 template<class T > static T coco::coco::coco::checking_my< T >::inf ( ) [inline, static]

Definition at line 54 of file search_graph.cc.

10.45.2.9 template<class T > static bool coco::coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.45.2.10 template<class T > static bool coco::coco::coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 84 of file search_graph.cc.

10.45.2.11 template<class T > static bool coco::coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.45.2.12 template<class T > static bool coco::coco::coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 65 of file search_graph.cc.

10.45.2.13 template<class T > static T coco::coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.45.2.14 template<class T > static T coco::coco::coco::checking_my< T >::nan ( ) [inline, static]

Definition at line 60 of file search_graph.cc.

10.45.2.15 template<class T > static T coco::coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.45.2.16 template<class T > static T coco::coco::coco::checking_my< T >::neg_inf ( ) [inline, static]

Definition at line 53 of file search_graph.cc.

10.45.2.17 template<class T > static T coco::coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

10.45.2.18 template<class T > static T coco::coco::coco::checking_my< T >::pos_inf ( ) [inline, static]

Definition at line 52 of file search_graph.cc.

The documentation for this struct was generated from the following file:

- interval_boost.h

10.46 coco::expression_node::children_compare Class Reference

`#include <expression.hpp>`

Public Member Functions

- bool operator() (const expression_node &__x, const expression_node &__y) const

10.46.1 Member Function Documentation

10.46.1.1 bool coco::expression_node::children_compare::operator() (const expression_node & __x, const expression_node & __y) const [inline]

This is the evaluation operator of this function class.

Definition at line 93 of file expression.hpp.

The documentation for this class was generated from the following file:

- expression.hpp

10.47 coco::cinterval_eval Class Reference

`#include <cint_evaluator.h>`

Inheritance diagram for coco::cinterval_eval:

Collaboration diagram for coco::cinterval_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- cinterval_eval (const std::vector< cinterval > &__x, const variable_indicator &__v, const model &__m, std::vector< cinterval > *__c)
- cinterval_eval (const cinterval_eval &__v)
- ~cinterval_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const cinterval &__rval)
- int update (const expression_node &__data, const cinterval &__rval)
- cinterval calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)

- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.47.1 Member Typedef Documentation

10.47.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.47.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.47.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.47.2 Constructor & Destructor Documentation

10.47.2.1 coco::cinterval_eval::cinterval_eval (const std::vector< cinterval > & __x, const variable_indicator & __v, const model & __m, std::vector< cinterval > * __c) [inline]

Definition at line 116 of file cint_evaluator.h.

10.47.2.2 coco::cinterval_eval::cinterval_eval (const cinterval_eval & __v) [inline]

Definition at line 128 of file cint_evaluator.h.

10.47.2.3 coco::cinterval_eval::~cinterval_eval ( ) [inline]

Definition at line 130 of file cint_evaluator.h.

10.47.3 Member Function Documentation

10.47.3.1 void coco::cinterval_eval::calculate (const expression_node & __data) [inline]

Definition at line 193 of file cint_evaluator.h.

10.47.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.47.3.3 cinterval coco::cinterval_eval::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 667 of file cint_evaluator.h.

10.47.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.47.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.47.3.6 void coco::cinterval_eval::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 135 of file cint_evaluator.h.

10.47.3.7 int coco::cinterval_eval::initialize (const expression_node & __data) [inline]

Definition at line 137 of file cint_evaluator.h.

10.47.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.47.3.9 bool coco::cinterval_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 70 of file cint_evaluator.h.

10.47.3.10 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.47.3.11 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.47.3.12 void coco::cinterval_eval::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 203 of file cint_evaluator.h.

10.47.3.13 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.47.3.14 expression_const_walker coco::cinterval_eval::short_cut_to (const expression_node & __data) [inline]

Definition at line 132 of file cint_evaluator.h.

10.47.3.15 int coco::cinterval_eval::update (const cinterval & __rval) [inline]

Definition at line 219 of file cint_evaluator.h.

10.47.3.16 int coco::cinterval_eval::update (const expression_node & __data, const cinterval & __rval) [inline]

Definition at line 225 of file cint_evaluator.h.

10.47.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.47.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.47.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.47.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.47.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.47.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

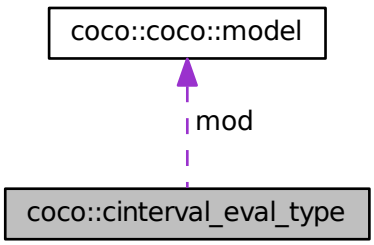
The documentation for this class was generated from the following file:

- cint_evaluator.h

10.48 coco::cinterval_eval_type Struct Reference

`#include <cint_evaluator.h>`

Collaboration diagram for coco::cinterval_eval_type:



Public Attributes

- const std::vector< cinterval > * x

- std::vector< cinterval > * cache
- const model * mod
- void * p
- cinterval d
- int info
- cinterval r
- unsigned int n

10.48.1 Member Data Documentation

10.48.1.1 std::vector<cinterval>* coco::cinterval_eval_type::cache

Definition at line 52 of file cint_evaluator.h.

10.48.1.2 cinterval coco::cinterval_eval_type::d

Definition at line 55 of file cint_evaluator.h.

10.48.1.3 int coco::cinterval_eval_type::info

Definition at line 56 of file cint_evaluator.h.

10.48.1.4 const model* coco::cinterval_eval_type::mod

Definition at line 53 of file cint_evaluator.h.

10.48.1.5 unsigned int coco::cinterval_eval_type::n

Definition at line 58 of file cint_evaluator.h.

10.48.1.6 void* coco::cinterval_eval_type::p

Definition at line 54 of file cint_evaluator.h.

10.48.1.7 cinterval coco::cinterval_eval_type::r

Definition at line 57 of file cint_evaluator.h.

10.48.1.8 const std::vector<cinterval>* coco::cinterval_eval_type::x

Definition at line 51 of file cint_evaluator.h.

The documentation for this struct was generated from the following file:

- cint_evaluator.h

10.49 coco::cmp_point_info_i Struct Reference

Public Member Functions

- bool operator() (const d1func::point_info &_a, const interval &_b) const

- bool operator() (const interval &_a, const d1func::point_info &_b) const
- bool operator() (const d1func::point_info &_a, double _b) const
- bool operator() (double _a, const d1func::point_info &_b) const
- bool operator() (const d1func::point_info &_a, const d1func::point_info &_b) const

10.49.1 Member Function Documentation

10.49.1.1 bool coco::cmp_point_info_i::operator() (const d1func::point_info & *_a*, const interval & *_b*) const [inline]

Definition at line 45 of file d1func.cc.

10.49.1.2 bool coco::cmp_point_info_i::operator() (const interval & *_a*, const d1func::point_info & *_b*) const [inline]

Definition at line 47 of file d1func.cc.

10.49.1.3 bool coco::cmp_point_info_i::operator() (const d1func::point_info & *_a*, double *_b*) const [inline]

Definition at line 49 of file d1func.cc.

10.49.1.4 bool coco::cmp_point_info_i::operator() (double *_a*, const d1func::point_info & *_b*) const [inline]

Definition at line 51 of file d1func.cc.

10.49.1.5 bool coco::cmp_point_info_i::operator() (const d1func::point_info & *_a*, const d1func::point_info & *_b*) const [inline]

Definition at line 53 of file d1func.cc.

The documentation for this struct was generated from the following file:

- d1func.cc

10.50 coco::cmp_point_info_s Struct Reference

Public Member Functions

- bool operator() (const d1func::point_info &_a, const interval &_b) const
- bool operator() (const interval &_a, const d1func::point_info &_b) const
- bool operator() (const d1func::point_info &_a, double _b) const
- bool operator() (double _a, const d1func::point_info &_b) const
- bool operator() (const d1func::point_info &_a, const d1func::point_info &_b) const

10.50.1 Member Function Documentation

10.50.1.1 bool coco::cmp_point_info_s::operator() (const d1func::point_info & _a, const interval & _b) const [inline]

Definition at line 61 of file d1func.cc.

10.50.1.2 bool coco::cmp_point_info_s::operator() (const interval & _a, const d1func::point_info & _b) const [inline]

Definition at line 63 of file d1func.cc.

10.50.1.3 bool coco::cmp_point_info_s::operator() (const d1func::point_info & _a, double _b) const [inline]

Definition at line 65 of file d1func.cc.

10.50.1.4 bool coco::cmp_point_info_s::operator() (double _a, const d1func::point_info & _b) const [inline]

Definition at line 67 of file d1func.cc.

10.50.1.5 bool coco::cmp_point_info_s::operator() (const d1func::point_info & _a, const d1func::point_info & _b) const [inline]

Definition at line 69 of file d1func.cc.

The documentation for this struct was generated from the following file:

- d1func.cc

10.51 coco::coconut_random_f Struct Reference

`#include <coconut_random.h>`

Public Member Functions

- interval operator() (const interval &i) const
- double operator() (double l, double u) const
- double operator() (double u) const
- double operator() () const
- int operator() (int n) const
- int operator() (long int n) const
- size_t operator() (size_t n) const
- coconut_random_f (double _b=10.0, double _a=0.9)

Public Attributes

- double beta
- double alpha

10.51.1 Detailed Description

This is a class which can be used as random number object in STL like algorithms.

10.51.2 Constructor & Destructor Documentation

10.51.2.1 coco::coconut_random_f::coconut_random_f (double _b = 10.0, double _a = 0.9) [inline]

Definition at line 275 of file coconut_random.h.

10.51.3 Member Function Documentation

10.51.3.1 interval coco::coconut_random_f::operator() (const interval & i) const [inline]

Definition at line 254 of file coconut_random.h.

10.51.3.2 double coco::coconut_random_f::operator() (double l, double u) const [inline]

Definition at line 257 of file coconut_random.h.

10.51.3.3 double coco::coconut_random_f::operator() (double u) const [inline]

Definition at line 260 of file coconut_random.h.

10.51.3.4 double coco::coconut_random_f::operator() ( ) const [inline]

Definition at line 263 of file coconut_random.h.

10.51.3.5 int coco::coconut_random_f::operator() (int n) const [inline]

Definition at line 266 of file coconut_random.h.

10.51.3.6 int coco::coconut_random_f::operator() (long int n) const [inline]

Definition at line 269 of file coconut_random.h.

10.51.3.7 size_t coco::coconut_random_f::operator() (size_t n) const [inline]

Definition at line 272 of file coconut_random.h.

10.51.4 Member Data Documentation

10.51.4.1 double coco::coconut_random_f::alpha

Definition at line 252 of file coconut_random.h.

10.51.4.2 double coco::coconut_random_f::beta

Definition at line 251 of file coconut_random.h.

The documentation for this struct was generated from the following file:

- coconut_random.h

10.52 coco::compare_intervals Class Reference

Public Types

- typedef interval first_argument_type
- typedef interval second_argument_type
- typedef bool result_type

Public Member Functions

- bool operator() (const interval &a, const interval &b)

10.52.1 Member Typedef Documentation

10.52.1.1 typedef interval coco::compare_intervals::first_argument_type

Definition at line 151 of file dagd1func_bp.cc.

10.52.1.2 typedef bool coco::compare_intervals::result_type

Definition at line 153 of file dagd1func_bp.cc.

10.52.1.3 typedef interval coco::compare_intervals::second_argument_type

Definition at line 152 of file dagd1func_bp.cc.

10.52.2 Member Function Documentation

10.52.2.1 bool coco::compare_intervals::operator() (const interval & *a*, const interval & *b*) [inline]

Definition at line 155 of file dagd1func_bp.cc.

The documentation for this class was generated from the following file:

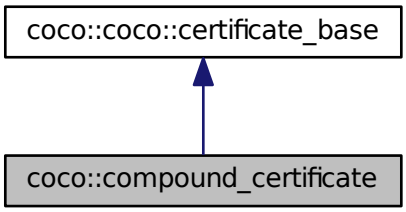
- dagd1func_bp.cc

10.53 coco::compound_certificate Class Reference

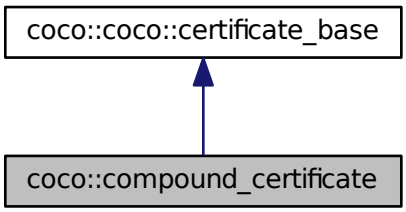
The certificate for deltas formed by compressing bound_delta entries.

`#include <api_cert.h>`

Inheritance diagram for coco::compound_certificate:



Collaboration diagram for coco::compound_certificate:



Public Member Functions

- compound_certificate (bool verified)
- compound_certificate (const compound_certificate &__c)
- virtual ~compound_certificate ()
- compound_certificate * new_copy () const
- void destroy_copy (certificate_base *__c) const
- bool verify (const work_node &_x) const
- bool operator== (const certificate_base &_c) const
- bool operator!= (const certificate_base &_c) const

- bool operator== (const compound_certificate &_c) const
- bool operator!= (const compound_certificate &_c) const
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- virtual std::string get_contents () const
- virtual std::string get_contents () const
- virtual std::string get_contents () const
- virtual std::string get_contents () const

Protected Attributes

- std::string _contents

10.53.1 Detailed Description

The compound certificate for compound deltas generated by management modules

10.53.2 Constructor & Destructor Documentation

10.53.2.1 coco::compound_certificate::compound_certificate (bool *verified*) [inline]

Constructor setting the `is_verfied` information to `verified`

Definition at line 210 of file api_cert.h.

10.53.2.2 coco::compound_certificate::compound_certificate (const compound_certificate & *__c*) [inline]

Standard Copy Constructor

Definition at line 213 of file api_cert.h.

10.53.2.3 virtual coco::compound_certificate::~compound_certificate ( ) [inline, virtual]

Standard Destructor

Definition at line 222 of file api_cert.h.

10.53.3 Member Function Documentation

10.53.3.1 void coco::compound_certificate::destroy_copy (certificate_base * *__c*) const [inline]

Clone Destructor

Definition at line 233 of file api_cert.h.

10.53.3.2 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.53.3.3 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.53.3.4 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.53.3.5 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.53.3.6 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents `c`.

Definition at line 156 of file search_graph.cc.

10.53.3.7 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents `c`.

Definition at line 156 of file search_graph.cc.

10.53.3.8 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents `c`.

Definition at line 156 of file search_graph.cc.

10.53.3.9 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.53.3.10 compound_certificate* coco::compound_certificate::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::certificate_base.

Definition at line 225 of file api_cert.h.

10.53.3.11 bool coco::compound_certificate::operator!= (const certificate_base & *_c*) const [inline]

Definition at line 245 of file api_cert.h.

10.53.3.12 bool coco::compound_certificate::operator!= (const compound_certificate & *_c*) const [inline]

Definition at line 252 of file api_cert.h.

10.53.3.13 bool coco::compound_certificate::operator== (const certificate_base & *_c*) const [inline]

Comparison operators

Definition at line 242 of file api_cert.h.

10.53.3.14 bool coco::compound_certificate::operator== (const compound_certificate & *_c*) const [inline]

Comparison operators

Definition at line 250 of file api_cert.h.

10.53.3.15 bool coco::compound_certificate::verify (const work_node & *_x*) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. In this case the value is recorded in the is_verified member.

Reimplemented from coco::coco::certificate_base.

Definition at line 239 of file api_cert.h.

10.53.4 Member Data Documentation

10.53.4.1 std::string coco::coco::certificate_base::_contents [protected, inherited]

The contents (descriptive string, type) of this certificate

Definition at line 128 of file search_graph.cc.

The documentation for this class was generated from the following file:

- api_cert.h

10.54 coco::coco::work_node::constraint_iterator_base Class Reference

The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

Public Types

- typedef std::bidirectional_iterator_tag iterator_category
- typedef _TW value_type
- typedef _TR reference
- typedef _TP pointer
- typedef size_t size_type
- typedef ptrdiff_t difference_type

Public Member Functions

- constraint_iterator_base ()
- constraint_iterator_base (_Reference cs, unsigned int tp)
- constraint_iterator_base (const _Self &__c)
- ~constraint_iterator_base ()
- reference operator* () const
- pointer operator-> () const
- bool operator== (const _Self &_c)
- bool operator!= (const _Self &_c)
- _Self & set (const _Iterator &_c)
- _Self & operator++ ()
- _Self operator++ (int)
- _Self & operator-- ()
- _Self operator-- (int)
- _Self & operator= (const _Self &_c)

10.54.1 Detailed Description

This is the base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

See also

work_node::get_begin

10.54.2 Member Typedef Documentation

10.54.2.1 typedef ptrdiff_t coco::coco::work_node::constraint_iterator_base::difference_type

These types are needed for an iterator.

Definition at line 849 of file search_graph.cc.

10.54.2.2 typedef std::bidirectional_iterator_tag coco::coco::work_node::constraint_iterator_base::iterator_category

These types are needed for an iterator.

Definition at line 842 of file search_graph.cc.

10.54.2.3 typedef _TP coco::coco::work_node::constraint_iterator_base::pointer

These types are needed for an iterator.

Definition at line 846 of file search_graph.cc.

10.54.2.4 typedef _TR coco::coco::work_node::constraint_iterator_base::reference

These types are needed for an iterator.

Definition at line 845 of file search_graph.cc.

10.54.2.5 typedef size_t coco::coco::work_node::constraint_iterator_base::size_type

These types are needed for an iterator.

Definition at line 848 of file search_graph.cc.

10.54.2.6 typedef _TW coco::coco::work_node::constraint_iterator_base::value_type

These types are needed for an iterator.

Definition at line 844 of file search_graph.cc.

10.54.3 Constructor & Destructor Documentation

10.54.3.1 coco::coco::work_node::constraint_iterator_base::constraint_iterator_base ( ) [inline]

Standard Constructor

Definition at line 854 of file search_graph.cc.

10.54.3.2 coco::coco::work_node::constraint_iterator_base::constraint_iterator_base (_Reference *cs*, unsigned int *tp*) [inline]

Constructor, which initializes the vector with `cs` and the type with `tp`.

Definition at line 857 of file search_graph.cc.

10.54.3.3 coco::coco::work_node::constraint_iterator_base::constraint_iterator_base (const _Self & __c) [inline]

Standard Copy Constructor

Definition at line 860 of file search_graph.cc.

10.54.3.4 coco::coco::work_node::constraint_iterator_base::~constraint_iterator_base ( ) [inline]

Standard Destructor

Definition at line 864 of file search_graph.cc.

10.54.4 Member Function Documentation

10.54.4.1 bool coco::coco::work_node::constraint_iterator_base::operator!= (const _Self & _c) [inline]

Disequality Comparison Operator

Definition at line 876 of file search_graph.cc.

10.54.4.2 reference coco::coco::work_node::constraint_iterator_base::operator* ( ) const [inline]

Dereference Operator

Definition at line 867 of file search_graph.cc.

10.54.4.3 work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self & coco::work_node::constraint_iterator_base::operator++ ( ) [inline]

Increment operator: go to the next constraint of the specified type

Definition at line 581 of file search_node.hpp.

10.54.4.4 work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self coco::work_node::constraint_iterator_base::operator++ (int) [inline]

Increment operator: go to the next constraint of the specified type

Definition at line 596 of file search_node.hpp.

10.54.4.5 work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self & coco::work_node::constraint_iterator_base::operator-- ( ) [inline]

Decrement operator: go to the previous constraint of the specified type

Definition at line 606 of file search_node.hpp.

10.54.4.6 work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self coco::work_node::constraint_iterator_base::operator-- (int) [inline]

Decrement operator: go to the previous constraint of the specified type

Definition at line 625 of file search_node.hpp.

10.54.4.7 pointer coco::coco::work_node::constraint_iterator_base::operator-> ( ) const [inline]

Arrow Operator

Definition at line 869 of file search_graph.cc.

10.54.4.8 work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self & coco::work_node::constraint_iterator_base::operator= (const _Self & *_c*) [inline]

Standard Assignment Operator

Definition at line 635 of file search_node.hpp.

10.54.4.9 bool coco::coco::work_node::constraint_iterator_base::operator== (const _Self & *_c*) [inline]

Equality Comparison Operator

Definition at line 872 of file search_graph.cc.

10.54.4.10 work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self & coco::work_node::constraint_iterator_base::set (const _Iterator & *_c*) [inline]

This method sets the constraint_iterator to the first constraint of the specified type, which is not before _c.

Definition at line 567 of file search_node.hpp.

The documentation for this class was generated from the following files:

- search_node.h
- search_node.hpp

10.55 coco::work_node::constraint_iterator_base Class Reference

The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

`#include <search_node.h>`

Public Types

- typedef std::bidirectional_iterator_tag iterator_category
- typedef _TW value_type
- typedef _TR reference
- typedef _TP pointer
- typedef size_t size_type
- typedef ptrdiff_t difference_type

Public Member Functions

- constraint_iterator_base ()
- constraint_iterator_base (_Reference cs, unsigned int tp)
- constraint_iterator_base (const _Self &__c)
- ~constraint_iterator_base ()
- reference operator* () const
- pointer operator-> () const
- bool operator== (const _Self &_c)
- bool operator!= (const _Self &_c)
- _Self & set (const _Iterator &_c)
- _Self & operator++ ()
- _Self operator++ (int)
- _Self & operator-- ()
- _Self operator-- (int)
- _Self & operator= (const _Self &_c)

10.55.1 Detailed Description

This is the base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

See also

work_node::get_begin

10.55.2 Member Typedef Documentation

10.55.2.1 typedef ptrdiff_t coco::work_node::constraint_iterator_base::difference_type

These types are needed for an iterator.

Definition at line 849 of file search_node.h.

10.55.2.2 typedef std::bidirectional_iterator_tag coco::work_node::constraint_iterator_base::iterator_category

These types are needed for an iterator.

Definition at line 842 of file search_node.h.

10.55.2.3 typedef _TP coco::work_node::constraint_iterator_base::pointer

These types are needed for an iterator.

Definition at line 846 of file search_node.h.

10.55.2.4 typedef _TR coco::work_node::constraint_iterator_base::reference

These types are needed for an iterator.

Definition at line 845 of file search_node.h.

10.55.2.5 typedef size_t coco::work_node::constraint_iterator_base::size_type

These types are needed for an iterator.

Definition at line 848 of file search_node.h.

10.55.2.6 typedef _TW coco::work_node::constraint_iterator_base::value_type

These types are needed for an iterator.

Definition at line 844 of file search_node.h.

10.55.3 Constructor & Destructor Documentation

10.55.3.1 coco::work_node::constraint_iterator_base::constraint_iterator_base ( ) [inline]

Standard Constructor

Definition at line 854 of file search_node.h.

10.55.3.2 coco::work_node::constraint_iterator_base::constraint_iterator_base (_Reference *cs,* unsigned int *tp*) [inline]

Constructor, which initializes the vector with `cs` and the type with `tp`.

Definition at line 857 of file search_node.h.

10.55.3.3 coco::work_node::constraint_iterator_base::constraint_iterator_base (const _Self & *__c*) [inline]

Standard Copy Constructor

Definition at line 860 of file search_node.h.

10.55.3.4 coco::work_node::constraint_iterator_base::~constraint_iterator_base ( ) [inline]

Standard Destructor

Definition at line 864 of file search_node.h.

10.55.4 Member Function Documentation

10.55.4.1 bool coco::work_node::constraint_iterator_base::operator!= (const _Self & *_c*) [inline]

Disequality Comparison Operator

Definition at line 876 of file search_node.h.

10.55.4.2 reference coco::work_node::constraint_iterator_base::operator* ( ) const [inline]

Dereference Operator

Definition at line 867 of file search_node.h.

10.55.4.3 _Self& coco::work_node::constraint_iterator_base::operator++ ( )

Increment operator: go to the next constraint of the specified type

10.55.4.4 _Self coco::work_node::constraint_iterator_base::operator++ (int)

Increment operator: go to the next constraint of the specified type

10.55.4.5 _Self& coco::work_node::constraint_iterator_base::operator-- ( )

Decrement operator: go to the previous constraint of the specified type

10.55.4.6 _Self coco::work_node::constraint_iterator_base::operator-- (int)

Decrement operator: go to the previous constraint of the specified type

10.55.4.7 pointer coco::work_node::constraint_iterator_base::operator-> ( ) const [inline]

Arrow Operator

Definition at line 869 of file search_node.h.

10.55.4.8 _Self& coco::work_node::constraint_iterator_base::operator= (const _Self & *_c*)

Standard Assignment Operator

10.55.4.9 bool coco::work_node::constraint_iterator_base::operator== (const _Self & *_c*) [inline]

Equality Comparison Operator

Definition at line 872 of file search_node.h.

10.55.4.10 _Self& coco::work_node::constraint_iterator_base::set (const _Iterator & *_c*)

This method sets the constraint_iterator to the first constraint of the specified type, which is not before _c.

The documentation for this class was generated from the following file:

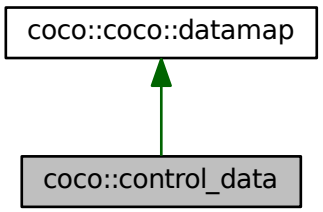
- search_node.h

10.56 coco::control_data Class Reference

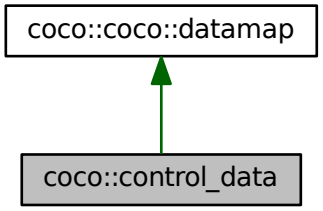
The class for communicating parameter information to COCONUT modules.

`#include <control_data.h>`

Inheritance diagram for coco::control_data:



Collaboration diagram for coco::control_data:



Public Member Functions

- control_data ()
- control_data (const std::string &__n, const basic_alltype &__v)
- control_data (const char *__n, const basic_alltype &__v)
- control_data (const control_data &__c)
- control_data (const std::string &serv, const info_contents &inf)
- virtual ~control_data ()
- control_data & operator= (const control_data &__c)
- control_data & operator= (const info_contents &__i)
- void service (const std::string &__s)
- void service (const char *__s)
- const std::string & service () const throw (api_exception)
- bool check_service (const std::string &__n) const
- bool check_service (const char *__n) const

- void set (const info_contents &__i)
- void list (std::vector< std::string > &__v) const

- void set (const std::string &__s, const basic_alltype &__h)
- void set (const char *__cp, const basic_alltype &__h)

- void set (const std::string &__s, int i, const basic_alltype &__h)
- void set (const char *__cp, int i, const basic_alltype &__h)

- const basic_alltype & get (const std::string &__s) const
- const basic_alltype & get (const char *__cp) const

- const basic_alltype & get (const std::string &__s, int i) const
- const basic_alltype & get (const char *__cp, int i) const

- void unset (const std::string &__s)
- void unset (const char *__cp)

- void unset (const std::string &__s, int i)
- void unset (const char *__cp, int i)

- bool is_set (const std::string &__s) const
- bool is_set (const char *__cp) const

- bool is_set (const std::string &__s, int i) const
- bool is_set (const char *__cp, int i) const

- bool which_set (const std::string &__s, std::vector< int > &__idx) const
- bool which_set (const char *__cp, std::vector< int > &__idx) const

- template<class _TS >
  void assign (const std::string &__s, const std::vector< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign (const std::string &__s, const vmtl::sparse_vector< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign (const std::string &__s, const vmtl::dense_matrix< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign (const std::string &__s, const vmtl::sparse_matrix< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign (const std::string &__s, _TS &__b) const throw (api_exception)
- template<class _TS >
  void assign (const char *__s, const std::vector< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign (const char *__s, const vmtl::sparse_vector< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign (const char *__s, const vmtl::dense_matrix< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign (const char *__s, const vmtl::sparse_matrix< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign (const char *__s, _TS &__b) const throw (api_exception)

- void assign (const std::string &__s, std::string &__b, const char *__def) const
- void assign (const char *__s, std::string &__b, const char *__def) const
- template<class _TS >
  void assign (const std::string &__s, const std::vector< _TS > *&__b, const std::vector< _TS > *__def) const
- template<class _TS >
  void assign (const std::string &__s, const vmtl::sparse_vector< _TS > *&__b, const vmtl::sparse_vector< _TS > *__def) const
- template<class _TS >
  void assign (const std::string &__s, const vmtl::dense_matrix< _TS > *&__b, const vmtl::dense_matrix< _TS > *__def) const
- template<class _TS >
  void assign (const std::string &__s, const vmtl::sparse_matrix< _TS > *&__b, const vmtl::sparse_matrix< _TS > *__def) const
- template<class _TS >
  void assign (const std::string &__s, _TS &__b, _TS __def) const
- template<class _TS >
  void assign (const char *__s, const std::vector< _TS > *&__b, const std::vector< _TS > *__def) const
- template<class _TS >
  void assign (const char *__s, const vmtl::sparse_vector< _TS > *&__b, const vmtl::sparse_vector< _TS > *__def) const
- template<class _TS >
  void assign (const char *__s, const vmtl::dense_matrix< _TS > *&__b, const vmtl::dense_matrix< _TS > *__def) const

- template<class _TS >
  void assign (const char *__s, const vmtl::sparse_matrix< _TS > *&__b, const vmtl::sparse_matrix< _TS > *__def) const
- template<class _TS >
  void assign (const char *__s, _TS &__b, _TS __def) const

- template<class _TS >
  void assign_i (const std::string &__s, int i, const std::vector< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const std::string &__s, int i, const vmtl::sparse_vector< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const std::string &__s, int i, const vmtl::dense_matrix< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const std::string &__s, int i, const vmtl::sparse_matrix< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const std::string &__s, int i, _TS &__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const char *__s, int i, const std::vector< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const char *__s, int i, const vmtl::sparse_vector< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const char *__s, int i, const vmtl::dense_matrix< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const char *__s, int i, const vmtl::sparse_matrix< _TS > *&__b) const throw (api_exception)
- template<class _TS >
  void assign_i (const char *__s, int i, _TS &__b) const throw (api_exception)

- void assign_i (const std::string &__s, int i, std::string &__b, const char *__def) const
- void assign_i (const char *__s, int i, std::string &__b, const char *__def) const
- template<class _TS >
  void assign_i (const std::string &__s, int i, const std::vector< _TS > *&__b, const std::vector< _TS > *__def) const
- template<class _TS >
  void assign_i (const std::string &__s, int i, const vmtl::sparse_vector< _TS > *&__b, const vmtl::sparse_vector< _TS > *__def) const
- template<class _TS >
  void assign_i (const std::string &__s, int i, const vmtl::dense_matrix< _TS > *&__b, const vmtl::dense_matrix< _TS > *__def) const
- template<class _TS >
  void assign_i (const std::string &__s, int i, const vmtl::sparse_matrix< _TS > *&__b, const vmtl::sparse_matrix< _TS > *__def) const

- template<class _TS >
  void assign_i (const std::string &__s, int i, _TS &__b, _TS __def) const
- template<class _TS >
  void assign_i (const char *__s, int i, const std::vector< _TS > *&__b, const std::vector< _TS > *__def) const
- template<class _TS >
  void assign_i (const char *__s, int i, const vmtl::sparse_vector< _TS > *&__b, const vmtl::sparse_vector< _TS > *__def) const
- template<class _TS >
  void assign_i (const char *__s, int i, const vmtl::dense_matrix< _TS > *&__b, const vmtl::dense_matrix< _TS > *__def) const
- template<class _TS >
  void assign_i (const char *__s, int i, const vmtl::sparse_matrix< _TS > *&__b, const vmtl::sparse_matrix< _TS > *__def) const
- template<class _TS >
  void assign_i (const char *__s, int i, _TS &__b, _TS __def) const

Protected Member Functions

- bool sinsert (const std::string &__s, const basic_alltype &__h, bool replace)
- bool sinsert (const char *__s, const basic_alltype &__h, bool replace)

- bool sinsert (const std::string &__s, int i, const basic_alltype &__h, bool replace)
- bool sinsert (const char *__s, int i, const basic_alltype &__h, bool replace)

- const basic_alltype & sfind (const std::string &__s) const
- const basic_alltype & sfind (const char *__s) const

- const basic_alltype & sfind (const std::string &__s, int i) const
- const basic_alltype & sfind (const char *__s, int i) const

- void remove (const std::string &__s)
- void remove (const char *__s)

- void remove (const std::string &__s, int i)
- void remove (const char *__s, int i)

- bool defd (const std::string &__s) const
- bool defd (const char *__s) const

- bool defd (const std::string &__s, int i) const
- bool defd (const char *__s, int i) const

- bool which (const std::string &__s, std::vector< int > &__idx) const
- bool which (const char *__s, std::vector< int > &__idx) const

- bool retrieve (const std::string &__s, bool &__v) const
- bool retrieve (const std::string &__s, int &__v) const
- bool retrieve (const std::string &__s, unsigned int &__v) const
- bool retrieve (const std::string &__s, double &__v) const
- bool retrieve (const std::string &__s, interval &__v) const
- bool retrieve (const std::string &__s, std::string &__is) const
- bool retrieve (const std::string &__s, num::Number &__v) const
- bool retrieve (const std::string &__s, const std::vector< bool > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< int > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< unsigned int > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< double > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< interval > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< std::string > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< num::Number > *&__v) const
- bool retrieve (const char *__s, bool &__v) const
- bool retrieve (const char *__s, int &__v) const
- bool retrieve (const char *__s, unsigned int &__v) const
- bool retrieve (const char *__s, double &__v) const
- bool retrieve (const char *__s, interval &__v) const
- bool retrieve (const char *__s, std::string &__is) const
- bool retrieve (const char *__s, num::Number &__is) const
- bool retrieve (const char *__s, const std::vector< bool > *&__v) const

- bool retrieve (const char *__s, const std::vector< unsigned int > *&__v) const
- bool retrieve (const char *__s, const std::vector< int > *&__v) const
- bool retrieve (const char *__s, const std::vector< double > *&__v) const
- bool retrieve (const char *__s, const std::vector< interval > *&__v) const
- bool retrieve (const char *__s, const std::vector< std::string > *&__v) const
- bool retrieve (const char *__s, const std::vector< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< num::Number > *&__v) const

- bool retrieve (const std::string &__s, bool &__v, bool __def) const
- bool retrieve (const std::string &__s, int &__v, int __def) const
- bool retrieve (const std::string &__s, unsigned int &__v, unsigned int __def) const
- bool retrieve (const std::string &__s, double &__v, double __def) const
- bool retrieve (const std::string &__s, interval &__v, const interval &__def) const
- bool retrieve (const std::string &__s, std::string &__is, const std::string &__def) const
- bool retrieve (const std::string &__s, num::Number &__is, const num::Number &__def) const
- bool retrieve (const std::string &__s, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve (const std::string &__s, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve (const std::string &__s, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve (const std::string &__s, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve (const std::string &__s, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve (const std::string &__s, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve (const std::string &__s, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const

- bool retrieve (const std::string &__s, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const
- bool retrieve (const char *__s, bool &__v, bool __def) const
- bool retrieve (const char *__s, int &__v, int __def) const
- bool retrieve (const char *__s, unsigned int &__v, unsigned int __def) const
- bool retrieve (const char *__s, double &__v, double __def) const
- bool retrieve (const char *__s, interval &__v, const interval &__def) const
- bool retrieve (const char *__s, std::string &__v, const std::string &__def) const
- bool retrieve (const char *__s, num::Number &__v, const num::Number &__def) const
- bool retrieve (const char *__s, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve (const char *__s, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve (const char *__s, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve (const char *__s, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve (const char *__s, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve (const char *__s, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve (const char *__s, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const

- bool retrieve (const char *__s, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const

- bool retrieve_i (const std::string &__s, int i, bool &__v) const
- bool retrieve_i (const std::string &__s, int i, int &__v) const
- bool retrieve_i (const std::string &__s, int i, unsigned int &__v) const
- bool retrieve_i (const std::string &__s, int i, double &__v) const
- bool retrieve_i (const std::string &__s, int i, interval &__v) const
- bool retrieve_i (const std::string &__s, int i, std::string &__v) const
- bool retrieve_i (const std::string &__s, int i, num::Number &__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< bool > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< unsigned int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< num::Number > *&__v) const

- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, bool &__v) const
- bool retrieve_i (const char *__s, int i, int &__v) const
- bool retrieve_i (const char *__s, int i, unsigned int &__v) const
- bool retrieve_i (const char *__s, int i, double &__v) const
- bool retrieve_i (const char *__s, int i, interval &__v) const
- bool retrieve_i (const char *__s, int i, std::string &__is) const
- bool retrieve_i (const char *__s, int i, num::Number &__is) const
- bool retrieve_i (const char *__s, int i, const std::vector< bool > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< unsigned int > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< num::Number > *&__v) const

- bool retrieve_i (const std::string &__s, int i, bool &__v, bool __def) const
- bool retrieve_i (const std::string &__s, int i, int &__v, int __def) const
- bool retrieve_i (const std::string &__s, int i, unsigned int &__v, unsigned int __def) const
- bool retrieve_i (const std::string &__s, int i, double &__v, double __def) const
- bool retrieve_i (const std::string &__s, int i, interval &__v, const interval &__def) const
- bool retrieve_i (const std::string &__s, int i, std::string &__is, const std::string &__def) const
- bool retrieve_i (const std::string &__s, int i, num::Number &__is, const num::Number &__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const

- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, bool &__v, bool __def) const
- bool retrieve_i (const char *__s, int i, int &__v, int __def) const
- bool retrieve_i (const char *__s, int i, unsigned int &__v, unsigned int __def) const
- bool retrieve_i (const char *__s, int i, double &__v, double __def) const
- bool retrieve_i (const char *__s, int i, interval &__v, const interval &__def) const
- bool retrieve_i (const char *__s, int i, std::string &__v, const std::string &__def) const
- bool retrieve_i (const char *__s, int i, num::Number &__v, const num::Number &__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const

- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const

10.56.1 Detailed Description

This class is the main vehicle for communicating service information and module parameters to all COCONUT modules. It is a datamap (i.e. a map of strings (parameter names) to basic_alltype values (the parameter values) with the addition of a service identifier.

10.56.2 Constructor & Destructor Documentation

10.56.2.1 coco::control_data::control_data ( ) [inline]

Standard Constructor

Definition at line 55 of file control_data.h.

10.56.2.2 coco::control_data::control_data (const std::string & *__n*, const basic_alltype & *__v*) [inline]

Constructor, which sets the parameter with name __n to value __v.

Definition at line 58 of file control_data.h.

10.56.2.3 coco::control_data::control_data (const char * *__n*, const basic_alltype & *__v*) [inline]

Constructor, which sets the parameter with name __n to value __v.

Definition at line 62 of file control_data.h.

10.56.2.4 coco::control_data::control_data (const control_data & *__c*) [inline]

Constructor, which sets the service to __c.

Definition at line 65 of file control_data.h.

10.56.2.5 coco::control_data::control_data (const std::string & *serv*, const info_contents & *inf*) [inline]

Constructor, which sets the service to serv and initializes the parameter structure from the info_contents inf.

Definition at line 68 of file control_data.h.

10.56.2.6 virtual coco::control_data::~control_data ( ) [inline, virtual]

Standard Destructor

Definition at line 71 of file control_data.h.

10.56.3 Member Function Documentation

10.56.3.1 template<class _TS > void coco::control_data::assign (const std::string & __s, const std::vector< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 67 of file control_data.hpp.

10.56.3.2 template<class _TS > void coco::control_data::assign (const std::string & __s, const vmtl::sparse_vector< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 80 of file control_data.hpp.

10.56.3.3 template<class _TS > void coco::control_data::assign (const std::string & __s, const vmtl::dense_matrix< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 93 of file control_data.hpp.

10.56.3.4 template<class _TS > void coco::control_data::assign (const std::string & __s, const vmtl::sparse_matrix< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 106 of file control_data.hpp.

10.56.3.5 template<class _TS > void coco::control_data::assign (const std::string & __s, _TS & __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly

into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 119 of file control_data.hpp.

10.56.3.6 template<class _TS > void coco::control_data::assign (const char * __s, const std::vector< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 176 of file control_data.hpp.

10.56.3.7 template<class _TS > void coco::control_data::assign (const char * __s, const vmtl::sparse_vector< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 184 of file control_data.hpp.

10.56.3.8 template<class _TS > void coco::control_data::assign (const char * __s, const vmtl::dense_matrix< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 192 of file control_data.hpp.

10.56.3.9 template<class _TS > void coco::control_data::assign (const char * __s, const vmtl::sparse_matrix< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 200 of file control_data.hpp.

10.56.3.10 template<class _TS > void coco::control_data::assign (const char * __s, _TS & __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 208 of file control_data.hpp.

10.56.3.11 void coco::control_data::assign (const std::string & __s, std::string & __b, const char * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 130 of file control_data.hpp.

10.56.3.12 void coco::control_data::assign (const char * __s, std::string & __b, const char * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 214 of file control_data.hpp.

10.56.3.13 template<class _TS > void coco::control_data::assign (const std::string & __s, const std::vector< _TS > *& __b, const std::vector< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 137 of file control_data.hpp.

10.56.3.14 template<class _TS > void coco::control_data::assign (const std::string & __s, const vmtl::sparse_vector< _TS > *& __b, const vmtl::sparse_vector< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 145 of file control_data.hpp.

10.56.3.15 template<class _TS > void coco::control_data::assign (const std::string & __s, const vmtl::dense_matrix< _TS > *& __b, const vmtl::dense_matrix< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 153 of file control_data.hpp.

10.56.3.16 template<class _TS > void coco::control_data::assign (const std::string & __s, const vmtl::sparse_matrix< _TS > *& __b, const vmtl::sparse_matrix< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 161 of file control_data.hpp.

10.56.3.17 template<class _TS > void coco::control_data::assign (const std::string & __s, _TS & __b, _TS __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 169 of file control_data.hpp.

10.56.3.18 template<class _TS > void coco::control_data::assign (const char * __s, const std::vector< _TS > *& __b, const std::vector< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 221 of file control_data.hpp.

10.56.3.19 template<class _TS > void coco::control_data::assign (const char * __s, const vmtl::sparse_vector< _TS > *& __b, const vmtl::sparse_vector< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 228 of file control_data.hpp.

10.56.3.20 template<class _TS > void coco::control_data::assign (const char * __s, const vmtl::dense_matrix< _TS > *& __b, const vmtl::dense_matrix< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 236 of file control_data.hpp.

10.56.3.21 template<class _TS > void coco::control_data::assign (const char * __s, const vmtl::sparse_matrix< _TS > *& __b, const vmtl::sparse_matrix< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 244 of file control_data.hpp.

10.56.3.22 template<class _TS > void coco::control_data::assign (const char * __s, _TS & __b, _TS __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 252 of file control_data.hpp.

10.56.3.23 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, const std::vector< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 259 of file control_data.hpp.

10.56.3.24 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, const vmtl::sparse_vector< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 272 of file control_data.hpp.

10.56.3.25 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, const vmtl::dense_matrix< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 285 of file control_data.hpp.

10.56.3.26 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, const vmtl::sparse_matrix< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 298 of file control_data.hpp.

10.56.3.27 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, _TS & __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 311 of file control_data.hpp.

10.56.3.28 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, const std::vector< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 368 of file control_data.hpp.

10.56.3.29 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, const vmtl::sparse_vector< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 376 of file control_data.hpp.

10.56.3.30 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, const vmtl::dense_matrix< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 384 of file control_data.hpp.

10.56.3.31 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, const vmtl::sparse_matrix< _TS > *& __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set

pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 392 of file control_data.hpp.

10.56.3.32 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, _TS & __b) const throw (api_exception) [inline]

These assign methods assign the value of the **required** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the required parameter is not set an api_exception of type apiee_communication_data is thrown.

Definition at line 400 of file control_data.hpp.

10.56.3.33 void coco::control_data::assign_i (const std::string & __s, int i, std::string & __b, const char * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 322 of file control_data.hpp.

10.56.3.34 void coco::control_data::assign_i (const char * __s, int i, std::string & __b, const char * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 406 of file control_data.hpp.

10.56.3.35 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, const std::vector< _TS > *& __b, const std::vector< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 329 of file control_data.hpp.

10.56.3.36 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, const vmtl::sparse_vector< _TS > *& __b, const vmtl::sparse_vector< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 337 of file control_data.hpp.

10.56.3.37 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, const vmtl::dense_matrix< _TS > *& __b, const vmtl::dense_matrix< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 345 of file control_data.hpp.

10.56.3.38 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, const vmtl::sparse_matrix< _TS > *& __b, const vmtl::sparse_matrix< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 353 of file control_data.hpp.

10.56.3.39 template<class _TS > void coco::control_data::assign_i (const std::string & __s, int i, _TS & __b, _TS __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 361 of file control_data.hpp.

10.56.3.40 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, const std::vector< _TS > *& __b, const std::vector< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 413 of file control_data.hpp.

10.56.3.41 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, const vmtl::sparse_vector< _TS > *& __b, const vmtl::sparse_vector< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name ___s and index i to the variable ___b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value ___def is assigned to ___b.

Definition at line 421 of file control_data.hpp.

10.56.3.42 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, const vmtl::dense_matrix< _TS > *& __b, const vmtl::dense_matrix< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s and index i to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 429 of file control_data.hpp.

10.56.3.43 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, const vmtl::sparse_matrix< _TS > *& __b, const vmtl::sparse_matrix< _TS > * __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s and index i to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 437 of file control_data.hpp.

10.56.3.44 template<class _TS > void coco::control_data::assign_i (const char * __s, int i, _TS & __b, _TS __def) const [inline]

These assign methods assign the value of the **optional** parameter with name __s and index i to the variable __b. Note that scalar variables are assigned but for non-scalar variables rather a pointer is set pointing directly into the control data structure to avoid unnecessary big copy operations. If the optional parameter is not set the default value __def is assigned to __b.

Definition at line 445 of file control_data.hpp.

10.56.3.45 bool coco::control_data::check_service (const std::string & __n) const [inline]

This method checks whether the service is set to __n.

Definition at line 51 of file control_data.hpp.

10.56.3.46 bool coco::control_data::check_service (const char * __n) const [inline]

This method checks whether the service is set to __n.

Definition at line 56 of file control_data.hpp.

10.56.3.47 bool coco::datamap::defd (const std::string & __s) const [inline, inherited]

This method returns whether the variable with name __s is defined.

Definition at line 109 of file datamap.hpp.

10.56.3.48 bool coco::datamap::defd (const char * __s) const [inline, inherited]

This method returns whether the variable with name __s is defined.

Definition at line 114 of file datamap.hpp.

10.56.3.49 bool coco::datamap::defd (const std::string & __s, int i) const [inline, inherited]

This method returns whether the variable with name ___s and index is defined.

Definition at line 119 of file datamap.hpp.

10.56.3.50 bool coco::datamap::defd (const char * __s, int i) const [inline, inherited]

This method returns whether the variable with name ___s and index is defined.

Definition at line 124 of file datamap.hpp.

10.56.3.51 const basic_alltype& coco::control_data::get (const std::string & __s) const [inline]

These methods retrieve the value of the parameter with name ___s.

Definition at line 132 of file control_data.h.

10.56.3.52 const basic_alltype& coco::control_data::get (const char * __cp) const [inline]

These methods retrieve the value of the parameter with name ___s.

Definition at line 134 of file control_data.h.

10.56.3.53 const basic_alltype& coco::control_data::get (const std::string & __s, int i) const [inline]

These methods retrieve the value of the parameter with name ___s and index i.

Definition at line 140 of file control_data.h.

10.56.3.54 const basic_alltype& coco::control_data::get (const char * __cp, int i) const [inline]

These methods retrieve the value of the parameter with name ___s and index i.

Definition at line 142 of file control_data.h.

10.56.3.55 bool coco::control_data::is_set (const std::string & __s) const [inline]

These methods check whether the parameter with name ___s is defined.

Definition at line 170 of file control_data.h.

10.56.3.56 bool coco::control_data::is_set (const char * __cp) const [inline]

These methods check whether the parameter with name ___s is defined.

Definition at line 171 of file control_data.h.

10.56.3.57 bool coco::control_data::is_set (const std::string & __s, int i) const [inline]

These methods check whether the parameter with name ___s and index i is defined.

Definition at line 176 of file control_data.h.

10.56.3.58 bool coco::control_data::is_set (const char * __cp, int i) const [inline]

These methods check whether the parameter with name ___s and index i is defined.

Definition at line 177 of file control_data.h.

10.56.3.59 void coco::control_data::list (std::vector< std::string > & __v) const [inline]

The list method sets ___v to the list of entry keys stored in this control data structure.

Reimplemented from coco::coco::datamap.

Definition at line 61 of file control_data.hpp.

10.56.3.60 control_data& coco::control_data::operator= (const control_data & __c) [inline]

Standard Assignment Operator

Definition at line 74 of file control_data.h.

10.56.3.61 control_data& coco::control_data::operator= (const info_contents & __i) [inline]

Assignment Operator, which assigns the parameter structure from an info_contents structure.

Definition at line 83 of file control_data.h.

10.56.3.62 void coco::datamap::remove (const std::string & __s) [inline, inherited]

The remove methods remove the variable with name ___s from the datamap.

Definition at line 57 of file datamap.hpp.

10.56.3.63 void coco::datamap::remove (const char * __s) [inline, inherited]

The remove methods remove the variable with name ___s from the datamap.

Definition at line 65 of file datamap.hpp.

10.56.3.64 void coco::datamap::remove (const std::string & __s, int i) [inline, inherited]

The remove methods remove the variable with name ___s and index i from the datamap.

Definition at line 70 of file datamap.hpp.

10.56.3.65 void coco::datamap::remove (const char * __s, int i) [inline, inherited]

The remove methods remove the variable with name ___s and index i from the datamap.

Definition at line 76 of file datamap.hpp.

10.56.3.66 bool coco::datamap::retrieve (const std::string & __s, bool & __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 149 of file datamap.hpp.

10.56.3.67 bool coco::datamap::retrieve (const std::string & __s, int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 169 of file datamap.hpp.

10.56.3.68 bool coco::datamap::retrieve (const std::string & __s, unsigned int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 191 of file datamap.hpp.

10.56.3.69 bool coco::datamap::retrieve (const std::string & __s, double & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 213 of file datamap.hpp.

10.56.3.70 bool coco::datamap::retrieve (const std::string & __s, interval & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 235 of file datamap.hpp.

10.56.3.71 bool coco::datamap::retrieve (const std::string & __s, std::string & __is) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 257 of file datamap.hpp.

10.56.3.72 bool coco::datamap::retrieve (const std::string & __s, num::Number & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 279 of file datamap.hpp.

10.56.3.73 bool coco::datamap::retrieve (const std::string & __s, const std::vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 301 of file datamap.hpp.

10.56.3.74 bool coco::datamap::retrieve (const std::string & __s, const std::vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 323 of file datamap.hpp.

10.56.3.75 bool coco::datamap::retrieve (const std::string & __s, const std::vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 345 of file datamap.hpp.

10.56.3.76 bool coco::datamap::retrieve (const std::string & __s, const std::vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 367 of file datamap.hpp.

10.56.3.77 bool coco::datamap::retrieve (const std::string & __s, const std::vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 389 of file datamap.hpp.

10.56.3.78 bool coco::datamap::retrieve (const std::string & __s, const std::vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 411 of file datamap.hpp.

10.56.3.79 bool coco::datamap::retrieve (const std::string & __s, const std::vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 433 of file datamap.hpp.

10.56.3.80 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 455 of file datamap.hpp.

10.56.3.81 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 477 of file datamap.hpp.

10.56.3.82 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 499 of file datamap.hpp.

10.56.3.83 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 521 of file datamap.hpp.

10.56.3.84 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 543 of file datamap.hpp.

10.56.3.85 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 565 of file datamap.hpp.

10.56.3.86 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 587 of file datamap.hpp.

10.56.3.87 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 609 of file datamap.hpp.

10.56.3.88 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 631 of file datamap.hpp.

10.56.3.89 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 653 of file datamap.hpp.

10.56.3.90 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 675 of file datamap.hpp.

10.56.3.91 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 697 of file datamap.hpp.

10.56.3.92 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 719 of file datamap.hpp.

10.56.3.93 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 741 of file datamap.hpp.

10.56.3.94 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 763 of file datamap.hpp.

10.56.3.95 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 785 of file datamap.hpp.

10.56.3.96 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 807 of file datamap.hpp.

10.56.3.97 bool coco::datamap::retrieve (const char * __s, bool & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 829 of file datamap.hpp.

10.56.3.98 bool coco::datamap::retrieve (const char * __s, int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 839 of file datamap.hpp.

10.56.3.99 bool coco::datamap::retrieve (const char * __s, unsigned int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 849 of file datamap.hpp.

10.56.3.100 bool coco::datamap::retrieve (const char * __s, double & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 859 of file datamap.hpp.

10.56.3.101 bool coco::datamap::retrieve (const char * __s, interval & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 869 of file datamap.hpp.

10.56.3.102 bool coco::datamap::retrieve (const char * __s, std::string & __is) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 879 of file datamap.hpp.

10.56.3.103 bool coco::datamap::retrieve (const char * __s, num::Number & __is) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 889 of file datamap.hpp.

10.56.3.104 bool coco::datamap::retrieve (const char * __s, const std::vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 899 of file datamap.hpp.

10.56.3.105 bool coco::datamap::retrieve (const char * __s, const std::vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 919 of file datamap.hpp.

10.56.3.106 bool coco::datamap::retrieve (const char * __s, const std::vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 909 of file datamap.hpp.

10.56.3.107 bool coco::datamap::retrieve (const char * __s, const std::vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 929 of file datamap.hpp.

10.56.3.108 bool coco::datamap::retrieve (const char * __s, const std::vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 939 of file datamap.hpp.

10.56.3.109 bool coco::datamap::retrieve (const char * __s, const std::vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 949 of file datamap.hpp.

10.56.3.110 bool coco::datamap::retrieve (const char * __s, const std::vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 959 of file datamap.hpp.

10.56.3.111 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 969 of file datamap.hpp.

10.56.3.112 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 989 of file datamap.hpp.

10.56.3.113 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 979 of file datamap.hpp.

10.56.3.114 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 999 of file datamap.hpp.

10.56.3.115 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1009 of file datamap.hpp.

10.56.3.116 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1019 of file datamap.hpp.

10.56.3.117 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1029 of file datamap.hpp.

10.56.3.118 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1039 of file datamap.hpp.

10.56.3.119 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1050 of file datamap.hpp.

10.56.3.120 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1061 of file datamap.hpp.

10.56.3.121 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1072 of file datamap.hpp.

10.56.3.122 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1083 of file datamap.hpp.

10.56.3.123 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1094 of file datamap.hpp.

10.56.3.124 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1105 of file datamap.hpp.

10.56.3.125 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1116 of file datamap.hpp.

10.56.3.126 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1127 of file datamap.hpp.

10.56.3.127 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1138 of file datamap.hpp.

10.56.3.128 bool coco::datamap::retrieve (const std::string & __s, bool & __v, bool __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 159 of file datamap.hpp.

10.56.3.129 bool coco::datamap::retrieve (const std::string & __s, int & __v, int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 181 of file datamap.hpp.

10.56.3.130 bool coco::datamap::retrieve (const std::string & __s, unsigned int & __v, unsigned int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 203 of file datamap.hpp.

10.56.3.131 bool coco::datamap::retrieve (const std::string & __s, double & __v, double __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 225 of file datamap.hpp.

10.56.3.132 bool coco::datamap::retrieve (const std::string & __s, interval & __v, const interval & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 247 of file datamap.hpp.

10.56.3.133 bool coco::datamap::retrieve (const std::string & __s, std::string & __is, const std::string & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 269 of file datamap.hpp.

10.56.3.134 bool coco::datamap::retrieve (const std::string & __s, num::Number & __is, const num::Number & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 291 of file datamap.hpp.

10.56.3.135 bool coco::datamap::retrieve (const std::string & __s, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 313 of file datamap.hpp.

10.56.3.136 bool coco::datamap::retrieve (const std::string & __s, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 335 of file datamap.hpp.

10.56.3.137 bool coco::datamap::retrieve (const std::string & __s, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 357 of file datamap.hpp.

10.56.3.138 bool coco::datamap::retrieve (const std::string & __s, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 379 of file datamap.hpp.

10.56.3.139 bool coco::datamap::retrieve (const std::string & __s, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 401 of file datamap.hpp.

10.56.3.140 bool coco::datamap::retrieve (const std::string & __s, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 423 of file datamap.hpp.

10.56.3.141 bool coco::datamap::retrieve (const std::string & __s, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 445 of file datamap.hpp.

10.56.3.142 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 467 of file datamap.hpp.

10.56.3.143 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 489 of file datamap.hpp.

10.56.3.144 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 511 of file datamap.hpp.

10.56.3.145 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 533 of file datamap.hpp.

10.56.3.146 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 555 of file datamap.hpp.

10.56.3.147 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 577 of file datamap.hpp.

10.56.3.148 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 599 of file datamap.hpp.

10.56.3.149 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 621 of file datamap.hpp.

10.56.3.150 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 643 of file datamap.hpp.

10.56.3.151 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 665 of file datamap.hpp.

10.56.3.152 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 687 of file datamap.hpp.

10.56.3.153 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 709 of file datamap.hpp.

10.56.3.154 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 731 of file datamap.hpp.

10.56.3.155 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 753 of file datamap.hpp.

10.56.3.156 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 775 of file datamap.hpp.

10.56.3.157 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 797 of file datamap.hpp.

10.56.3.158 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 819 of file datamap.hpp.

10.56.3.159 bool coco::datamap::retrieve (const char * __s, bool & __v, bool __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 834 of file datamap.hpp.

10.56.3.160 bool coco::datamap::retrieve (const char * __s, int & __v, int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 844 of file datamap.hpp.

10.56.3.161 bool coco::datamap::retrieve (const char * __s, unsigned int & __v, unsigned int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 854 of file datamap.hpp.

10.56.3.162 bool coco::datamap::retrieve (const char * __s, double & __v, double __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 864 of file datamap.hpp.

10.56.3.163 bool coco::datamap::retrieve (const char * __s, interval & __v, const interval & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 874 of file datamap.hpp.

10.56.3.164 bool coco::datamap::retrieve (const char * __s, std::string & __v, const std::string & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 884 of file datamap.hpp.

10.56.3.165 bool coco::datamap::retrieve (const char * __s, num::Number & __v, const num::Number & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 894 of file datamap.hpp.

10.56.3.166 bool coco::datamap::retrieve (const char * __s, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 904 of file datamap.hpp.

10.56.3.167 bool coco::datamap::retrieve (const char * __s, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 924 of file datamap.hpp.

10.56.3.168 bool coco::datamap::retrieve (const char * __s, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 914 of file datamap.hpp.

10.56.3.169 bool coco::datamap::retrieve (const char * __s, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 934 of file datamap.hpp.

10.56.3.170 bool coco::datamap::retrieve (const char * __s, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 944 of file datamap.hpp.

10.56.3.171 bool coco::datamap::retrieve (const char * __s, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 954 of file datamap.hpp.

10.56.3.172 bool coco::datamap::retrieve (const char * __s, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 964 of file datamap.hpp.

10.56.3.173 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 974 of file datamap.hpp.

10.56.3.174 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 994 of file datamap.hpp.

10.56.3.175 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 984 of file datamap.hpp.

10.56.3.176 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1004 of file datamap.hpp.

10.56.3.177 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1014 of file datamap.hpp.

10.56.3.178 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1024 of file datamap.hpp.

10.56.3.179 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1034 of file datamap.hpp.

10.56.3.180 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1044 of file datamap.hpp.

10.56.3.181 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1055 of file datamap.hpp.

10.56.3.182 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1066 of file datamap.hpp.

10.56.3.183 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1077 of file datamap.hpp.

10.56.3.184 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1088 of file datamap.hpp.

10.56.3.185 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1099 of file datamap.hpp.

10.56.3.186 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1110 of file datamap.hpp.

10.56.3.187 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1121 of file datamap.hpp.

10.56.3.188 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1132 of file datamap.hpp.

10.56.3.189 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1143 of file datamap.hpp.

10.56.3.190 bool coco::datamap::retrieve_i (const std::string & __s, int i, bool & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1150 of file datamap.hpp.

10.56.3.191 bool coco::datamap::retrieve_i (const std::string & __s, int i, int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1170 of file datamap.hpp.

10.56.3.192 bool coco::datamap::retrieve_i (const std::string & __s, int i, unsigned int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1190 of file datamap.hpp.

10.56.3.193 bool coco::datamap::retrieve_i (const std::string & __s, int i, double & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1210 of file datamap.hpp.

10.56.3.194 bool coco::datamap::retrieve_i (const std::string & __s, int i, interval & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1230 of file datamap.hpp.

10.56.3.195 bool coco::datamap::retrieve_i (const std::string & __s, int i, std::string & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1250 of file datamap.hpp.

10.56.3.196 bool coco::datamap::retrieve_i (const std::string & __s, int i, num::Number & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1270 of file datamap.hpp.

10.56.3.197 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1290 of file datamap.hpp.

10.56.3.198 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1310 of file datamap.hpp.

10.56.3.199 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1330 of file datamap.hpp.

10.56.3.200 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1350 of file datamap.hpp.

10.56.3.201 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1370 of file datamap.hpp.

10.56.3.202 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1390 of file datamap.hpp.

10.56.3.203 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1410 of file datamap.hpp.

10.56.3.204 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1430 of file datamap.hpp.

10.56.3.205 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1450 of file datamap.hpp.

10.56.3.206 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1470 of file datamap.hpp.

10.56.3.207 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1490 of file datamap.hpp.

10.56.3.208 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1510 of file datamap.hpp.

10.56.3.209 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1530 of file datamap.hpp.

10.56.3.210 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1550 of file datamap.hpp.

10.56.3.211 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1570 of file datamap.hpp.

10.56.3.212 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1590 of file datamap.hpp.

10.56.3.213 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1610 of file datamap.hpp.

10.56.3.214 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1630 of file datamap.hpp.

10.56.3.215 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1650 of file datamap.hpp.

10.56.3.216 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1670 of file datamap.hpp.

10.56.3.217 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1690 of file datamap.hpp.

10.56.3.218 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1710 of file datamap.hpp.

10.56.3.219 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1730 of file datamap.hpp.

10.56.3.220 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1750 of file datamap.hpp.

10.56.3.221 bool coco::datamap::retrieve_i (const char * __s, int i, bool & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1770 of file datamap.hpp.

10.56.3.222 bool coco::datamap::retrieve_i (const char * __s, int i, int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1780 of file datamap.hpp.

10.56.3.223 bool coco::datamap::retrieve_i (const char * __s, int i, unsigned int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1790 of file datamap.hpp.

10.56.3.224 bool coco::datamap::retrieve_i (const char * __s, int i, double & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1800 of file datamap.hpp.

10.56.3.225 bool coco::datamap::retrieve_i (const char * __s, int i, interval & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1810 of file datamap.hpp.

10.56.3.226 bool coco::datamap::retrieve_i (const char * __s, int i, std::string & __is) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1820 of file datamap.hpp.

10.56.3.227 bool coco::datamap::retrieve_i (const char * __s, int i, num::Number & __is) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1830 of file datamap.hpp.

10.56.3.228 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1840 of file datamap.hpp.

10.56.3.229 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1850 of file datamap.hpp.

10.56.3.230 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1860 of file datamap.hpp.

10.56.3.231 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1870 of file datamap.hpp.

10.56.3.232 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1880 of file datamap.hpp.

10.56.3.233 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1890 of file datamap.hpp.

10.56.3.234 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1900 of file datamap.hpp.

10.56.3.235 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1910 of file datamap.hpp.

10.56.3.236 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1920 of file datamap.hpp.

10.56.3.237 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1930 of file datamap.hpp.

10.56.3.238 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1940 of file datamap.hpp.

10.56.3.239 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1950 of file datamap.hpp.

10.56.3.240 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1960 of file datamap.hpp.

10.56.3.241 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1970 of file datamap.hpp.

10.56.3.242 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1980 of file datamap.hpp.

10.56.3.243 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1991 of file datamap.hpp.

10.56.3.244 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2002 of file datamap.hpp.

10.56.3.245 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2013 of file datamap.hpp.

10.56.3.246 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2024 of file datamap.hpp.

10.56.3.247 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2035 of file datamap.hpp.

10.56.3.248 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2046 of file datamap.hpp.

10.56.3.249 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2057 of file datamap.hpp.

10.56.3.250 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2068 of file datamap.hpp.

10.56.3.251 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2079 of file datamap.hpp.

10.56.3.252 bool coco::datamap::retrieve_i (const std::string & __s, int i, bool & __v, bool __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1161 of file datamap.hpp.

10.56.3.253 bool coco::datamap::retrieve_i (const std::string & __s, int i, int & __v, int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1181 of file datamap.hpp.

10.56.3.254 bool coco::datamap::retrieve_i (const std::string & __s, int i, unsigned int & __v, unsigned int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1201 of file datamap.hpp.

10.56.3.255 bool coco::datamap::retrieve_i (const std::string & __s, int i, double & __v, double __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1221 of file datamap.hpp.

10.56.3.256 bool coco::datamap::retrieve_i (const std::string & __s, int i, interval & __v, const interval & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1241 of file datamap.hpp.

10.56.3.257 bool coco::datamap::retrieve_i (const std::string & __s, int i, std::string & __is, const std::string & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1261 of file datamap.hpp.

10.56.3.258 bool coco::datamap::retrieve_i (const std::string & __s, int i, num::Number & __is, const num::Number & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1281 of file datamap.hpp.

10.56.3.259 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1301 of file datamap.hpp.

10.56.3.260 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1321 of file datamap.hpp.

10.56.3.261 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1341 of file datamap.hpp.

10.56.3.262 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1361 of file datamap.hpp.

10.56.3.263 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1381 of file datamap.hpp.

10.56.3.264 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1401 of file datamap.hpp.

10.56.3.265 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1421 of file datamap.hpp.

10.56.3.266 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1441 of file datamap.hpp.

10.56.3.267 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1461 of file datamap.hpp.

10.56.3.268 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1481 of file datamap.hpp.

10.56.3.269 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1501 of file datamap.hpp.

10.56.3.270 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1521 of file datamap.hpp.

10.56.3.271 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1541 of file datamap.hpp.

10.56.3.272 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1561 of file datamap.hpp.

10.56.3.273 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1581 of file datamap.hpp.

10.56.3.274 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1601 of file datamap.hpp.

10.56.3.275 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1621 of file datamap.hpp.

10.56.3.276 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1641 of file datamap.hpp.

10.56.3.277 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1661 of file datamap.hpp.

10.56.3.278 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1681 of file datamap.hpp.

10.56.3.279 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1701 of file datamap.hpp.

10.56.3.280 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1721 of file datamap.hpp.

10.56.3.281 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1741 of file datamap.hpp.

10.56.3.282 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1761 of file datamap.hpp.

10.56.3.283 bool coco::datamap::retrieve_i (const char * __s, int i, bool & __v, bool __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1775 of file datamap.hpp.

10.56.3.284 bool coco::datamap::retrieve_i (const char * __s, int i, int & __v, int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1785 of file datamap.hpp.

10.56.3.285 bool coco::datamap::retrieve_i (const char * __s, int i, unsigned int & __v, unsigned int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1795 of file datamap.hpp.

10.56.3.286 bool coco::datamap::retrieve_i (const char * __s, int i, double & __v, double __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1805 of file datamap.hpp.

10.56.3.287 bool coco::datamap::retrieve_i (const char * __s, int i, interval & __v, const interval & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1815 of file datamap.hpp.

10.56.3.288 bool coco::datamap::retrieve_i (const char * __s, int i, std::string & __v, const std::string & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1825 of file datamap.hpp.

10.56.3.289 bool coco::datamap::retrieve_i (const char * __s, int i, num::Number & __v, const num::Number & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1835 of file datamap.hpp.

10.56.3.290 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1845 of file datamap.hpp.

10.56.3.291 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1855 of file datamap.hpp.

10.56.3.292 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1865 of file datamap.hpp.

10.56.3.293 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1875 of file datamap.hpp.

10.56.3.294 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1885 of file datamap.hpp.

10.56.3.295 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1895 of file datamap.hpp.

10.56.3.296 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1905 of file datamap.hpp.

10.56.3.297 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1915 of file datamap.hpp.

10.56.3.298 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1925 of file datamap.hpp.

10.56.3.299 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1935 of file datamap.hpp.

10.56.3.300 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1945 of file datamap.hpp.

10.56.3.301 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1955 of file datamap.hpp.

10.56.3.302 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1965 of file datamap.hpp.

10.56.3.303 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1975 of file datamap.hpp.

10.56.3.304 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1985 of file datamap.hpp.

10.56.3.305 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1996 of file datamap.hpp.

10.56.3.306 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2007 of file datamap.hpp.

10.56.3.307 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2018 of file datamap.hpp.

10.56.3.308 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2029 of file datamap.hpp.

10.56.3.309 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2040 of file datamap.hpp.

10.56.3.310 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2051 of file datamap.hpp.

10.56.3.311 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2062 of file datamap.hpp.

10.56.3.312 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2073 of file datamap.hpp.

10.56.3.313 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2084 of file datamap.hpp.

10.56.3.314 void coco::control_data::service (const std::string & __s) [inline]

This method sets the service to __s.

Definition at line 35 of file control_data.hpp.

10.56.3.315 void coco::control_data::service (const char * __s) [inline]

This method sets the service to __s.

Definition at line 37 of file control_data.hpp.

10.56.3.316 const std::string & coco::control_data::service ( ) const throw (api_exception) [inline]

This method returns the service. If no service is defined, an api_exception of type apiee_communication_-data is thrown.

Definition at line 42 of file control_data.hpp.

10.56.3.317 void coco::control_data::set (const info_contents & __i) [inline]

This method sets all parameters which are set in the info_contents structure __i to the values specified there.

Definition at line 104 of file control_data.h.

10.56.3.318 void coco::control_data::set (const std::string & __s, const basic_alltype & __h) [inline]

These methods set the parameter with name __s to value __h. Note that the constructors of the basic_-alltype class make it possible to use the set methods for every "subtype" of the basic_alltype class without explicitly using a basic_alltype constructor

Definition at line 112 of file control_data.h.

10.56.3.319 void coco::control_data::set (const char * __cp, const basic_alltype & __h) [inline]

These methods set the parameter with name ___s to value ___h. Note that the constructors of the basic_alltype class make it possible to use the set methods for every "subtype" of the basic_alltype class without explicitly using a basic_alltype constructor

Definition at line 114 of file control_data.h.

10.56.3.320 void coco::control_data::set (const std::string & __s, int i, const basic_alltype & __h) [inline]

These methods set the parameter with name ___s and index i to value ___h. Note that the constructors of the basic_alltype class * make it possible to use the set methods for every "subtype" of the basic_alltype class without explicitly using a basic_alltype constructor

Definition at line 123 of file control_data.h.

10.56.3.321 void coco::control_data::set (const char * __cp, int i, const basic_alltype & __h) [inline]

These methods set the parameter with name ___s and index i to value ___h. Note that the constructors of the basic_alltype class * make it possible to use the set methods for every "subtype" of the basic_alltype class without explicitly using a basic_alltype constructor

Definition at line 125 of file control_data.h.

10.56.3.322 const basic_alltype & coco::datamap::sfind (const std::string & __s) const [inline, inherited]

This method returns the value (as basic_alltype) of the variable ___s. If the variable is not set, an empty basic_alltype is returned.

Definition at line 81 of file datamap.hpp.

10.56.3.323 const basic_alltype & coco::datamap::sfind (const char * __s) const [inline, inherited]

This method returns the value (as basic_alltype) of the variable ___s. If the variable is not set, an empty basic_alltype is returned.

Definition at line 90 of file datamap.hpp.

10.56.3.324 const basic_alltype & coco::datamap::sfind (const std::string & __s, int i) const [inline, inherited]

This method returns the value (as basic_alltype) of the variable ___s with index i. If the variable is not set, an empty basic_alltype is returned.

Definition at line 95 of file datamap.hpp.

10.56.3.325 const basic_alltype & coco::datamap::sfind (const char * __s, int i) const [inline, inherited]

This method returns the value (as basic_alltype) of the variable ___s with index i. If the variable is not set, an empty basic_alltype is returned.

Definition at line 104 of file datamap.hpp.

10.56.3.326 bool coco::datamap::sinsert (const std::string & __s, const basic_alltype & __h, bool replace) [inherited]

This method inserts the variable __s with value __h into the datamap. If replace is true, the value of an already defined variable __s would be replaced, if it is false the new value would not be set. The method returns whether the variable __s was indeed set to __h.

Definition at line 36 of file datamap.cc.

10.56.3.327 bool coco::datamap::sinsert (const char * __s, const basic_alltype & __h, bool replace) [inline, inherited]

This method inserts the variable __s with value __h into the datamap. If replace is true, the value of an already defined variable __s would be replaced, if it is false the new value would not be set. The method returns whether the variable __s was indeed set to __h.

Definition at line 36 of file datamap.hpp.

10.56.3.328 bool coco::datamap::sinsert (const std::string & __s, int i, const basic_alltype & __h, bool replace) [inline, inherited]

This method inserts the variable __s with index i with value __h into the datamap. If replace is true, the value of an already defined variable __s with index i would be replaced, if it is false the new value would not be set. The method returns whether the variable __s with index i was indeed set to __h.

Definition at line 42 of file datamap.hpp.

10.56.3.329 bool coco::datamap::sinsert (const char * __s, int i, const basic_alltype & __h, bool replace) [inline, inherited]

This method inserts the variable __s with index i with value __h into the datamap. If replace is true, the value of an already defined variable __s with index i would be replaced, if it is false the new value would not be set. The method returns whether the variable __s with index i was indeed set to __h.

Definition at line 50 of file datamap.hpp.

10.56.3.330 void coco::control_data::unset (const std::string & __s) [inline]

These unset methods remove the parameter with name __s from the control data.

Definition at line 150 of file control_data.h.

10.56.3.331 void coco::control_data::unset (const char * __cp) [inline]

These unset methods remove the parameter with name __s from the control data.

Definition at line 151 of file control_data.h.

10.56.3.332 void coco::control_data::unset (const std::string & __s, int i) [inline]

These unset methods remove the parameter with name __s and index i from the control data.

Definition at line 156 of file control_data.h.

10.56.3.333 void coco::control_data::unset (const char * __cp, int i) [inline]

These unset methods remove the parameter with name __s and index i from the control data.

Definition at line 157 of file control_data.h.

10.56.3.334 bool coco::datamap::which (const std::string & __s, std::vector< int > & __idx) const [inline, inherited]

The which methods sets __idx to the list of indices defined for the variable with name __s. The methods return whether the variable __s is defined.

Definition at line 129 of file datamap.hpp.

10.56.3.335 bool coco::datamap::which (const char * __s, std::vector< int > & __idx) const [inline, inherited]

The which methods sets __idx to the list of indices defined for the variable with name __s. The methods return whether the variable __s is defined.

Definition at line 143 of file datamap.hpp.

10.56.3.336 bool coco::control_data::which_set (const std::string & __s, std::vector< int > & __idx) const [inline]

These methods set the vector __idx to all indices which are defined for the parameter with name __s.

Definition at line 183 of file control_data.h.

10.56.3.337 bool coco::control_data::which_set (const char * __cp, std::vector< int > & __idx) const [inline]

These methods set the vector __idx to all indices which are defined for the parameter with name __s.

Definition at line 185 of file control_data.h.

The documentation for this class was generated from the following files:

- control_data.h
- control_data.hpp

10.57 coco::convex_e Class Reference

Convexity information.

`#include <convex_info.h>`

Public Member Functions

- convex_e ()
- convex_e (const convex_info &__e, uint16_t __t)
- convex_e (const convex_e &__e)

- convex_e (int16_t __e)
- ~convex_e ()
- const convex_info & i () const
- uint16_t t () const
- convex_e operator- () const
- convex_e & operator= (const convex_e &__c)
- convex_e & operator= (const convex_info &__i)
- convex_e & operator= (unsigned int __t)
- convex_e & operator= (uint16_t __t)
- void read (char *c)
- void merge (const convex_e &__c)

Friends

- bool operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.
- bool operator== (const convex_e &__c, const convex_info &__d)
- bool operator== (const convex_info &__c, const convex_e &__d)
- bool operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.
- bool operator!= (const convex_e &__c, const convex_info &__d)
- bool operator!= (const convex_info &__c, const convex_e &__d)
- std::ostream & operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.

10.57.1 Detailed Description

This class stores the convexity information of a node. It is part of the semantics structure.

10.57.2 Constructor & Destructor Documentation

10.57.2.1 coco::convex_e::convex_e ( ) [inline]

Standard Constructor

Definition at line 70 of file convex_info.h.

10.57.2.2 coco::convex_e::convex_e (const convex_info & __e, uint16_t __t) [inline]

Constructor, which initializes the convex_info with __e and the algorithm complexity with __t.

Definition at line 73 of file convex_info.h.

10.57.2.3 coco::convex_e::convex_e (const convex_e & __e) [inline]

Standard Copy Constructor

Definition at line 75 of file convex_info.h.

10.57.2.4 coco::convex_e::convex_e (int16_t __e) [inline, explicit]

Constructor, which initializes the convex_info with the equivalent integer ___e and sets the algorithm complexity to 0.

Definition at line 78 of file convex_info.h.

10.57.2.5 coco::convex_e::~convex_e ( ) [inline]

Standard Destructor

Definition at line 81 of file convex_info.h.

10.57.3 Member Function Documentation

10.57.3.1 const convex_info& coco::convex_e::i ( ) const [inline]

This method returns the convex_info stored.

Definition at line 84 of file convex_info.h.

10.57.3.2 void coco::convex_e::merge (const convex_e & __c) [inline]

This method merges a convex_e with another convex_e ___c.

Definition at line 136 of file convex_info.h.

10.57.3.3 convex_e coco::convex_e::operator- ( ) const [inline]

The unary minus operator changes a convex_e in the same way as the convexity structure of f changes for the transition $f \to -f$.

Definition at line 91 of file convex_info.h.

10.57.3.4 convex_e& coco::convex_e::operator= (const convex_e & __c) [inline]

Standard Assignment Operator

Definition at line 104 of file convex_info.h.

10.57.3.5 convex_e& coco::convex_e::operator= (const convex_info & __i) [inline]

Assignment Operator, which only assigns the convex_info part.

Definition at line 112 of file convex_info.h.

10.57.3.6 convex_e& coco::convex_e::operator= (unsigned int __t) [inline]

Assignment Operator, which only assigns the algorithm complexity part.

Definition at line 119 of file convex_info.h.

10.57.3.7 convex_e& coco::convex_e::operator= (uint16_t __t) [inline]

Assignment Operator, which only assigns the algorithm complexity part.

Definition at line 126 of file convex_info.h.

10.57.3.8 void coco::convex_e::read (char * c)

This method reads a convex_e from the string c in .dag encoding.

10.57.3.9 uint16_t coco::convex_e::t ( ) const [inline]

This method returns the algorithm complexity stored.

Definition at line 86 of file convex_info.h.

10.57.4 Friends And Related Function Documentation

10.57.4.1 bool operator!= (const convex_e & __c, const convex_e & __d) [friend]

Disequality comparison operator. Note that only the convex_info part counts!

Definition at line 195 of file convex_info.h.

10.57.4.2 bool operator!= (const convex_e & __c, const convex_info & __d) [friend]

Disequality comparison operator with a convex_info.

Definition at line 203 of file convex_info.h.

10.57.4.3 bool operator!= (const convex_info & __c, const convex_e & __d) [friend]

Disequality comparison operator with a convex_info.

Definition at line 211 of file convex_info.h.

10.57.4.4 std::ostream& operator<< (std::ostream & o, const convex_e & __s) [friend]

This operator writes a convex_e in .dag format to an ostream.

Definition at line 219 of file convex_info.h.

10.57.4.5 bool operator== (const convex_e & __c, const convex_e & __d) [friend]

Equality comparison operator. Note that only the convex_info part counts!

Definition at line 171 of file convex_info.h.

10.57.4.6 bool operator== (const convex_e & __c, const convex_info & __d) [friend]

Equality comparison operator with a convex_info.

Definition at line 179 of file convex_info.h.

10.57.4.7 bool operator== (const convex_info & __c, const convex_e & __d) [friend]

Equality comparison operator with a convex_info.

Definition at line 187 of file convex_info.h.

The documentation for this class was generated from the following file:

- convex_info.h

10.58 coco::coco::convex_e Class Reference

Convexity information.

Public Member Functions

- convex_e ()
- convex_e (const convex_info &__e, uint16_t __t)
- convex_e (const convex_e &__e)
- convex_e (int16_t __e)
- ~convex_e ()
- const convex_info & i () const
- uint16_t t () const
- convex_e operator- () const
- convex_e & operator= (const convex_e &__c)
- convex_e & operator= (const convex_info &__i)
- convex_e & operator= (unsigned int __t)
- convex_e & operator= (uint16_t __t)
- void read (char *c)
- void merge (const convex_e &__c)

Friends

- bool operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.
- bool operator== (const convex_e &__c, const convex_info &__d)
- bool operator== (const convex_info &__c, const convex_e &__d)
- bool operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.
- bool operator!= (const convex_e &__c, const convex_info &__d)
- bool operator!= (const convex_info &__c, const convex_e &__d)
- std::ostream & operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.

10.58.1 Detailed Description

This class stores the convexity information of a node. It is part of the semantics structure.

10.58.2 Constructor & Destructor Documentation

10.58.2.1 coco::coco::convex_e::convex_e ( ) [inline]

Standard Constructor

Definition at line 70 of file search_graph.cc.

10.58.2.2 coco::coco::convex_e::convex_e (const convex_info & __e, uint16_t __t) [inline]

Constructor, which initializes the convex_info with ___e and the algorithm complexity with ___t.

Definition at line 73 of file search_graph.cc.

10.58.2.3 coco::coco::convex_e::convex_e (const convex_e & __e) [inline]

Standard Copy Constructor

Definition at line 75 of file search_graph.cc.

10.58.2.4 coco::coco::convex_e::convex_e (int16_t __e) [inline, explicit]

Constructor, which initializes the convex_info with the equivalent integer ___e and sets the algorithm complexity to 0.

Definition at line 78 of file search_graph.cc.

10.58.2.5 coco::coco::convex_e::~convex_e ( ) [inline]

Standard Destructor

Definition at line 81 of file search_graph.cc.

10.58.3 Member Function Documentation

10.58.3.1 const convex_info& coco::coco::convex_e::i ( ) const [inline]

This method returns the convex_info stored.

Definition at line 84 of file search_graph.cc.

10.58.3.2 void coco::coco::convex_e::merge (const convex_e & __c) [inline]

This method merges a convex_e with another convex_e ___c.

Definition at line 136 of file search_graph.cc.

10.58.3.3 convex_e coco::coco::convex_e::operator- ( ) const [inline]

The unary minus operator changes a convex_e in the same way as the convexity structure of f changes for the transition $f \to -f$.

Definition at line 91 of file search_graph.cc.

10.58.3.4 convex_e& coco::coco::convex_e::operator= (const convex_e & __c) [inline]

Standard Assignment Operator

Definition at line 104 of file search_graph.cc.

10.58.3.5 convex_e& coco::coco::convex_e::operator= (const convex_info & __i) [inline]

Assignment Operator, which only assigns the convex_info part.

Definition at line 112 of file search_graph.cc.

10.58.3.6 convex_e& coco::coco::convex_e::operator= (unsigned int __t) [inline]

Assignment Operator, which only assigns the algorithm complexity part.

Definition at line 119 of file search_graph.cc.

10.58.3.7 convex_e& coco::coco::convex_e::operator= (uint16_t __t) [inline]

Assignment Operator, which only assigns the algorithm complexity part.

Definition at line 126 of file search_graph.cc.

10.58.3.8 void coco::convex_e::read (char * c)

This method reads a convex_e from the string c in .dag encoding.

Definition at line 39 of file convex_info.cc.

10.58.3.9 uint16_t coco::coco::convex_e::t ( ) const [inline]

This method returns the algorithm complexity stored.

Definition at line 86 of file search_graph.cc.

10.58.4 Friends And Related Function Documentation

10.58.4.1 bool operator!= (const convex_e & __c, const convex_e & __d) [friend]

Disequality comparison operator. Note that only the convex_info part counts!

Definition at line 195 of file search_graph.cc.

10.58.4.2 bool operator!= (const convex_e & __c, const convex_info & __d) [friend]

Disequality comparison operator with a convex_info.

Definition at line 203 of file search_graph.cc.

10.58.4.3 bool operator!= (const convex_info & __c, const convex_e & __d) [friend]

Disequality comparison operator with a convex_info.

Definition at line 211 of file search_graph.cc.

10.58.4.4 std::ostream& operator<< (std::ostream & *o*, const convex_e & *__s*) [friend]

This operator writes a convex_e in .dag format to an `ostream`.

Definition at line 219 of file search_graph.cc.

10.58.4.5 bool operator== (const convex_e & *__c*, const convex_e & *__d*) [friend]

Equality comparison operator. Note that only the convex_info part counts!

Definition at line 171 of file search_graph.cc.

10.58.4.6 bool operator== (const convex_e & *__c*, const convex_info & *__d*) [friend]

Equality comparison operator with a convex_info.

Definition at line 179 of file search_graph.cc.

10.58.4.7 bool operator== (const convex_info & *__c*, const convex_e & *__d*) [friend]

Equality comparison operator with a convex_info.

Definition at line 187 of file search_graph.cc.

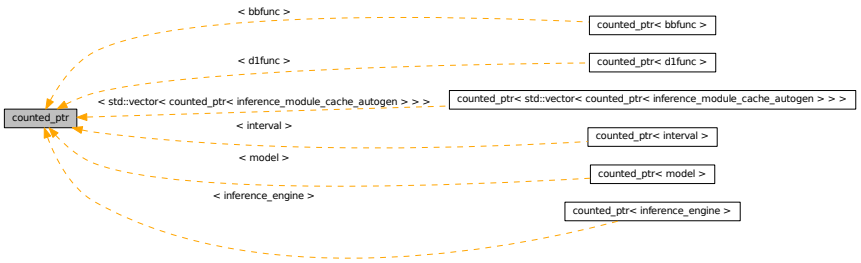
The documentation for this class was generated from the following files:

- convex_info.h
- convex_info.cc

10.59 counted_ptr Class Reference

`#include <counted_ptr.h>`

Inheritance diagram for counted_ptr:



Classes

- class counter

Public Types

- typedef _Tp element_type

Public Member Functions

- counted_ptr (_Tp *p=0)
- ~counted_ptr ()
- counted_ptr (const counted_ptr &r) throw ()
- counted_ptr & operator= (const counted_ptr &r)
- counted_ptr & operator= (element_type *r)
- template<typename _Tq >
  counted_ptr (const counted_ptr< _Tq > &r) throw ()
- template<typename _Tq >
  counted_ptr & operator= (const counted_ptr< _Tq > &r)
- bool operator== (const counted_ptr< _Tp > &a) const
- bool operator!= (const counted_ptr< _Tp > &a) const
- _Tp & operator* () const throw ()
- _Tp * operator-> () const throw ()
- _Tp * get () const throw ()
- bool is_unique () const throw ()
- bool is_empty () const

10.59.1 Detailed Description

This class provides a simple reference counted pointer. The implementation is non-intrusive; it allocates an additional int and pointer for every counted object.

10.59.2 Member Typedef Documentation

10.59.2.1 typedef _Tp counted_ptr::element_type

The element type of the counted pointer

Definition at line 63 of file counted_ptr.h.

10.59.3 Constructor & Destructor Documentation

10.59.3.1 counted_ptr::counted_ptr (_Tp * *p* = `0`) [inline, explicit]

The constructor allocates a new counter object, at least if `is` non-NULL.

Definition at line 68 of file counted_ptr.h.

10.59.3.2 counted_ptr::~counted_ptr ( ) [inline]

The Standard Destructor just deletes this reference to the counted object.

Definition at line 72 of file counted_ptr.h.

10.59.3.3 counted_ptr::counted_ptr (const counted_ptr & *r*) throw () [inline]

The Copy Constructor gets a new reference to the counted object.

Definition at line 75 of file counted_ptr.h.

10.59.3.4 template<typename _Tq > counted_ptr::counted_ptr (const counted_ptr< _Tq > & *r*) throw () [inline]

The constructor allocates a new counter object.

Definition at line 105 of file counted_ptr.h.

10.59.4 Member Function Documentation

10.59.4.1 _Tp* counted_ptr::get ( ) const throw () [inline]

This is a safeguarded pointer operator

Definition at line 155 of file counted_ptr.h.

10.59.4.2 bool counted_ptr::is_empty ( ) const [inline]

Definition at line 160 of file counted_ptr.h.

10.59.4.3 bool counted_ptr::is_unique ( ) const throw () [inline]

This checks whether the object has only this (unique) reference.

Definition at line 158 of file counted_ptr.h.

10.59.4.4 bool counted_ptr::operator!= (const counted_ptr< _Tp > & *a*) const [inline]

The comparison operator checks whether the pointer to the objects coincide.

Definition at line 134 of file counted_ptr.h.

10.59.4.5 _Tp& counted_ptr::operator* ( ) const throw () [inline]

This is the dereferentiation operator

Definition at line 149 of file counted_ptr.h.

10.59.4.6 _Tp* counted_ptr::operator-> ( ) const throw () [inline]

This is the pointer operator

Definition at line 152 of file counted_ptr.h.

10.59.4.7 counted_ptr& counted_ptr::operator= (const counted_ptr & *r*) [inline]

The Assignment Operator first deletes the reference for the object this counted_ptr points at and then gets a new reference for the assigned object.

Definition at line 81 of file counted_ptr.h.

10.59.4.8 counted_ptr& counted_ptr::operator= (element_type * r) [inline]

This Assignment Operator explicitly sets a new element_type pointer.

Definition at line 92 of file counted_ptr.h.

10.59.4.9 template<typename _Tq > counted_ptr& counted_ptr::operator= (const counted_ptr< _Tq > & r) [inline]

The Assignment Operator first deletes the reference for the object this counted_ptr points at and then gets a new reference for the assigned object.

Definition at line 112 of file counted_ptr.h.

10.59.4.10 bool counted_ptr::operator== (const counted_ptr< _Tp > & a) const [inline]

The comparison operator checks whether the pointer to the objects coincide.

Definition at line 124 of file counted_ptr.h.

The documentation for this class was generated from the following file:

- counted_ptr.h

10.60 counted_ptr::counter Class Reference

Public Member Functions

- counter (_Tp *ptr=0, unsigned int cntr=1)

Public Attributes

- _Tp * p
- unsigned int c

10.60.1 Detailed Description

This is the counter class allocated for keeping track of the reference count

10.60.2 Constructor & Destructor Documentation

10.60.2.1 counted_ptr::counter::counter (_Tp * ptr = 0, unsigned int cntr = 1) [inline]

Standard constructor, assigning to and to .

Definition at line 54 of file counted_ptr.h.

10.60.3 Member Data Documentation

10.60.3.1 unsigned int counted_ptr::counter::c

This is the counter

Definition at line 50 of file counted_ptr.h.

10.60.3.2 _Tp* counted_ptr::counter::p

This is the pointer to the reference counted data

Definition at line 48 of file counted_ptr.h.

The documentation for this class was generated from the following file:

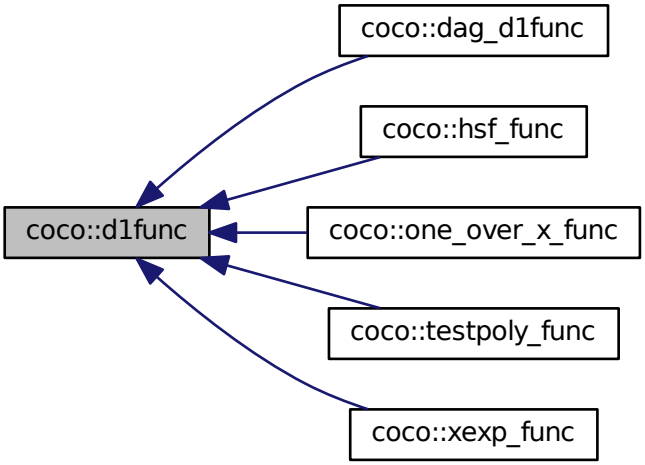
- counted_ptr.h

10.61 coco::d1func Class Reference

The d1func class (one dimensional function)

`#include <d1func.h>`

Inheritance diagram for coco::d1func:



Classes

- struct func_info

the func_info class collects all information needed for optimal evaluation

Public Types

- enum d1func_point_t { d1fp_unknown = -2, d1fp_regular = -1, d1fp_extremum = 0, d1fp_evenpoleup = 1, d1fp_evenpoledn = 2, d1fp_oddpoleup = 3, d1fp_oddpoledn = 4, d1fp_jumpldef = 5, d1fp_jumpludef = 6, d1fp_point = 7, d1fp_undefined = 8, d1fp_wild = 9, d1fp_jumprdef = 10, d1fp_jumprudef = 11 }
- enum d1solve_t { d1solve_0, d1solve_1, d1solve_2 }
- typedef std::triple< interval, interval, d1func_point_t > point_info
- typedef std::vector< point_info > point_list

Public Member Functions

- d1func ()
- d1func (int _d)
- d1func (double l, double r)
- d1func (double l, double r, int _d)
- d1func (const d1func &wh)
- virtual ~d1func ()
- virtual interval imperfect_eval (const interval &x, unsigned int k) const =0
- virtual double eval (double x, unsigned int k) const =0
- virtual const char * func_name () const =0
- virtual uint64_t thash ()=0
- virtual uint64_t chash ()=0
- virtual std::string dag_out () const
- virtual std::ostream & print_add (std::ostream &o) const
- virtual std::string print_params () const
- virtual diffNumber eval (const diffNumber &x, unsigned int k) const
- virtual hessNumber eval (const hessNumber &x, unsigned int k) const
- virtual interval eval (const interval &x, unsigned int k) const
- virtual diffI eval (const diffI &x, unsigned int k) const
- virtual ihessNumber eval (const ihessNumber &x, unsigned int k) const
- virtual Islope eval (const Islope &x, unsigned int k) const
- virtual interval_set eval (const interval_set &x, unsigned int k) const
- virtual interval eval_slp (const interval &z, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &y, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &x, unsigned int k) const
- virtual void internal_eval_d1fp (const diffI &x, diffI &res, const std::vector< d1func_point_t > &ptin, std::vector< d1func_point_t > &ptres, unsigned int order) const
- virtual interval_set intersect_inv (const interval &x, const interval &r, unsigned int k) const
- virtual interval_set intersect_inv (const interval_set &x, const interval_set &r, unsigned int k) const
- virtual std::pair< double, double > evald (double x, unsigned int k) const
- virtual std::triple< double, double, double > evalt (double x, unsigned int k) const
- virtual void evalf (double x, double *r, unsigned int k) const
- virtual convex_info convexity (const interval &i, unsigned int k) const
- virtual d1func_monotonicity_t monotonicity (const interval &i, unsigned int k) const
- virtual int differentiability (const interval &i, unsigned int k) const

- virtual int degree_update (int in_degree) const
- virtual bool order_is_supported (unsigned int k) const
- virtual unsigned int supported_eval_order () const
- virtual const func_info & ff (unsigned int k) const
- virtual bool operator== (const d1func &b) const
- virtual bool operator!= (const d1func &b) const
- const char * toString (const d1func::d1func_point_t &t) const

Public Attributes

- volatile int MAXLEN

Protected Member Functions

- virtual void raise_order (unsigned int neworder)

Protected Attributes

- double dod_left
- double dod_right
- int basic_diff
- std::vector< func_info > _f

10.61.1 Detailed Description

The d1func class is the base class for one dimensional functions. It implements methods for range enclosure of function values, first and second derivatives. Furthermore, backward propagation using an interval - Newton method is included.

10.61.2 Member Typedef Documentation

10.61.2.1 typedef std::triple<interval, interval, d1func_point_t> coco::d1func::point_info

The information on one point .first is the point, .second the value

Definition at line 110 of file d1func.h.

10.61.2.2 typedef std::vector<point_info> coco::d1func::point_list

a list of points for function value, first and second derivatives

Definition at line 113 of file d1func.h.

10.61.3 Member Enumeration Documentation

10.61.3.1 enum coco::d1func::d1func_point_t

The possible classes for special points in the lists: d1fp_regular: At this point the function is regular (for internal use only!) d1fp_extremum: The point is a local extremum d1fp_evenpoleup: At this point the function has a positive even pole d1fp_evenpoledn: At this point the function has a negative even pole d1fp_oddpoleup: At this point the function has an odd pole like 1/x at 0 d1fp_oddpoledn: At this point the function has an odd pole like -1/x at 0 d1fp_jumpldef: At this point the function has a jump (left side defined) d1fp_jumpludef: At this point the function has a jump (left side not defined) d1fp_point-: At this point the function has the specified value d1fp_undefined: In the given interval the function is undefined d1fp_wild: In the given interval the function has "wild" behaviour d1fp_jumprdef: At this point the function has a jump (right side defined) d1fp_jumprudef: At this point the function has a jump (right side not defined) d1fp_unknown: At this point the function has a unknown behaviour (for internal use only!)

Enumerator:

d1fp_unknown

d1fp_regular

d1fp_extremum

d1fp_evenpoleup

d1fp_evenpoledn

d1fp_oddpoleup

d1fp_oddpoledn

d1fp_jumpldef

d1fp_jumpludef

d1fp_point

d1fp_undefined

d1fp_wild

d1fp_jumprdef

d1fp_jumprudef

Definition at line 95 of file d1func.h.

10.61.3.2 enum coco::d1func::d1solve_t

The possible functions for 1D Newton solving: d1solve_0: f(x) = c d1solve_1: f(x) - f(z) - f'(x)(x-z) = 0 d1solve_2: 1/(x-z)(f(x)-f(z))(1+(x-w)(x-z)) - (f(w)-f(z))/(w-z) - (x-w)/(x-z) f'(x) = 0

Enumerator:

d1solve_0

d1solve_1

d1solve_2

Definition at line 106 of file d1func.h.

10.61.4 Constructor & Destructor Documentation

10.61.4.1 coco::d1func::d1func ( ) [inline]

Default Constructor

Definition at line 171 of file d1func.h.

10.61.4.2 coco::d1func::d1func (int _d) [inline]

Constructor setting basic differentiability

Definition at line 174 of file d1func.h.

10.61.4.3 coco::d1func::d1func (double l, double r) [inline]

Default Constructor setting domain of definition

Definition at line 177 of file d1func.h.

10.61.4.4 coco::d1func::d1func (double l, double r, int _d) [inline]

Default Constructor setting domain of definition and basic differentiability

Definition at line 180 of file d1func.h.

10.61.4.5 coco::d1func::d1func (const d1func & wh) [inline]

Copy Constructor

Definition at line 183 of file d1func.h.

10.61.4.6 virtual coco::d1func::~d1func ( ) [inline, virtual]

Standard Destructor

Definition at line 186 of file d1func.h.

10.61.5 Member Function Documentation

10.61.5.1 virtual uint64_t coco::d1func::chash ( ) [pure virtual]

the const hash value for hash calculation in the DAG

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.61.5.2 convex_info coco::d1func::convexity (const interval & i, unsigned int k) const [virtual]

return convexity information on `i`.

Definition at line 2308 of file d1func.cc.

10.61.5.3 virtual std::string coco::d1func::dag_out ( ) const [inline, virtual]

the parameters written in DAG format (it must contain necessary ':'s)

Reimplemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

Definition at line 205 of file d1func.h.

10.61.5.4 virtual int coco::d1func::degree_update (int *in_degree*) const [inline, virtual]

return degree information for in-argument degree `in_degree`.

Definition at line 457 of file d1func.h.

10.61.5.5 int coco::d1func::differentiability (const interval & *x*, unsigned int *k*) const [virtual]

return differentiability information on `i`.

Reimplemented in coco::hsf_func, and coco::xexp_func.

Definition at line 2351 of file d1func.cc.

10.61.5.6 virtual double coco::d1func::eval (double *x*, unsigned int *k*) const [pure virtual]

the point evaluation

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.61.5.7 diffNumber coco::d1func::eval (const diffNumber & *x*, unsigned int *k*) const [virtual]

the diffNumber evaluation

Reimplemented in coco::dag_d1func.

Definition at line 349 of file d1func.cc.

10.61.5.8 hessNumber coco::d1func::eval (const hessNumber & *x*, unsigned int *k*) const [virtual]

the hessNumber evaluation

Definition at line 334 of file d1func.cc.

10.61.5.9 interval coco::d1func::eval (const interval & *x_in*, unsigned int *k*) const [virtual]

the range evaluation

Definition at line 390 of file d1func.cc.

10.61.5.10 diffI coco::d1func::eval (const diffI & *x*, unsigned int *k*) const [virtual]

the interval diffNumber evaluation

Definition at line 715 of file d1func.cc.

10.61.5.11 ihessNumber coco::d1func::eval (const ihessNumber & *x*, unsigned int *k*) const [virtual]

the interval hessNumber evaluation

the hessNumber evaluation

Definition at line 411 of file d1func.cc.

10.61.5.12 Islope coco::d1func::eval (const Islope & *x*, unsigned int *k*) const [virtual]

the interval slope evaluation

the interval diffNumber evaluation

Definition at line 429 of file d1func.cc.

10.61.5.13 interval_set coco::d1func::eval (const interval_set & *x*, unsigned int *k*) const [virtual]

the range evaluation

Definition at line 794 of file d1func.cc.

10.61.5.14 interval coco::d1func::eval_slp (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual]

the interval first order slope evaluation at center z

Definition at line 1403 of file d1func.cc.

10.61.5.15 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *y*, const interval & *x*, unsigned int *k*) const [virtual]

the interval second order slope evaluation at centers z and y

Definition at line 1984 of file d1func.cc.

10.61.5.16 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual]

the interval second order slope evaluation at double center z

Definition at line 1991 of file d1func.cc.

10.61.5.17 virtual std::pair<double,double> coco::d1func::evald (double *x*, unsigned int *k*) const [inline, virtual]

the point and derivative evaluation

Reimplemented in coco::xexp_func, and coco::dag_d1func.

Definition at line 437 of file d1func.h.

10.61.5.18 virtual void coco::d1func::evalf (double *x*, double * *r*, unsigned int *k*) const [inline, virtual]

the point, first to third derivative evaluation

Reimplemented in coco::xexp_func.

Definition at line 443 of file d1func.h.

10.61.5.19 virtual std::triple<double,double,double> coco::d1func::evalt (double *x*, unsigned int *k*) const [inline, virtual]

the point and first and second derivative evaluation

Reimplemented in coco::xexp_func, and coco::dag_d1func.

Definition at line 440 of file d1func.h.

10.61.5.20 virtual const func_info& coco::d1func::ff (unsigned int *k*) const [inline, virtual]

retrieve the k th special points list

Definition at line 469 of file d1func.h.

10.61.5.21 virtual const char* coco::d1func::func_name ( ) const [pure virtual]

the name of the d1func.

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.61.5.22 virtual interval coco::d1func::imperfect_eval (const interval & *x*, unsigned int *k*) const [pure virtual]

the imperfect evaluation of k th derivative.

Implemented in coco::xexp_func, coco::hsf_func, and coco::dag_d1func.

10.61.5.23 void coco::d1func::internal_eval_d1fp (const diffI & *x*, diffI & *res*, const std::vector< d1func_point_t > & *ptin*, std::vector< d1func_point_t > & *ptres*, unsigned int *k*) const [virtual]

perform internal evaluations to update eventual point lists of "higher" functions.

Definition at line 168 of file d1func.cc.

10.61.5.24 interval_set coco::d1func::intersect_inv (const interval & *x*, const interval & *r*, unsigned int *k*) const [virtual]

back propagation

Definition at line 1708 of file d1func.cc.

10.61.5.25 interval_set coco::d1func::intersect_inv (const interval_set & *x*, const interval_set & *r*, unsigned int *k*) const [virtual]

back propagation

Definition at line 1778 of file d1func.cc.

10.61.5.26 d1func_monotonicity_t coco::d1func::monotonicity (const interval & *x*, unsigned int *k*) const [virtual]

return monotonicity information on `i`.

Definition at line 2185 of file d1func.cc.

10.61.5.27 virtual bool coco::d1func::operator!= (const d1func & *b*) const [inline, virtual]

the other comparison operator

Definition at line 476 of file d1func.h.

10.61.5.28 virtual bool coco::d1func::operator== (const d1func & *b*) const [inline, virtual]

the comparison operator

Reimplemented in coco::hsf_func, coco::xexp_func, and coco::dag_d1func.

Definition at line 473 of file d1func.h.

10.61.5.29 virtual bool coco::d1func::order_is_supported (unsigned int *k*) const [inline, virtual]

check whether evaluation of this order is supported.

Definition at line 460 of file d1func.h.

10.61.5.30 virtual std::ostream& coco::d1func::print_add (std::ostream & *o*) const [inline, virtual]

additional info written in DAG format (it must contain dag lines)

Reimplemented in coco::dag_d1func.

Definition at line 208 of file d1func.h.

10.61.5.31 virtual std::string coco::d1func::print_params ( ) const [inline, virtual]

the parameters written in printable form

Reimplemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

Definition at line 211 of file d1func.h.

10.61.5.32 virtual void coco::d1func::raise_order (unsigned int *neworder*) [inline, protected, virtual]

raise the order of the supported function evaluation to `neworder`

Reimplemented in coco::hsf_func, and coco::xexp_func.

Definition at line 215 of file d1func.h.

10.61.5.33 virtual unsigned int coco::d1func::supported_eval_order ( ) const [inline, virtual]

return the maximal evaluation order supported.

Definition at line 466 of file d1func.h.

10.61.5.34 virtual uint64_t coco::d1func::thash ( ) [pure virtual]

the pure hash value for hash calculation in the DAG

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.61.5.35 const char * coco::d1func::toString (const d1func::d1func_point_t & *t*) const

Definition at line 2415 of file d1func.cc.

10.61.6 Member Data Documentation

10.61.6.1 std::vector<func_info> coco::d1func::_f [protected]

the function info tables for function values and derivatives.

Definition at line 163 of file d1func.h.

10.61.6.2 int coco::d1func::basic_diff [protected]

basic differentiability

Definition at line 160 of file d1func.h.

10.61.6.3 double coco::d1func::dod_left [protected]

the left limit of the DOD

Definition at line 156 of file d1func.h.

10.61.6.4 double coco::d1func::dod_right [protected]

the right limit of the DOD

Definition at line 158 of file d1func.h.

10.61.6.5 volatile int coco::d1func::MAXLEN

maximal lenght of the interval set returned

Definition at line 167 of file d1func.h.

The documentation for this class was generated from the following files:

- d1func.h
- d1func.cc

10.62 coco::d1func_expr Class Reference

The d1func_expr class (one dimensional function expression)

`#include <d1func_expr.h>`

Public Member Functions

- std::string func_name () const
- unsigned int get_order () const
- std::string dag_out () const
- std::ostream & print_add (std::ostream &o) const
- std::string print_params () const
- d1func_expr (d1func *_d, unsigned int _k=0)
- ~d1func_expr ()
- double eval (double x, unsigned int order=0) const
- std::pair< double, double > evald (double x, unsigned int order=0) const
- std::triple< double, double, double > evalt (double x, unsigned int order=0) const
- void evalf (double x, double *r, unsigned int order=0) const
- diffNumber eval (const diffNumber &x, unsigned int order=0) const
- hessNumber eval (const hessNumber &x, unsigned int order=0) const
- interval eval (const interval &x, unsigned int order=0) const
- interval_set eval (const interval_set &x, unsigned int order=0) const
- interval eval_slp (const interval &z, const interval &x, unsigned int order=0) const
- interval eval_slp2 (const interval &z, const interval &y, const interval &x, unsigned int order=0) const
- interval eval_slp2 (const interval &z, const interval &x, unsigned int order=0) const
- diffI eval (const diffI &x, unsigned int order=0) const
- ihessNumber eval (const ihessNumber &x, unsigned int order=0) const
- Islope eval (const Islope &x, unsigned int order=0) const
- void internal_eval_d1fp (const diffI &x, diffI &res, const std::vector< d1func::d1func_point_t > &ptin, std::vector< d1func::d1func_point_t > &ptres, unsigned int order=0) const
- interval_set intersect_inv (const interval &x, const interval &r, unsigned int order=0) const
- interval_set intersect_inv (const interval_set &x, const interval_set &r, unsigned int order=0) const
- unsigned int supported_eval_order () const
- int degree_update (int in_degree) const
- bool operator== (const d1func_expr &b) const
- bool operator!= (const d1func_expr &b) const
- d1func_expr * differentiate (unsigned int order) const
- convex_e convex_update (const convex_e &c, const interval &i) const
- int differentiability (int diff, const interval &i) const

10.62.1 Detailed Description

The d1func_expr class is the class for one dimensional function expressions. It wraps the d1func class with an additional parameter for the order of the derivative.

10.62.2 Constructor & Destructor Documentation

10.62.2.1 coco::d1func_expr::d1func_expr (d1func * _d, unsigned int _k = 0) [inline]

constructor: note the _d should be allocated with new!

Definition at line 73 of file d1func_expr.h.

10.62.2.2 coco::d1func_expr::~d1func_expr ( ) [inline]

the destructor

Definition at line 76 of file d1func_expr.h.

10.62.3 Member Function Documentation

10.62.3.1 convex_e coco::d1func_expr::convex_update (const convex_e & c, const interval & i) const

update the convexity information for in argument convexity `c` on interval `i`.

Definition at line 35 of file d1func_expr.cc.

10.62.3.2 std::string coco::d1func_expr::dag_out ( ) const [inline]

the parameters written in DAG format (it must contain necessary ':'s)

Definition at line 63 of file d1func_expr.h.

10.62.3.3 int coco::d1func_expr::degree_update (int in_degree) const [inline]

update the degree for in-argument degree `in_degree`.

Definition at line 143 of file d1func_expr.h.

10.62.3.4 int coco::d1func_expr::differentiability (int diff, const interval & i) const

return differentiability information on `i`.

Definition at line 88 of file d1func_expr.cc.

10.62.3.5 d1func_expr* coco::d1func_expr::differentiate (unsigned int order) const [inline]

differentiate a d1func_expr

Definition at line 154 of file d1func_expr.h.

10.62.3.6 double coco::d1func_expr::eval (double x, unsigned int order = 0) const [inline]

the point evaluation

Definition at line 80 of file d1func_expr.h.

10.62.3.7 diffNumber coco::d1func_expr::eval (const diffNumber & *x*, unsigned int *order* = 0) const [inline]

the differential number evaluation

Definition at line 92 of file d1func_expr.h.

10.62.3.8 hessNumber coco::d1func_expr::eval (const hessNumber & *x*, unsigned int *order* = 0) const [inline]

the Hessian number evaluation

Definition at line 95 of file d1func_expr.h.

10.62.3.9 interval coco::d1func_expr::eval (const interval & *x*, unsigned int *order* = 0) const [inline]

the range evaluation

Definition at line 98 of file d1func_expr.h.

10.62.3.10 interval_set coco::d1func_expr::eval (const interval_set & *x*, unsigned int *order* = 0) const [inline]

the range evaluation

Definition at line 101 of file d1func_expr.h.

10.62.3.11 diffI coco::d1func_expr::eval (const diffI & *x*, unsigned int *order* = 0) const [inline]

the interval differential number evaluation

Definition at line 115 of file d1func_expr.h.

10.62.3.12 ihessNumber coco::d1func_expr::eval (const ihessNumber & *x*, unsigned int *order* = 0) const [inline]

the interval Hessian number evaluation

Definition at line 118 of file d1func_expr.h.

10.62.3.13 Islope coco::d1func_expr::eval (const Islope & *x*, unsigned int *order* = 0) const [inline]

the interval slope evaluation

Definition at line 121 of file d1func_expr.h.

10.62.3.14 interval coco::d1func_expr::eval_slp (const interval & *z*, const interval & *x*, unsigned int *order* = 0) const [inline]

the interval first order slope evaluation at center `z`

Definition at line 104 of file d1func_expr.h.

10.62.3.15 interval coco::d1func_expr::eval_slp2 (const interval & *z*, const interval & *y*, const interval & *x*, unsigned int *order* = 0) const [inline]

the interval second order slope evaluation at centers z and y

Definition at line 107 of file d1func_expr.h.

10.62.3.16 interval coco::d1func_expr::eval_slp2 (const interval & *z*, const interval & *x*, unsigned int *order* = 0) const [inline]

the interval second order slope evaluation at double center z

Definition at line 111 of file d1func_expr.h.

10.62.3.17 std::pair<double,double> coco::d1func_expr::evald (double *x*, unsigned int *order* = 0) const [inline]

the point and first derivative evaluation

Definition at line 83 of file d1func_expr.h.

10.62.3.18 void coco::d1func_expr::evalf (double *x*, double * *r*, unsigned int *order* = 0) const [inline]

the point, and first to third derivative evaluation

Definition at line 89 of file d1func_expr.h.

10.62.3.19 std::triple<double,double,double> coco::d1func_expr::evalt (double *x*, unsigned int *order* = 0) const [inline]

the point, first and second derivative evaluation

Definition at line 86 of file d1func_expr.h.

10.62.3.20 std::string coco::d1func_expr::func_name ( ) const [inline]

the name of the d1func.

Definition at line 57 of file d1func_expr.h.

10.62.3.21 unsigned int coco::d1func_expr::get_order ( ) const [inline]

the order of the derivative of the d1func.

Definition at line 60 of file d1func_expr.h.

10.62.3.22 void coco::d1func_expr::internal_eval_d1fp (const diffI & *x*, diffI & *res*, const std::vector< d1func::d1func_point_t > & *ptin*, std::vector< d1func::d1func_point_t > & *ptres*, unsigned int *order* = 0) const [inline]

Definition at line 124 of file d1func_expr.h.

10.62.3.23 interval_set coco::d1func_expr::intersect_inv (const interval & *x*, const interval & *r*, unsigned int *order* = 0) const [inline]

back propagation

Definition at line 130 of file d1func_expr.h.

10.62.3.24 interval_set coco::d1func_expr::intersect_inv (const interval_set & *x*, const interval_set & *r*, unsigned int *order* = 0) const [inline]

back propagation

Definition at line 134 of file d1func_expr.h.

10.62.3.25 bool coco::d1func_expr::operator!= (const d1func_expr & *b*) const [inline]

the other comparison operator

Definition at line 150 of file d1func_expr.h.

10.62.3.26 bool coco::d1func_expr::operator== (const d1func_expr & *b*) const [inline]

the comparison operator

Definition at line 146 of file d1func_expr.h.

10.62.3.27 std::ostream& coco::d1func_expr::print_add (std::ostream & *o*) const [inline]

additional info written in DAG format (it must contain dag lines)

Definition at line 66 of file d1func_expr.h.

10.62.3.28 std::string coco::d1func_expr::print_params ( ) const [inline]

the parameters written in printable form

Definition at line 69 of file d1func_expr.h.

10.62.3.29 unsigned int coco::d1func_expr::supported_eval_order ( ) const [inline]

the maximal order of derivatives which can be evaluated.

Definition at line 139 of file d1func_expr.h.

The documentation for this class was generated from the following files:

- d1func_expr.h
- d1func_expr.cc

10.63 coco::d1func_visitor_0 Class Reference

Public Member Functions

- d1func_visitor_0 ()

- d1func_visitor_0 (std::vector< bool > *__vt, std::vector< bool > *__cvt, std::vector< uint64_t > *__hsh, std::vector< uint64_t > *__chsh, std::set< unsigned int > &__in, model *m)
- d1func_visitor_0 (const d1func_visitor_0 &__x)
- void vinit ()
- void init ()
- bool postorder (expression_node &r)
- bool preorder (expression_node &r)
- d1func_visitor_0 value ()
- d1func_visitor_0 vvalue ()
- void vcollect (const d1func_visitor_0 &__s)
- void collect (expression_node &r, const d1func_visitor_0 &__s)
- void set_base_node (unsigned int bn)

10.63.1 Detailed Description

This class is a postorder visitor used for the phase 0 of the d1func converter.

10.63.2 Constructor & Destructor Documentation

10.63.2.1 coco::d1func_visitor_0::d1func_visitor_0 ( ) [inline]

Definition at line 66 of file model_d1simplify.cc.

10.63.2.2 coco::d1func_visitor_0::d1func_visitor_0 (std::vector< bool > * *__vt*, std::vector< bool > * *__cvt*, std::vector< uint64_t > * *__hsh*, std::vector< uint64_t > * *__chsh*, std::set< unsigned int > & *__in*, model * *m*) [inline]

Definition at line 70 of file model_d1simplify.cc.

10.63.2.3 coco::d1func_visitor_0::d1func_visitor_0 (const d1func_visitor_0 & *__x*) [inline]

Definition at line 78 of file model_d1simplify.cc.

10.63.3 Member Function Documentation

10.63.3.1 void coco::d1func_visitor_0::collect (expression_node & *r*, const d1func_visitor_0 & *__s*)

Definition at line 127 of file model_d1simplify.cc.

10.63.3.2 void coco::d1func_visitor_0::init ( ) [inline]

Definition at line 85 of file model_d1simplify.cc.

10.63.3.3 bool coco::d1func_visitor_0::postorder (expression_node & *r*)

Definition at line 120 of file model_d1simplify.cc.

10.63.3.4 bool coco::d1func_visitor_0::preorder (expression_node & *r*) [inline]

Definition at line 89 of file model_d1simplify.cc.

10.63.3.5 void coco::d1func_visitor_0::set_base_node (unsigned int *bn*) [inline]

Definition at line 117 of file model_d1simplify.cc.

10.63.3.6 d1func_visitor_0 coco::d1func_visitor_0::value ( ) [inline]

Definition at line 110 of file model_d1simplify.cc.

10.63.3.7 void coco::d1func_visitor_0::vcollect (const d1func_visitor_0 & *__s*)

Definition at line 302 of file model_d1simplify.cc.

10.63.3.8 void coco::d1func_visitor_0::vinit ( ) [inline]

Definition at line 84 of file model_d1simplify.cc.

10.63.3.9 d1func_visitor_0 coco::d1func_visitor_0::vvalue ( ) [inline]

Definition at line 111 of file model_d1simplify.cc.

The documentation for this class was generated from the following file:

- model_d1simplify.cc

10.64 coco::d1func_visitor_1 Class Reference

Public Member Functions

- d1func_visitor_1 (std::ostream &r)
- d1func_visitor_1 (std::ostream &r, model *m)
- d1func_visitor_1 (const d1func_visitor_1 &__x)
- void vinit ()
- void init ()
- bool postorder (expression_node &r)
- bool preorder (expression_node &r)
- bool value ()
- bool vvalue ()
- void vcollect (const bool &__s)
- void collect (expression_node &r, const bool &__s)
- void set_base_node (unsigned int bn)

10.64.1 Detailed Description

This class is a postorder visitor used for the phase 1 of the d1func converter.

10.64.2 Constructor & Destructor Documentation

10.64.2.1 coco::d1func_visitor_1::d1func_visitor_1 (std::ostream & *r*) [inline]

Definition at line 317 of file model_d1simplify.cc.

10.64.2.2 coco::d1func_visitor_1::d1func_visitor_1 (std::ostream & *r,* model * *m*) [inline]

Definition at line 321 of file model_d1simplify.cc.

10.64.2.3 coco::d1func_visitor_1::d1func_visitor_1 (const d1func_visitor_1 & *__x*) [inline]

Definition at line 326 of file model_d1simplify.cc.

10.64.3 Member Function Documentation

10.64.3.1 void coco::d1func_visitor_1::collect (expression_node & *r,* const bool & *__s*)

Definition at line 506 of file model_d1simplify.cc.

10.64.3.2 void coco::d1func_visitor_1::init ( ) [inline]

Definition at line 331 of file model_d1simplify.cc.

10.64.3.3 bool coco::d1func_visitor_1::postorder (expression_node & *r*)

Definition at line 366 of file model_d1simplify.cc.

10.64.3.4 bool coco::d1func_visitor_1::preorder (expression_node & *r*) [inline]

Definition at line 335 of file model_d1simplify.cc.

10.64.3.5 void coco::d1func_visitor_1::set_base_node (unsigned int *bn*) [inline]

Definition at line 363 of file model_d1simplify.cc.

10.64.3.6 bool coco::d1func_visitor_1::value ( ) [inline]

Definition at line 356 of file model_d1simplify.cc.

10.64.3.7 void coco::d1func_visitor_1::vcollect (const bool & *__s*) [inline]

Definition at line 359 of file model_d1simplify.cc.

10.64.3.8 void coco::d1func_visitor_1::vinit ( ) [inline]

Definition at line 330 of file model_d1simplify.cc.

10.64.3.9 bool coco::d1func_visitor_1::vvalue ( ) [inline]

Definition at line 357 of file model_d1simplify.cc.

The documentation for this class was generated from the following file:

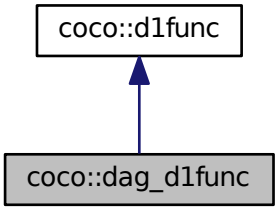
- model_d1simplify.cc

10.65 coco::dag_d1func Class Reference

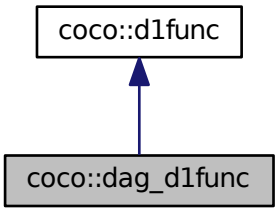
The dag_d1func class (one dimensional function)

`#include <dagd1func.h>`

Inheritance diagram for coco::dag_d1func:



Collaboration diagram for coco::dag_d1func:



Public Types

- enum d1func_point_t { d1fp_unknown = -2, d1fp_regular = -1, d1fp_extremum = 0, d1fp_evenpoleup = 1, d1fp_evenpoledn = 2, d1fp_oddpoleup = 3, d1fp_oddpoledn = 4, d1fp_jumpldef = 5, d1fp_jumpludef = 6, d1fp_point = 7, d1fp_undefined = 8, d1fp_wild = 9, d1fp_jumprdef = 10, d1fp_jumprudef = 11 }

- enum d1solve_t { d1solve_0, d1solve_1, d1solve_2 }
- typedef std::triple< interval, interval, d1func_point_t > point_info
- typedef std::vector< point_info > point_list

Public Member Functions

- virtual uint64_t thash ()
- virtual uint64_t chash ()
- virtual const char * func_name () const
- std::string dag_out () const
- virtual std::ostream & print_add (std::ostream &o) const
- virtual std::string print_params () const
- virtual double eval (double x, unsigned int k) const
- virtual std::pair< double, double > evald (double x, unsigned int k) const
- virtual std::triple< double, double, double > evalt (double x, unsigned int k) const
- virtual interval imperfect_eval (const interval &x, unsigned int k) const
- virtual diffNumber eval (const diffNumber &x, unsigned int k) const
- virtual bool operator== (const d1func &b) const
- virtual ~dag_d1func ()
- dag_d1func (model *m, const interval &borders, int PREPORDER=-1)
- dag_d1func (const char *name, const interval &borders, int PREPORDER=-1)
- virtual hessNumber eval (const hessNumber &x, unsigned int k) const
- virtual interval eval (const interval &x, unsigned int k) const
- virtual diffI eval (const diffI &x, unsigned int k) const
- virtual ihessNumber eval (const ihessNumber &x, unsigned int k) const
- virtual Islope eval (const Islope &x, unsigned int k) const
- virtual interval_set eval (const interval_set &x, unsigned int k) const
- virtual interval eval_slp (const interval &z, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &y, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &x, unsigned int k) const
- virtual void internal_eval_d1fp (const diffI &x, diffI &res, const std::vector< d1func_point_t > &ptin, std::vector< d1func_point_t > &ptres, unsigned int order) const
- virtual interval_set intersect_inv (const interval &x, const interval &r, unsigned int k) const
- virtual interval_set intersect_inv (const interval_set &x, const interval_set &r, unsigned int k) const
- virtual void evalf (double x, double *r, unsigned int k) const
- virtual convex_info convexity (const interval &i, unsigned int k) const
- virtual d1func_monotonicity_t monotonicity (const interval &i, unsigned int k) const
- virtual int differentiability (const interval &i, unsigned int k) const
- virtual int degree_update (int in_degree) const
- virtual unsigned int supported_eval_order () const
- virtual const func_info & ff (unsigned int k) const
- virtual bool operator!= (const d1func &b) const
- const char * toString (const d1func::d1func_point_t &t) const

Public Attributes

- volatile int MAXLEN

Protected Attributes

- double dod_left
- double dod_right
- int basic_diff
- std::vector< func_info > _f

10.65.1 Detailed Description

dag_d1func is a one dimensional function defined by a DAG.

10.65.2 Member Typedef Documentation

10.65.2.1 typedef std::triple<interval, interval, d1func_point_t> coco::d1func::point_info [inherited]

The information on one point .first is the point, .second the value

Definition at line 110 of file d1func.h.

10.65.2.2 typedef std::vector<point_info> coco::d1func::point_list [inherited]

a list of points for function value, first and second derivatives

Definition at line 113 of file d1func.h.

10.65.3 Member Enumeration Documentation

10.65.3.1 enum coco::d1func::d1func_point_t [inherited]

The possible classes for special points in the lists: d1fp_regular: At this point the function is regular (for internal use only!) d1fp_extremum: The point is a local extremum d1fp_evenpoleup: At this point the function has a positive even pole d1fp_evenpoledn: At this point the function has a negative even pole d1fp_oddpoleup: At this point the function has an odd pole like 1/x at 0 d1fp_oddpoledn: At this point the function has an odd pole like -1/x at 0 d1fp_jumpldef: At this point the function has a jump (left side defined) d1fp_jumpludef: At this point the function has a jump (left side not defined) d1fp_point- : At this point the function has the specified value d1fp_undefined: In the given interval the function is undefined d1fp_wild: In the given interval the function has "wild" behaviour d1fp_jumprdef: At this point the function has a jump (right side defined) d1fp_jumprudef: At this point the function has a jump (right side not defined) d1fp_unknown: At this point the function has a unknown behaviour (for internal use only!)

Enumerator:

d1fp_unknown

d1fp_regular

d1fp_extremum

d1fp_evenpoleup

d1fp_evenpoledn

d1fp_oddpoleup
d1fp_oddpoledn
d1fp_jumpldef
d1fp_jumpludef
d1fp_point
d1fp_undefined
d1fp_wild
d1fp_jumprdef
d1fp_jumprudef

Definition at line 95 of file d1func.h.

10.65.3.2 enum coco::d1func::d1solve_t [inherited]

The possible functions for 1D Newton solving: d1solve_0: f(x) = c d1solve_1: f(x) - f(z) - f'(x)(x-z) = 0 d1solve_2: 1/(x-z)(f(x)-f(z))(1+(x-w)(x-z)) - (f(w)-f(z))/(w-z) - (x-w)/(x-z) f'(x) = 0

Enumerator:

d1solve_0
d1solve_1
d1solve_2

Definition at line 106 of file d1func.h.

10.65.4 Constructor & Destructor Documentation

10.65.4.1 coco::dag_d1func::~dag_d1func ( ) [virtual]

standard destructor

Definition at line 79 of file dagd1func.cc.

10.65.4.2 coco::dag_d1func::dag_d1func (model * *m*, const interval & *borders*, int *PREPORDER* = -1) [inline]

this constructs a new dag_d1func from a model.

Definition at line 154 of file dagd1func.h.

10.65.4.3 coco::dag_d1func::dag_d1func (const char * *name*, const interval & *borders*, int *PREPORDER* = -1) [inline]

this constructs a new dag_d1func from a file.

Definition at line 163 of file dagd1func.h.

10.65.5 Member Function Documentation

10.65.5.1 virtual uint64_t coco::dag_d1func::chash ( ) [inline, virtual]

the const hash value for hash calculation in the DAG

Implements coco::d1func.

Definition at line 61 of file dagd1func.h.

10.65.5.2 convex_info coco::d1func::convexity (const interval & *i*, unsigned int *k*) const [virtual, inherited]

return convexity information on `i`.

Definition at line 2308 of file d1func.cc.

10.65.5.3 std::string coco::dag_d1func::dag_out ( ) const [virtual]

the parameters written in DAG format (it must contain necessary ':'s)

Reimplemented from coco::d1func.

Definition at line 104 of file dagd1func.cc.

10.65.5.4 virtual int coco::d1func::degree_update (int *in_degree*) const [inline, virtual, inherited]

return degree information for in-argument degree `in_degree`.

Definition at line 457 of file d1func.h.

10.65.5.5 int coco::d1func::differentiability (const interval & *x*, unsigned int *k*) const [virtual, inherited]

return differentiability information on `i`.

Reimplemented in coco::hsf_func, and coco::xexp_func.

Definition at line 2351 of file d1func.cc.

10.65.5.6 double coco::dag_d1func::eval (double *x*, unsigned int *k*) const [virtual]

the imperfect function evaluations

Implements coco::d1func.

Definition at line 108 of file dagd1func.cc.

10.65.5.7 diffNumber coco::dag_d1func::eval (const diffNumber & *x*, unsigned int *k*) const [virtual]

the diffNumber evaluation

Reimplemented from coco::d1func.

Definition at line 138 of file dagd1func.cc.

10.65.5.8 hessNumber coco::d1func::eval (const hessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the hessNumber evaluation

Definition at line 334 of file d1func.cc.

10.65.5.9 interval coco::d1func::eval (const interval & *x_in*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 390 of file d1func.cc.

10.65.5.10 diffI coco::d1func::eval (const diffI & *x*, unsigned int *k*) const [virtual, inherited]

the interval diffNumber evaluation

Definition at line 715 of file d1func.cc.

10.65.5.11 ihessNumber coco::d1func::eval (const ihessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the interval hessNumber evaluation

the hessNumber evaluation

Definition at line 411 of file d1func.cc.

10.65.5.12 Islope coco::d1func::eval (const Islope & *x*, unsigned int *k*) const [virtual, inherited]

the interval slope evaluation

the interval diffNumber evaluation

Definition at line 429 of file d1func.cc.

10.65.5.13 interval_set coco::d1func::eval (const interval_set & *x*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 794 of file d1func.cc.

10.65.5.14 interval coco::d1func::eval_slp (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval first order slope evaluation at center `z`

Definition at line 1403 of file d1func.cc.

10.65.5.15 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *y*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at centers `z` and `y`

Definition at line 1984 of file d1func.cc.

10.65.5.16 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at double center z

Definition at line 1991 of file d1func.cc.

10.65.5.17 std::pair< double, double > coco::dag_d1func::evald (double *x*, unsigned int *k*) const [virtual]

the point and derivative evaluation

Reimplemented from coco::d1func.

Definition at line 168 of file dagd1func.cc.

10.65.5.18 virtual void coco::d1func::evalf (double *x*, double * *r*, unsigned int *k*) const [inline, virtual, inherited]

the point, first to third derivative evaluation

Reimplemented in coco::xexp_func.

Definition at line 443 of file d1func.h.

10.65.5.19 std::triple< double, double, double > coco::dag_d1func::evalt (double *x*, unsigned int *k*) const [virtual]

the point and derivative evaluation

Reimplemented from coco::d1func.

Definition at line 186 of file dagd1func.cc.

10.65.5.20 virtual const func_info& coco::d1func::ff (unsigned int *k*) const [inline, virtual, inherited]

retrieve the k th special points list

Definition at line 469 of file d1func.h.

10.65.5.21 virtual const char* coco::dag_d1func::func_name ( ) const [inline, virtual]

the name of the d1func.

Implements coco::d1func.

Definition at line 64 of file dagd1func.h.

10.65.5.22 interval coco::dag_d1func::imperfect_eval (const interval & *x*, unsigned int *k*) const [virtual]

the imperfect function evaluations

Implements coco::d1func.

Definition at line 245 of file dagd1func.cc.

10.65.5.23 void coco::d1func::internal_eval_d1fp (const diffI & *x*, diffI & *res*, const std::vector< d1func_point_t > & *ptin*, std::vector< d1func_point_t > & *ptres*, unsigned int *k*) const [virtual, inherited]

perform internal evaluations to update eventual point lists of "higher" functions.

Definition at line 168 of file d1func.cc.

10.65.5.24 interval_set coco::d1func::intersect_inv (const interval & *x*, const interval & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1708 of file d1func.cc.

10.65.5.25 interval_set coco::d1func::intersect_inv (const interval_set & *x*, const interval_set & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1778 of file d1func.cc.

10.65.5.26 d1func_monotonicity_t coco::d1func::monotonicity (const interval & *x*, unsigned int *k*) const [virtual, inherited]

return monotonicity information on `i`.

Definition at line 2185 of file d1func.cc.

10.65.5.27 virtual bool coco::d1func::operator!= (const d1func & *b*) const [inline, virtual, inherited]

the other comparison operator

Definition at line 476 of file d1func.h.

10.65.5.28 virtual bool coco::dag_d1func::operator== (const d1func & *b*) const [inline, virtual]

the comparison operator

Reimplemented from coco::d1func.

Definition at line 91 of file dagd1func.h.

10.65.5.29 virtual std::ostream& coco::dag_d1func::print_add (std::ostream & *o*) const [inline, virtual]

additional info written in DAG format (it must contain dag lines)

Reimplemented from coco::d1func.

Definition at line 70 of file dagd1func.h.

10.65.5.30 virtual std::string coco::dag_d1func::print_params ( ) const [inline, virtual]

the parameters written in printable form

Reimplemented from coco::d1func.

Definition at line 74 of file dagd1func.h.

10.65.5.31 virtual unsigned int coco::d1func::supported_eval_order ( ) const [inline, virtual, inherited]

return the maximal evaluation order supported.

Definition at line 466 of file d1func.h.

10.65.5.32 virtual uint64_t coco::dag_d1func::thash ( ) [inline, virtual]

the pure hash value for hash calculation in the DAG

Implements coco::d1func.

Definition at line 58 of file dagd1func.h.

10.65.5.33 const char * coco::d1func::toString (const d1func::d1func_point_t & *t*) const [inherited]

Definition at line 2415 of file d1func.cc.

10.65.6 Member Data Documentation

10.65.6.1 std::vector<func_info> coco::d1func::_f [protected, inherited]

the function info tables for function values and derivatives.

Definition at line 163 of file d1func.h.

10.65.6.2 int coco::d1func::basic_diff [protected, inherited]

basic differentiability

Definition at line 160 of file d1func.h.

10.65.6.3 double coco::d1func::dod_left [protected, inherited]

the left limit of the DOD

Definition at line 156 of file d1func.h.

10.65.6.4 double coco::d1func::dod_right [protected, inherited]

the right limit of the DOD

Definition at line 158 of file d1func.h.

10.65.6.5 volatile int coco::d1func::MAXLEN [inherited]

maximal lenght of the interval set returned

Definition at line 167 of file d1func.h.

The documentation for this class was generated from the following files:

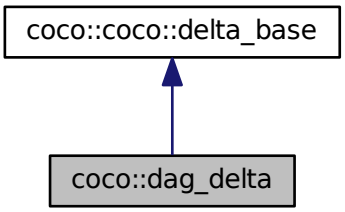
- dagd1func.h
- dagd1func.cc
- dagd1func_bp.cc

10.66 coco::dag_delta Class Reference

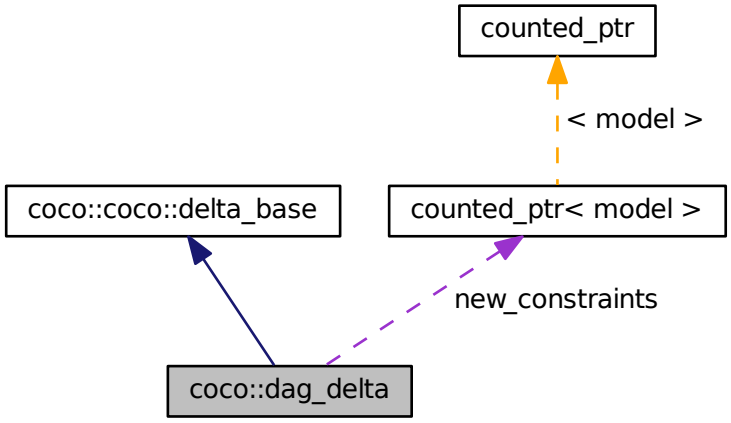
The DAG delta class for performing changes to the DAG of a model.

`#include <dag_delta.h>`

Inheritance diagram for coco::dag_delta:



Collaboration diagram for coco::dag_delta:



Classes

- class __check_nodes
- struct __docompare_nodes

Public Member Functions

- dag_delta (const std::string &__a, bool full=false)
- dag_delta (const std::string &__a, model *__nc, bool full=false)
- dag_delta (const dag_delta &__d)
- ~dag_delta ()
- dag_delta * new_copy () const
- void destroy_copy (delta_base *__d) const
- void add_new (model *__m)
- void add_new (model &__m)
- void remove (const walker &_nn)
- void remove (const std::vector< walker > &_nn)
- void unkeep ()
- bool apply (work_node &x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const dag_delta &_c) const
- bool operator!= (const dag_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)

- const std::string & get_action () const
- const std::string & get_action () const
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Public Attributes

- counted_ptr< model > new_constraints
- std::vector< walker > rm_nodes
- bool is_full_delta

Protected Attributes

- std::string _action

Friends

- class dag_undelta

10.66.1 Detailed Description

This class is used to specify information for changes to the DAG of a model.

10.66.2 Constructor & Destructor Documentation

10.66.2.1 coco::dag_delta::dag_delta (const std::string & __a, bool full = false) [inline]

Constructor setting the action identifier to __a and the is_full_delta member to full.

Definition at line 190 of file dag_delta.h.

10.66.2.2 coco::dag_delta::dag_delta (const std::string & __a, model * __nc, bool full = false) [inline]

Constructor setting the action identifier to __a and the is_full_delta member to full. Furthermore, the new_constraints member is initialized with __nc.

Definition at line 198 of file dag_delta.h.

10.66.2.3 coco::dag_delta::dag_delta (const dag_delta & __d) [inline]

Standard Copy Constructor

Definition at line 205 of file dag_delta.h.

10.66.2.4 coco::dag_delta::~dag_delta ( ) [inline]

Standard Destructor

Definition at line 216 of file dag_delta.h.

10.66.3 Member Function Documentation

10.66.3.1 void coco::dag_delta::add_new (model * __m) [inline]

This method initializes the new_constraints member.

Definition at line 233 of file dag_delta.h.

10.66.3.2 void coco::dag_delta::add_new (model & __m) [inline]

This method initializes the new_constraints member.

Definition at line 238 of file dag_delta.h.

10.66.3.3 virtual bool coco::coco::delta_base::apply (work_node & _x, undelta_base *& _u, const delta_id & _di, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.66.3.4 bool coco::dag_delta::apply (work_node & x, undelta_base *& _u, const delta_id & _did, size_t & delta_size) const

Apply the delta with delta_id _d to work node _x, hereby changing the bounds as requested. save node numbers corresponding to the various new things

Definition at line 112 of file dag_delta.cc.

10.66.3.5 bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.66.3.6 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.66.3.7 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.66.3.8 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.66.3.9 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.66.3.10 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.66.3.11 void coco::dag_delta::destroy_copy (delta_base * __d) const [inline]

Clone Destructor

Definition at line 230 of file dag_delta.h.

10.66.3.12 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.66.3.13 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.66.3.14 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.66.3.15 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.66.3.16 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.66.3.17 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.66.3.18 dag_delta* coco::dag_delta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::delta_base.

Definition at line 228 of file dag_delta.h.

10.66.3.19 bool coco::dag_delta::operator!= (const delta_base & *_c*) const [inline]

Definition at line 311 of file dag_delta.h.

10.66.3.20 bool coco::dag_delta::operator!= (const dag_delta & *_c*) const [inline]

Definition at line 318 of file dag_delta.h.

10.66.3.21 bool coco::dag_delta::operator== (const delta_base & *_c*) const [inline]

Comparison operators

Definition at line 308 of file dag_delta.h.

10.66.3.22 bool coco::dag_delta::operator== (const dag_delta & _c) const [inline]

Comparison operators

Definition at line 316 of file dag_delta.h.

10.66.3.23 void coco::dag_delta::remove (const walker & _nn) [inline]

This method adds another node _nn to be removed.

Definition at line 244 of file dag_delta.h.

10.66.3.24 void coco::dag_delta::remove (const std::vector< walker > & _nn) [inline]

This method adds the nodes _nn to the vector of nodes which are to be removed.

Definition at line 247 of file dag_delta.h.

10.66.3.25 void coco::dag_delta::unkeep ( ) [inline, virtual]

Free this delta from the responsibility to remove the nodes from the memory upon destruction of the delta.

Reimplemented from coco::coco::delta_base.

Definition at line 252 of file dag_delta.h.

10.66.4 Friends And Related Function Documentation

10.66.4.1 friend class dag_undelta [friend]

Definition at line 321 of file dag_delta.h.

10.66.5 Member Data Documentation

10.66.5.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

10.66.5.2 bool coco::dag_delta::is_full_delta

If this variable is true, the new_constraints member contains a full model, which is used to replace the model of the work_node when the delta is applied.

Definition at line 186 of file dag_delta.h.

10.66.5.3 counted_ptr<model> coco::dag_delta::new_constraints

This DAG will be added to the DAG of the work node when applied. The ghost nodes in this DAG are used to overlay the nodes with the same node number in the original DAG.

Definition at line 175 of file dag_delta.h.

10.66.5.4 std::vector<walker> coco::dag_delta::rm_nodes

This variable holds the nodes which are to be removed from the work node DAG. Actually, the minimal subgraph generated by the nodes is removed, i.e. the subgraph containing all nodes in rm_nodes and all nodes, which are only connected to them.

Definition at line 181 of file dag_delta.h.

The documentation for this class was generated from the following files:

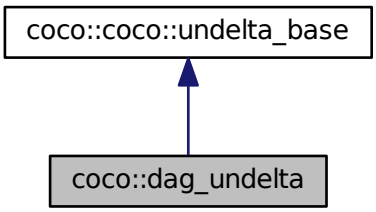
- dag_delta.h
- dag_delta.cc

10.67 coco::dag_undelta Class Reference

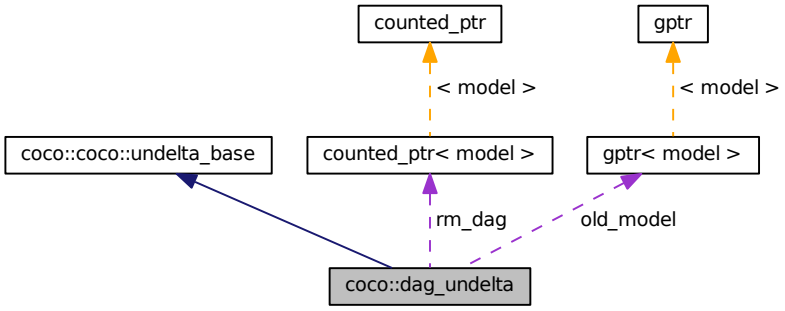
The DAG undelta class for undoing changes to the DAG of a model.

`#include <dag_delta.h>`

Inheritance diagram for coco::dag_undelta:



Collaboration diagram for coco::dag_undelta:



Classes

- class __check_walkers

Public Member Functions

- dag_undelta (bool full=false)
- dag_undelta (gptr< model > *__nc)
- dag_undelta (const dag_undelta &_du)
- ~dag_undelta ()
- dag_undelta * new_copy () const
- void destroy_copy (undelta_base *__d) const
- bool unapply (work_node &x, const delta_id &_did) const
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

Public Attributes

- gptr< model > * old_model
- counted_ptr< model > rm_dag
- std::vector< model::enhanced_edge > rm_e
- std::vector< walker > added_nodes
- std::vector< walker > added_objectives
- std::vector< walker > added_constraints
- std::vector< walker > added_ghosts
- std::vector< walker > added_vars
- std::map< unsigned int, interval > bounds_chgd
- bool is_full_undelta

Friends

- class dag_delta

10.67.1 Detailed Description

This class is used to specify undo information for changes to the DAG of a model. This class undoes the apply of a dag_delta.

10.67.2 Constructor & Destructor Documentation

10.67.2.1 coco::dag_undelta::dag_undelta (bool *full* = `false`) `[inline]`

Standard Constructor, setting the is_full_undelta member to `full`.

Definition at line 103 of file dag_delta.h.

10.67.2.2 coco::dag_undelta::dag_undelta (gptr< model > * *__nc*) `[inline]`

Constructor, setting the is_full_undelta member to `true` and storing the original model `__nc`.

Definition at line 114 of file dag_delta.h.

10.67.2.3 coco::dag_undelta::dag_undelta (const dag_undelta & *_du*) `[inline]`

Standard Copy Constructor

Definition at line 124 of file dag_delta.h.

10.67.2.4 coco::dag_undelta::~dag_undelta ( ) `[inline]`

Standard Destructor

Definition at line 141 of file dag_delta.h.

10.67.3 Member Function Documentation

10.67.3.1 void coco::dag_undelta::destroy_copy (undelta_base * *__d*) const `[inline]`

Clone Destructor

Definition at line 152 of file dag_delta.h.

10.67.3.2 undelta coco::coco::undelta_base::make_undelta (size_t *s*) `[inline, inherited]`

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.67.3.3 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.67.3.4 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.67.3.5 dag_undelta* coco::dag_undelta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::undelta_base.

Definition at line 150 of file dag_delta.h.

10.67.3.6 bool coco::dag_undelta::unapply (work_node & *x*, const delta_id & *did*) const [virtual]

Undo the dag_delta with delta_id _i in work node _x

Reimplemented from coco::coco::undelta_base.

Definition at line 342 of file dag_delta.cc.

10.67.3.7 virtual bool coco::coco::undelta_base::unapply3 (work_node & *x*, const work_node & *y*, const delta_id & *d*) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.67.3.8 virtual bool coco::coco::undelta_base::unapply3 (work_node & *x*, const work_node & *y*, const delta_id & *d*) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.67.3.9 bool coco::coco::undelta_base::unapply3 (work_node & *x*, const work_node & *y*, const delta_id & *d*) const [inline, virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

Definition at line 94 of file api_delta.h.

10.67.4 Friends And Related Function Documentation

10.67.4.1 friend class dag_delta [friend]

Definition at line 157 of file dag_delta.h.

10.67.5 Member Data Documentation

10.67.5.1 std::vector<walker> coco::dag_undelta::added_constraints

The vector of constraints added by the dag_delta

Definition at line 64 of file dag_delta.h.

10.67.5.2 std::vector<walker> coco::dag_undelta::added_ghosts

The vector of ghosts added by the dag_delta

Definition at line 66 of file dag_delta.h.

10.67.5.3 std::vector<walker> coco::dag_undelta::added_nodes

The vector of nodes added by the dag_delta

Definition at line 60 of file dag_delta.h.

10.67.5.4 std::vector<walker> coco::dag_undelta::added_objectives

The vector of objectives added by the dag_delta

Definition at line 62 of file dag_delta.h.

10.67.5.5 std::vector<walker> coco::dag_undelta::added_vars

The vector of variables added by the dag_delta

Definition at line 68 of file dag_delta.h.

10.67.5.6 std::map<unsigned int, interval> coco::dag_undelta::bounds_chgd

This map maps node numbers of ghosts overlaid over other nodes to the bounds of the nodes being overlaid prior to the application of the dag_delta.

Definition at line 73 of file dag_delta.h.

10.67.5.7 bool coco::dag_undelta::is_full_undelta

This variable is used to decide whether the dag_delta was a full model delta or an incremental delta.

Definition at line 77 of file dag_delta.h.

10.67.5.8 gptr<model>* coco::dag_undelta::old_model

This variable holds the complete old model, in case the dag_delta made a full model delta (is_full_undelta == `true`)

Definition at line 50 of file dag_delta.h.

10.67.5.9 counted_ptr<model> coco::dag_undelta::rm_dag

This variable holds the subgraph induced by all nodes which have been removed by the dag_delta.

Definition at line 54 of file dag_delta.h.

10.67.5.10 std::vector<model::enhanced_edge> coco::dag_undelta::rm_e

This variable holds the edges which had connected the removed subgraph to the remaining DAG prior to its removal by the dag_delta.

Definition at line 57 of file dag_delta.h.

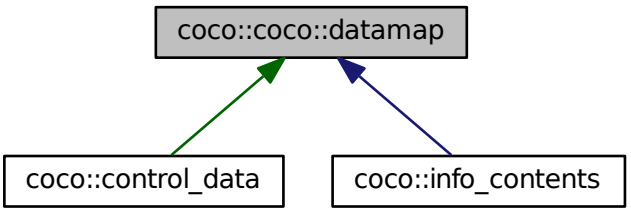
The documentation for this class was generated from the following files:

- dag_delta.h
- dag_delta.cc

10.68 coco::coco::datamap Class Reference

The base class for communicating with COCONUT modules.

Inheritance diagram for coco::coco::datamap:



Public Member Functions

- datamap ()
- datamap (const std::string &__n, const basic_alltype &__v)
- datamap (const char *__n, const basic_alltype &__v)

- datamap (const datamap &__c)
- virtual ~datamap ()
- void list (std::vector< std::string > &__v) const

- bool sinsert (const std::string &__s, const basic_alltype &__h, bool replace)
- bool sinsert (const char *__s, const basic_alltype &__h, bool replace)

- bool sinsert (const std::string &__s, int i, const basic_alltype &__h, bool replace)
- bool sinsert (const char *__s, int i, const basic_alltype &__h, bool replace)

- const basic_alltype & sfind (const std::string &__s) const
- const basic_alltype & sfind (const char *__s) const

- const basic_alltype & sfind (const std::string &__s, int i) const
- const basic_alltype & sfind (const char *__s, int i) const

- void remove (const std::string &__s)
- void remove (const char *__s)

- void remove (const std::string &__s, int i)
- void remove (const char *__s, int i)

- bool defd (const std::string &__s) const
- bool defd (const char *__s) const

- bool defd (const std::string &__s, int i) const
- bool defd (const char *__s, int i) const

- bool which (const std::string &__s, std::vector< int > &__idx) const
- bool which (const char *__s, std::vector< int > &__idx) const

- bool retrieve (const std::string &__s, bool &__v) const
- bool retrieve (const std::string &__s, int &__v) const
- bool retrieve (const std::string &__s, unsigned int &__v) const
- bool retrieve (const std::string &__s, double &__v) const
- bool retrieve (const std::string &__s, interval &__v) const
- bool retrieve (const std::string &__s, std::string &__is) const
- bool retrieve (const std::string &__s, num::Number &__v) const
- bool retrieve (const std::string &__s, const std::vector< bool > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< int > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< unsigned int > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< double > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< interval > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< std::string > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< num::Number > *&__v) const
- bool retrieve (const char *__s, bool &__v) const
- bool retrieve (const char *__s, int &__v) const
- bool retrieve (const char *__s, unsigned int &__v) const
- bool retrieve (const char *__s, double &__v) const
- bool retrieve (const char *__s, interval &__v) const
- bool retrieve (const char *__s, std::string &__is) const
- bool retrieve (const char *__s, num::Number &__is) const
- bool retrieve (const char *__s, const std::vector< bool > *&__v) const
- bool retrieve (const char *__s, const std::vector< unsigned int > *&__v) const
- bool retrieve (const char *__s, const std::vector< int > *&__v) const
- bool retrieve (const char *__s, const std::vector< double > *&__v) const
- bool retrieve (const char *__s, const std::vector< interval > *&__v) const
- bool retrieve (const char *__s, const std::vector< std::string > *&__v) const
- bool retrieve (const char *__s, const std::vector< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< int > *&__v) const

- bool retrieve (const char *__s, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< num::Number > *&__v) const

- bool retrieve (const std::string &__s, bool &__v, bool __def) const
- bool retrieve (const std::string &__s, int &__v, int __def) const
- bool retrieve (const std::string &__s, unsigned int &__v, unsigned int __def) const
- bool retrieve (const std::string &__s, double &__v, double __def) const
- bool retrieve (const std::string &__s, interval &__v, const interval &__def) const
- bool retrieve (const std::string &__s, std::string &__is, const std::string &__def) const
- bool retrieve (const std::string &__s, num::Number &__is, const num::Number &__def) const
- bool retrieve (const std::string &__s, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve (const std::string &__s, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve (const std::string &__s, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve (const std::string &__s, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve (const std::string &__s, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve (const std::string &__s, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve (const std::string &__s, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const

- bool retrieve (const std::string &__s, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const
- bool retrieve (const char *__s, bool &__v, bool __def) const
- bool retrieve (const char *__s, int &__v, int __def) const
- bool retrieve (const char *__s, unsigned int &__v, unsigned int __def) const
- bool retrieve (const char *__s, double &__v, double __def) const
- bool retrieve (const char *__s, interval &__v, const interval &__def) const
- bool retrieve (const char *__s, std::string &__v, const std::string &__def) const
- bool retrieve (const char *__s, num::Number &__v, const num::Number &__def) const
- bool retrieve (const char *__s, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve (const char *__s, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve (const char *__s, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve (const char *__s, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve (const char *__s, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve (const char *__s, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve (const char *__s, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const

- bool retrieve (const char *__s, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const

- bool retrieve_i (const std::string &__s, int i, bool &__v) const
- bool retrieve_i (const std::string &__s, int i, int &__v) const
- bool retrieve_i (const std::string &__s, int i, unsigned int &__v) const
- bool retrieve_i (const std::string &__s, int i, double &__v) const
- bool retrieve_i (const std::string &__s, int i, interval &__v) const
- bool retrieve_i (const std::string &__s, int i, std::string &__v) const
- bool retrieve_i (const std::string &__s, int i, num::Number &__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< bool > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< unsigned int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< std::string > *&__v) const

- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, bool &__v) const
- bool retrieve_i (const char *__s, int i, int &__v) const
- bool retrieve_i (const char *__s, int i, unsigned int &__v) const
- bool retrieve_i (const char *__s, int i, double &__v) const
- bool retrieve_i (const char *__s, int i, interval &__v) const
- bool retrieve_i (const char *__s, int i, std::string &__is) const
- bool retrieve_i (const char *__s, int i, num::Number &__is) const
- bool retrieve_i (const char *__s, int i, const std::vector< bool > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< unsigned int > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< num::Number > *&__v) const

- bool retrieve_i (const std::string &__s, int i, bool &__v, bool __def) const
- bool retrieve_i (const std::string &__s, int i, int &__v, int __def) const
- bool retrieve_i (const std::string &__s, int i, unsigned int &__v, unsigned int __def) const
- bool retrieve_i (const std::string &__s, int i, double &__v, double __def) const
- bool retrieve_i (const std::string &__s, int i, interval &__v, const interval &__def) const
- bool retrieve_i (const std::string &__s, int i, std::string &__is, const std::string &__def) const
- bool retrieve_i (const std::string &__s, int i, num::Number &__is, const num::Number &__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const

- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, bool &__v, bool __def) const
- bool retrieve_i (const char *__s, int i, int &__v, int __def) const
- bool retrieve_i (const char *__s, int i, unsigned int &__v, unsigned int __def) const
- bool retrieve_i (const char *__s, int i, double &__v, double __def) const
- bool retrieve_i (const char *__s, int i, interval &__v, const interval &__def) const
- bool retrieve_i (const char *__s, int i, std::string &__v, const std::string &__def) const
- bool retrieve_i (const char *__s, int i, num::Number &__v, const num::Number &__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const

- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const

10.68.1 Detailed Description

This class is the base class for all classes used for communicating with COCONUT modules. It is basically a map of strings (variable names) to basic_alltype values (the variable values).

10.68.2 Constructor & Destructor Documentation

10.68.2.1 coco::coco::datamap::datamap ( ) [inline]

Standard Constructor

Definition at line 66 of file search_graph.cc.

10.68.2.2 coco::coco::datamap::datamap (const std::string & __n, const basic_alltype & __v) [inline]

Constructor setting the variable __n to value __v.

Definition at line 68 of file search_graph.cc.

10.68.2.3 coco::coco::datamap::datamap (const char * __n, const basic_alltype & __v) [inline]

Constructor setting the variable __n to value __v.

Definition at line 72 of file search_graph.cc.

10.68.2.4 coco::coco::datamap::datamap (const datamap & __c) [inline]

Standard Copy Constructor

Definition at line 76 of file search_graph.cc.

10.68.2.5 virtual coco::coco::datamap::~datamap ( ) [inline, virtual]

Standard Destructor

Definition at line 78 of file search_graph.cc.

10.68.3 Member Function Documentation

10.68.3.1 bool coco::datamap::defd (const std::string & __s) const [inline]

This method returns whether the variable with name __s is defined.

Definition at line 109 of file datamap.hpp.

10.68.3.2 bool coco::datamap::defd (const char * __s) const [inline]

This method returns whether the variable with name __s is defined.

Definition at line 114 of file datamap.hpp.

10.68.3.3 bool coco::datamap::defd (const std::string & __s, int i) const [inline]

This method returns whether the variable with name __s and index is defined.

Definition at line 119 of file datamap.hpp.

10.68.3.4 bool coco::datamap::defd (const char * __s, int i) const [inline]

This method returns whether the variable with name __s and index is defined.

Definition at line 124 of file datamap.hpp.

10.68.3.5 void coco::datamap::list (std::vector< std::string > & __v) const

The list method sets __v to the list of entry keys stored in this datamap.

Reimplemented in coco::control_data.

Definition at line 101 of file datamap.cc.

10.68.3.6 void coco::datamap::remove (const std::string & __s) [inline]

The remove methods remove the variable with name __s from the datamap.

Definition at line 57 of file datamap.hpp.

10.68.3.7 void coco::datamap::remove (const char * __s) [inline]

The remove methods remove the variable with name __s from the datamap.

Definition at line 65 of file datamap.hpp.

10.68.3.8 void coco::datamap::remove (const std::string & __s, int i) [inline]

The remove methods remove the variable with name __s and index i from the datamap.

Definition at line 70 of file datamap.hpp.

10.68.3.9 void coco::datamap::remove (const char * __s, int i) [inline]

The remove methods remove the variable with name __s and index i from the datamap.

Definition at line 76 of file datamap.hpp.

10.68.3.10 bool coco::datamap::retrieve (const std::string & __s, bool & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 149 of file datamap.hpp.

10.68.3.11 bool coco::datamap::retrieve (const std::string & __s, int & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 169 of file datamap.hpp.

10.68.3.12 bool coco::datamap::retrieve (const std::string & __s, unsigned int & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 191 of file datamap.hpp.

10.68.3.13 bool coco::datamap::retrieve (const std::string & __s, double & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 213 of file datamap.hpp.

10.68.3.14 bool coco::datamap::retrieve (const std::string & __s, interval & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 235 of file datamap.hpp.

10.68.3.15 bool coco::datamap::retrieve (const std::string & __s, std::string & __is) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 257 of file datamap.hpp.

10.68.3.16 bool coco::datamap::retrieve (const std::string & __s, num::Number & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 279 of file datamap.hpp.

10.68.3.17 bool coco::datamap::retrieve (const std::string & __s, const std::vector< bool > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 301 of file datamap.hpp.

10.68.3.18 bool coco::datamap::retrieve (const std::string & __s, const std::vector< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 323 of file datamap.hpp.

10.68.3.19 bool coco::datamap::retrieve (const std::string & __s, const std::vector< unsigned int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 345 of file datamap.hpp.

10.68.3.20 bool coco::datamap::retrieve (const std::string & __s, const std::vector< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 367 of file datamap.hpp.

10.68.3.21 bool coco::datamap::retrieve (const std::string & __s, const std::vector< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 389 of file datamap.hpp.

10.68.3.22 bool coco::datamap::retrieve (const std::string & __s, const std::vector< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 411 of file datamap.hpp.

10.68.3.23 bool coco::datamap::retrieve (const std::string & __s, const std::vector< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 433 of file datamap.hpp.

10.68.3.24 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< bool > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 455 of file datamap.hpp.

10.68.3.25 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 477 of file datamap.hpp.

10.68.3.26 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< unsigned int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 499 of file datamap.hpp.

10.68.3.27 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 521 of file datamap.hpp.

10.68.3.28 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 543 of file datamap.hpp.

10.68.3.29 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 565 of file datamap.hpp.

10.68.3.30 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 587 of file datamap.hpp.

10.68.3.31 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 609 of file datamap.hpp.

10.68.3.32 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 631 of file datamap.hpp.

10.68.3.33 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 653 of file datamap.hpp.

10.68.3.34 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 675 of file datamap.hpp.

10.68.3.35 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 697 of file datamap.hpp.

10.68.3.36 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 719 of file datamap.hpp.

10.68.3.37 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 741 of file datamap.hpp.

10.68.3.38 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 763 of file datamap.hpp.

10.68.3.39 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 785 of file datamap.hpp.

10.68.3.40 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 807 of file datamap.hpp.

10.68.3.41 bool coco::datamap::retrieve (const char * __s, bool & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 829 of file datamap.hpp.

10.68.3.42 bool coco::datamap::retrieve (const char * __s, int & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 839 of file datamap.hpp.

10.68.3.43 bool coco::datamap::retrieve (const char * __s, unsigned int & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 849 of file datamap.hpp.

10.68.3.44 bool coco::datamap::retrieve (const char * __s, double & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 859 of file datamap.hpp.

10.68.3.45 bool coco::datamap::retrieve (const char * __s, interval & __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 869 of file datamap.hpp.

10.68.3.46 bool coco::datamap::retrieve (const char * __s, std::string & __is) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 879 of file datamap.hpp.

10.68.3.47 bool coco::datamap::retrieve (const char * __s, num::Number & __is) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 889 of file datamap.hpp.

10.68.3.48 bool coco::datamap::retrieve (const char * __s, const std::vector< bool > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 899 of file datamap.hpp.

10.68.3.49 bool coco::datamap::retrieve (const char * __s, const std::vector< unsigned int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 919 of file datamap.hpp.

10.68.3.50 bool coco::datamap::retrieve (const char * __s, const std::vector< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 909 of file datamap.hpp.

10.68.3.51 bool coco::datamap::retrieve (const char * __s, const std::vector< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 929 of file datamap.hpp.

10.68.3.52 bool coco::datamap::retrieve (const char * __s, const std::vector< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 939 of file datamap.hpp.

10.68.3.53 bool coco::datamap::retrieve (const char * __s, const std::vector< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 949 of file datamap.hpp.

10.68.3.54 bool coco::datamap::retrieve (const char * __s, const std::vector< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 959 of file datamap.hpp.

10.68.3.55 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< bool > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 969 of file datamap.hpp.

10.68.3.56 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< unsigned int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 989 of file datamap.hpp.

10.68.3.57 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 979 of file datamap.hpp.

10.68.3.58 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 999 of file datamap.hpp.

10.68.3.59 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1009 of file datamap.hpp.

10.68.3.60 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1019 of file datamap.hpp.

10.68.3.61 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1029 of file datamap.hpp.

10.68.3.62 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1039 of file datamap.hpp.

10.68.3.63 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1050 of file datamap.hpp.

10.68.3.64 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1061 of file datamap.hpp.

10.68.3.65 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1072 of file datamap.hpp.

10.68.3.66 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1083 of file datamap.hpp.

10.68.3.67 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1094 of file datamap.hpp.

10.68.3.68 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1105 of file datamap.hpp.

10.68.3.69 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1116 of file datamap.hpp.

10.68.3.70 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1127 of file datamap.hpp.

10.68.3.71 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1138 of file datamap.hpp.

10.68.3.72 bool coco::datamap::retrieve (const std::string & __s, bool & __v, bool __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 159 of file datamap.hpp.

10.68.3.73 bool coco::datamap::retrieve (const std::string & __s, int & __v, int __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 181 of file datamap.hpp.

10.68.3.74 bool coco::datamap::retrieve (const std::string & __s, unsigned int & __v, unsigned int __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 203 of file datamap.hpp.

10.68.3.75 bool coco::datamap::retrieve (const std::string & __s, double & __v, double __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 225 of file datamap.hpp.

10.68.3.76 bool coco::datamap::retrieve (const std::string & __s, interval & __v, const interval & __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 247 of file datamap.hpp.

10.68.3.77 bool coco::datamap::retrieve (const std::string & __s, std::string & __is, const std::string & __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 269 of file datamap.hpp.

10.68.3.78 bool coco::datamap::retrieve (const std::string & __s, num::Number & __is, const num::Number & __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 291 of file datamap.hpp.

10.68.3.79 bool coco::datamap::retrieve (const std::string & __s, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 313 of file datamap.hpp.

10.68.3.80 bool coco::datamap::retrieve (const std::string & __s, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 335 of file datamap.hpp.

10.68.3.81 bool coco::datamap::retrieve (const std::string & __s, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 357 of file datamap.hpp.

10.68.3.82 bool coco::datamap::retrieve (const std::string & __s, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 379 of file datamap.hpp.

10.68.3.83 bool coco::datamap::retrieve (const std::string & __s, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 401 of file datamap.hpp.

10.68.3.84 bool coco::datamap::retrieve (const std::string & __s, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 423 of file datamap.hpp.

10.68.3.85 bool coco::datamap::retrieve (const std::string & __s, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 445 of file datamap.hpp.

10.68.3.86 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 467 of file datamap.hpp.

10.68.3.87 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 489 of file datamap.hpp.

10.68.3.88 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 511 of file datamap.hpp.

10.68.3.89 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< double > *& __v*, const vmtl::sparse_vector< double > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 533 of file datamap.hpp.

10.68.3.90 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< interval > *& __v*, const vmtl::sparse_vector< interval > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 555 of file datamap.hpp.

10.68.3.91 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< std::string > *& __v*, const vmtl::sparse_vector< std::string > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 577 of file datamap.hpp.

10.68.3.92 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< num::Number > *& __v*, const vmtl::sparse_vector< num::Number > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 599 of file datamap.hpp.

10.68.3.93 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::dense_matrix< double > *& __v*, const vmtl::dense_matrix< double > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 621 of file datamap.hpp.

10.68.3.94 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 643 of file datamap.hpp.

10.68.3.95 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 665 of file datamap.hpp.

10.68.3.96 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 687 of file datamap.hpp.

10.68.3.97 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 709 of file datamap.hpp.

10.68.3.98 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 731 of file datamap.hpp.

10.68.3.99 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 753 of file datamap.hpp.

10.68.3.100 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 775 of file datamap.hpp.

10.68.3.101 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 797 of file datamap.hpp.

10.68.3.102 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 819 of file datamap.hpp.

10.68.3.103 bool coco::datamap::retrieve (const char * __s, bool & __v, bool __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 834 of file datamap.hpp.

10.68.3.104 bool coco::datamap::retrieve (const char * __s, int & __v, int __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 844 of file datamap.hpp.

10.68.3.105 bool coco::datamap::retrieve (const char * __s, unsigned int & __v, unsigned int __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 854 of file datamap.hpp.

10.68.3.106 bool coco::datamap::retrieve (const char * __s, double & __v, double __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 864 of file datamap.hpp.

10.68.3.107 bool coco::datamap::retrieve (const char * __s, interval & __v, const interval & __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 874 of file datamap.hpp.

10.68.3.108 bool coco::datamap::retrieve (const char * __s, std::string & __v, const std::string & __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 884 of file datamap.hpp.

10.68.3.109 bool coco::datamap::retrieve (const char * __s, num::Number & __v, const num::Number & __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 894 of file datamap.hpp.

10.68.3.110 bool coco::datamap::retrieve (const char * __s, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 904 of file datamap.hpp.

10.68.3.111 bool coco::datamap::retrieve (const char * __s, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 924 of file datamap.hpp.

10.68.3.112 bool coco::datamap::retrieve (const char * __s, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 914 of file datamap.hpp.

10.68.3.113 bool coco::datamap::retrieve (const char * __s, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 934 of file datamap.hpp.

10.68.3.114 bool coco::datamap::retrieve (const char * __s, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 944 of file datamap.hpp.

10.68.3.115 bool coco::datamap::retrieve (const char * __s, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 954 of file datamap.hpp.

10.68.3.116 bool coco::datamap::retrieve (const char * __s, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 964 of file datamap.hpp.

10.68.3.117 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 974 of file datamap.hpp.

10.68.3.118 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 994 of file datamap.hpp.

10.68.3.119 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 984 of file datamap.hpp.

10.68.3.120 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1004 of file datamap.hpp.

10.68.3.121 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1014 of file datamap.hpp.

10.68.3.122 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1024 of file datamap.hpp.

10.68.3.123 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1034 of file datamap.hpp.

10.68.3.124 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1044 of file datamap.hpp.

10.68.3.125 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1055 of file datamap.hpp.

10.68.3.126 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1066 of file datamap.hpp.

10.68.3.127 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1077 of file datamap.hpp.

10.68.3.128 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1088 of file datamap.hpp.

10.68.3.129 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1099 of file datamap.hpp.

10.68.3.130 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1110 of file datamap.hpp.

10.68.3.131 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1121 of file datamap.hpp.

10.68.3.132 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1132 of file datamap.hpp.

10.68.3.133 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1143 of file datamap.hpp.

10.68.3.134 bool coco::datamap::retrieve_i (const std::string & __s, int i, bool & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1150 of file datamap.hpp.

10.68.3.135 bool coco::datamap::retrieve_i (const std::string & __s, int i, int & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1170 of file datamap.hpp.

10.68.3.136 bool coco::datamap::retrieve_i (const std::string & __s, int i, unsigned int & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1190 of file datamap.hpp.

10.68.3.137 bool coco::datamap::retrieve_i (const std::string & __s, int i, double & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1210 of file datamap.hpp.

10.68.3.138 bool coco::datamap::retrieve_i (const std::string & __s, int i, interval & __v) const [inline]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1230 of file datamap.hpp.

10.68.3.139 bool coco::datamap::retrieve_i (const std::string & __s, int i, std::string & __v) const [inline]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1250 of file datamap.hpp.

10.68.3.140 bool coco::datamap::retrieve_i (const std::string & __s, int i, num::Number & __v) const [inline]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1270 of file datamap.hpp.

10.68.3.141 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< bool > *& __v) const [inline]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1290 of file datamap.hpp.

10.68.3.142 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< int > *& __v) const [inline]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1310 of file datamap.hpp.

10.68.3.143 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< unsigned int > *& __v) const [inline]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1330 of file datamap.hpp.

10.68.3.144 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1350 of file datamap.hpp.

10.68.3.145 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1370 of file datamap.hpp.

10.68.3.146 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1390 of file datamap.hpp.

10.68.3.147 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1410 of file datamap.hpp.

10.68.3.148 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< bool > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1430 of file datamap.hpp.

10.68.3.149 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1450 of file datamap.hpp.

10.68.3.150 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< unsigned int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1470 of file datamap.hpp.

10.68.3.151 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1490 of file datamap.hpp.

10.68.3.152 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1510 of file datamap.hpp.

10.68.3.153 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1530 of file datamap.hpp.

10.68.3.154 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1550 of file datamap.hpp.

10.68.3.155 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1570 of file datamap.hpp.

10.68.3.156 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1590 of file datamap.hpp.

10.68.3.157 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1610 of file datamap.hpp.

10.68.3.158 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1630 of file datamap.hpp.

10.68.3.159 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1650 of file datamap.hpp.

10.68.3.160 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1670 of file datamap.hpp.

10.68.3.161 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1690 of file datamap.hpp.

10.68.3.162 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1710 of file datamap.hpp.

10.68.3.163 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1730 of file datamap.hpp.

10.68.3.164 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1750 of file datamap.hpp.

10.68.3.165 bool coco::datamap::retrieve_i (const char * __s, int i, bool & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1770 of file datamap.hpp.

10.68.3.166 bool coco::datamap::retrieve_i (const char * __s, int i, int & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1780 of file datamap.hpp.

10.68.3.167 bool coco::datamap::retrieve_i (const char * __s, int i, unsigned int & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1790 of file datamap.hpp.

10.68.3.168 bool coco::datamap::retrieve_i (const char * __s, int i, double & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers

are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1800 of file datamap.hpp.

10.68.3.169 bool coco::datamap::retrieve_i (const char * __s, int i, interval & __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1810 of file datamap.hpp.

10.68.3.170 bool coco::datamap::retrieve_i (const char * __s, int i, std::string & __is) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1820 of file datamap.hpp.

10.68.3.171 bool coco::datamap::retrieve_i (const char * __s, int i, num::Number & __is) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1830 of file datamap.hpp.

10.68.3.172 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< bool > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1840 of file datamap.hpp.

10.68.3.173 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1850 of file datamap.hpp.

10.68.3.174 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< unsigned int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1860 of file datamap.hpp.

10.68.3.175 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1870 of file datamap.hpp.

10.68.3.176 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1880 of file datamap.hpp.

10.68.3.177 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1890 of file datamap.hpp.

10.68.3.178 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1900 of file datamap.hpp.

10.68.3.179 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< bool > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1910 of file datamap.hpp.

10.68.3.180 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1920 of file datamap.hpp.

10.68.3.181 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< unsigned int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1930 of file datamap.hpp.

10.68.3.182 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1940 of file datamap.hpp.

10.68.3.183 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1950 of file datamap.hpp.

10.68.3.184 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1960 of file datamap.hpp.

10.68.3.185 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1970 of file datamap.hpp.

10.68.3.186 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1980 of file datamap.hpp.

10.68.3.187 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1991 of file datamap.hpp.

10.68.3.188 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2002 of file datamap.hpp.

10.68.3.189 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2013 of file datamap.hpp.

10.68.3.190 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2024 of file datamap.hpp.

10.68.3.191 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< double > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2035 of file datamap.hpp.

10.68.3.192 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< int > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2046 of file datamap.hpp.

10.68.3.193 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< interval > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2057 of file datamap.hpp.

10.68.3.194 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< std::string > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2068 of file datamap.hpp.

10.68.3.195 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< num::Number > *& __v) const [inline]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2079 of file datamap.hpp.

10.68.3.196 bool coco::datamap::retrieve_i (const std::string & __s, int i, bool & __v, bool __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1161 of file datamap.hpp.

10.68.3.197 bool coco::datamap::retrieve_i (const std::string & __s, int i, int & __v, int __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1181 of file datamap.hpp.

10.68.3.198 bool coco::datamap::retrieve_i (const std::string & __s, int i, unsigned int & __v, unsigned int __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1201 of file datamap.hpp.

10.68.3.199 bool coco::datamap::retrieve_i (const std::string & __s, int i, double & __v, double __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1221 of file datamap.hpp.

10.68.3.200 bool coco::datamap::retrieve_i (const std::string & __s, int i, interval & __v, const interval & __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1241 of file datamap.hpp.

10.68.3.201 bool coco::datamap::retrieve_i (const std::string & __s, int i, std::string & __is, const std::string & __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1261 of file datamap.hpp.

10.68.3.202 bool coco::datamap::retrieve_i (const std::string & __s, int i, num::Number & __is, const num::Number & __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1281 of file datamap.hpp.

10.68.3.203 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1301 of file datamap.hpp.

10.68.3.204 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1321 of file datamap.hpp.

10.68.3.205 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const `[inline]`

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1341 of file datamap.hpp.

10.68.3.206 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< double > *& __v, const std::vector< double > * __def) const `[inline]`

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1361 of file datamap.hpp.

10.68.3.207 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< interval > *& __v, const std::vector< interval > * __def) const `[inline]`

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1381 of file datamap.hpp.

10.68.3.208 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const `[inline]`

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1401 of file datamap.hpp.

10.68.3.209 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const `[inline]`

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1421 of file datamap.hpp.

10.68.3.210 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_vector< bool > *__v*, const vmtl::sparse_vector< bool > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1441 of file datamap.hpp.

10.68.3.211 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_vector< int > *__v*, const vmtl::sparse_vector< int > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1461 of file datamap.hpp.

10.68.3.212 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_vector< unsigned int > *__v*, const vmtl::sparse_vector< unsigned int > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1481 of file datamap.hpp.

10.68.3.213 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_vector< double > *__v*, const vmtl::sparse_vector< double > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1501 of file datamap.hpp.

10.68.3.214 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_vector< interval > *__v*, const vmtl::sparse_vector< interval > * *__def*) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1521 of file datamap.hpp.

10.68.3.215 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1541 of file datamap.hpp.

10.68.3.216 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1561 of file datamap.hpp.

10.68.3.217 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1581 of file datamap.hpp.

10.68.3.218 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1601 of file datamap.hpp.

10.68.3.219 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1621 of file datamap.hpp.

10.68.3.220 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1641 of file datamap.hpp.

10.68.3.221 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1661 of file datamap.hpp.

10.68.3.222 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1681 of file datamap.hpp.

10.68.3.223 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1701 of file datamap.hpp.

10.68.3.224 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1721 of file datamap.hpp.

10.68.3.225 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1741 of file datamap.hpp.

10.68.3.226 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1761 of file datamap.hpp.

10.68.3.227 bool coco::datamap::retrieve_i (const char * __s, int i, bool & __v, bool __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1775 of file datamap.hpp.

10.68.3.228 bool coco::datamap::retrieve_i (const char * __s, int i, int & __v, int __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1785 of file datamap.hpp.

10.68.3.229 bool coco::datamap::retrieve_i (const char * __s, int i, unsigned int & __v, unsigned int __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1795 of file datamap.hpp.

10.68.3.230 bool coco::datamap::retrieve_i (const char * __s, int i, double & __v, double __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1805 of file datamap.hpp.

10.68.3.231 bool coco::datamap::retrieve_i (const char * __s, int i, interval & __v, const interval & __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1815 of file datamap.hpp.

10.68.3.232 bool coco::datamap::retrieve_i (const char * __s, int i, std::string & __v, const std::string & __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1825 of file datamap.hpp.

10.68.3.233 bool coco::datamap::retrieve_i (const char * __s, int i, num::Number & __v, const num::Number & __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1835 of file datamap.hpp.

10.68.3.234 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1845 of file datamap.hpp.

10.68.3.235 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1855 of file datamap.hpp.

10.68.3.236 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1865 of file datamap.hpp.

10.68.3.237 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1875 of file datamap.hpp.

10.68.3.238 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1885 of file datamap.hpp.

10.68.3.239 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1895 of file datamap.hpp.

10.68.3.240 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1905 of file datamap.hpp.

10.68.3.241 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1915 of file datamap.hpp.

10.68.3.242 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1925 of file datamap.hpp.

10.68.3.243 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1935 of file datamap.hpp.

10.68.3.244 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1945 of file datamap.hpp.

10.68.3.245 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1955 of file datamap.hpp.

10.68.3.246 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1965 of file datamap.hpp.

10.68.3.247 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1975 of file datamap.hpp.

10.68.3.248 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1985 of file datamap.hpp.

10.68.3.249 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1996 of file datamap.hpp.

10.68.3.250 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2007 of file datamap.hpp.

10.68.3.251 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2018 of file datamap.hpp.

10.68.3.252 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2029 of file datamap.hpp.

10.68.3.253 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2040 of file datamap.hpp.

10.68.3.254 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2051 of file datamap.hpp.

10.68.3.255 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2062 of file datamap.hpp.

10.68.3.256 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2073 of file datamap.hpp.

10.68.3.257 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2084 of file datamap.hpp.

10.68.3.258 const basic_alltype & coco::datamap::sfind (const std::string & __s) const [inline]

This method returns the value (as basic_alltype) of the variable __s. If the variable is not set, an empty basic_alltype is returned.

Definition at line 81 of file datamap.hpp.

10.68.3.259 const basic_alltype & coco::datamap::sfind (const char * __s) const [inline]

This method returns the value (as basic_alltype) of the variable __s. If the variable is not set, an empty basic_alltype is returned.

Definition at line 90 of file datamap.hpp.

10.68.3.260 const basic_alltype & coco::datamap::sfind (const std::string & __s, int i) const [inline]

This method returns the value (as basic_alltype) of the variable __s with index i. If the variable is not set, an empty basic_alltype is returned.

Definition at line 95 of file datamap.hpp.

10.68.3.261 const basic_alltype & coco::datamap::sfind (const char * __s, int i) const [inline]

This method returns the value (as basic_alltype) of the variable __s with index i. If the variable is not set, an empty basic_alltype is returned.

Definition at line 104 of file datamap.hpp.

10.68.3.262 bool coco::datamap::sinsert (const std::string & __s, const basic_alltype & __h, bool replace)

This method inserts the variable __s with value __h into the datamap. If replace is true, the value of an already defined variable __s would be replaced, if it is false the new value would not be set. The method returns whether the variable __s was indeed set to __h.

Definition at line 36 of file datamap.cc.

10.68.3.263 bool coco::datamap::sinsert (const char * __s, const basic_alltype & __h, bool replace) [inline]

This method inserts the variable __s with value __h into the datamap. If replace is true, the value of an already defined variable __s would be replaced, if it is false the new value would not be set. The method returns whether the variable __s was indeed set to __h.

Definition at line 36 of file datamap.hpp.

10.68.3.264 bool coco::datamap::sinsert (const std::string & __s, int i, const basic_alltype & __h, bool replace) [inline]

This method inserts the variable __s with index i with value __h into the datamap. If replace is true, the value of an already defined variable __s with index i would be replaced, if it is false the new value would not be set. The method returns whether the variable __s with index i was indeed set to __h.

Definition at line 42 of file datamap.hpp.

10.68.3.265 bool coco::datamap::sinsert (const char * __s, int i, const basic_alltype & __h, bool replace) [inline]

This method inserts the variable __s with index i with value __h into the datamap. If replace is true, the value of an already defined variable __s with index i would be replaced, if it is false the new value would not be set. The method returns whether the variable __s with index i was indeed set to __h.

Definition at line 50 of file datamap.hpp.

10.68.3.266 bool coco::datamap::which (const std::string & __s, std::vector< int > & __idx) const [inline]

The which methods sets __idx to the list of indices defined for the variable with name __s. The methods return whether the variable __s is defined.

Definition at line 129 of file datamap.hpp.

10.68.3.267 bool coco::datamap::which (const char * __s, std::vector< int > & __idx) const [inline]

The which methods sets __idx to the list of indices defined for the variable with name __s. The methods return whether the variable __s is defined.

Definition at line 143 of file datamap.hpp.

The documentation for this class was generated from the following files:

- datamap.h
- datamap.cc
- datamap.hpp

10.69 coco::datamap Class Reference

The base class for communicating with COCONUT modules.

`#include <datamap.h>`

Public Member Functions

- datamap ()
- datamap (const std::string &__n, const basic_alltype &__v)

- datamap (const char *__n, const basic_alltype &__v)
- datamap (const datamap &__c)
- virtual ~datamap ()
- void list (std::vector< std::string > &__v) const

- bool sinsert (const std::string &__s, const basic_alltype &__h, bool replace)
- bool sinsert (const char *__s, const basic_alltype &__h, bool replace)

- bool sinsert (const std::string &__s, int i, const basic_alltype &__h, bool replace)
- bool sinsert (const char *__s, int i, const basic_alltype &__h, bool replace)

- const basic_alltype & sfind (const std::string &__s) const
- const basic_alltype & sfind (const char *__s) const

- const basic_alltype & sfind (const std::string &__s, int i) const
- const basic_alltype & sfind (const char *__s, int i) const

- void remove (const std::string &__s)
- void remove (const char *__s)

- void remove (const std::string &__s, int i)
- void remove (const char *__s, int i)

- bool defd (const std::string &__s) const
- bool defd (const char *__s) const

- bool defd (const std::string &__s, int i) const
- bool defd (const char *__s, int i) const

- bool which (const std::string &__s, std::vector< int > &__idx) const
- bool which (const char *__s, std::vector< int > &__idx) const

- bool retrieve (const std::string &__s, bool &__v) const
- bool retrieve (const std::string &__s, int &__v) const
- bool retrieve (const std::string &__s, unsigned int &__v) const
- bool retrieve (const std::string &__s, double &__v) const
- bool retrieve (const std::string &__s, interval &__v) const
- bool retrieve (const std::string &__s, std::string &__is) const
- bool retrieve (const std::string &__s, num::Number &__v) const
- bool retrieve (const std::string &__s, const std::vector< bool > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< int > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< unsigned int > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< double > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< interval > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< std::string > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< num::Number > *&__v) const
- bool retrieve (const char *__s, bool &__v) const
- bool retrieve (const char *__s, int &__v) const
- bool retrieve (const char *__s, unsigned int &__v) const
- bool retrieve (const char *__s, double &__v) const
- bool retrieve (const char *__s, interval &__v) const
- bool retrieve (const char *__s, std::string &__is) const
- bool retrieve (const char *__s, num::Number &__is) const
- bool retrieve (const char *__s, const std::vector< bool > *&__v) const
- bool retrieve (const char *__s, const std::vector< unsigned int > *&__v) const
- bool retrieve (const char *__s, const std::vector< int > *&__v) const
- bool retrieve (const char *__s, const std::vector< double > *&__v) const
- bool retrieve (const char *__s, const std::vector< interval > *&__v) const
- bool retrieve (const char *__s, const std::vector< std::string > *&__v) const
- bool retrieve (const char *__s, const std::vector< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< int > *&__v) const

- bool retrieve (const char *__s, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< num::Number > *&__v) const

- bool retrieve (const std::string &__s, bool &__v, bool __def) const
- bool retrieve (const std::string &__s, int &__v, int __def) const
- bool retrieve (const std::string &__s, unsigned int &__v, unsigned int __def) const
- bool retrieve (const std::string &__s, double &__v, double __def) const
- bool retrieve (const std::string &__s, interval &__v, const interval &__def) const
- bool retrieve (const std::string &__s, std::string &__is, const std::string &__def) const
- bool retrieve (const std::string &__s, num::Number &__is, const num::Number &__def) const
- bool retrieve (const std::string &__s, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve (const std::string &__s, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve (const std::string &__s, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve (const std::string &__s, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve (const std::string &__s, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve (const std::string &__s, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve (const std::string &__s, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const

- bool retrieve (const std::string &__s, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const
- bool retrieve (const char *__s, bool &__v, bool __def) const
- bool retrieve (const char *__s, int &__v, int __def) const
- bool retrieve (const char *__s, unsigned int &__v, unsigned int __def) const
- bool retrieve (const char *__s, double &__v, double __def) const
- bool retrieve (const char *__s, interval &__v, const interval &__def) const
- bool retrieve (const char *__s, std::string &__v, const std::string &__def) const
- bool retrieve (const char *__s, num::Number &__v, const num::Number &__def) const
- bool retrieve (const char *__s, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve (const char *__s, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve (const char *__s, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve (const char *__s, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve (const char *__s, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve (const char *__s, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve (const char *__s, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const

- bool retrieve (const char *__s, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const

- bool retrieve_i (const std::string &__s, int i, bool &__v) const
- bool retrieve_i (const std::string &__s, int i, int &__v) const
- bool retrieve_i (const std::string &__s, int i, unsigned int &__v) const
- bool retrieve_i (const std::string &__s, int i, double &__v) const
- bool retrieve_i (const std::string &__s, int i, interval &__v) const
- bool retrieve_i (const std::string &__s, int i, std::string &__v) const
- bool retrieve_i (const std::string &__s, int i, num::Number &__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< bool > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< unsigned int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< std::string > *&__v) const

- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, bool &__v) const
- bool retrieve_i (const char *__s, int i, int &__v) const
- bool retrieve_i (const char *__s, int i, unsigned int &__v) const
- bool retrieve_i (const char *__s, int i, double &__v) const
- bool retrieve_i (const char *__s, int i, interval &__v) const
- bool retrieve_i (const char *__s, int i, std::string &__is) const
- bool retrieve_i (const char *__s, int i, num::Number &__is) const
- bool retrieve_i (const char *__s, int i, const std::vector< bool > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< unsigned int > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< num::Number > *&__v) const

- bool retrieve_i (const std::string &__s, int i, bool &__v, bool __def) const
- bool retrieve_i (const std::string &__s, int i, int &__v, int __def) const
- bool retrieve_i (const std::string &__s, int i, unsigned int &__v, unsigned int __def) const
- bool retrieve_i (const std::string &__s, int i, double &__v, double __def) const
- bool retrieve_i (const std::string &__s, int i, interval &__v, const interval &__def) const
- bool retrieve_i (const std::string &__s, int i, std::string &__is, const std::string &__def) const
- bool retrieve_i (const std::string &__s, int i, num::Number &__is, const num::Number &__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const

- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, bool &__v, bool __def) const
- bool retrieve_i (const char *__s, int i, int &__v, int __def) const
- bool retrieve_i (const char *__s, int i, unsigned int &__v, unsigned int __def) const
- bool retrieve_i (const char *__s, int i, double &__v, double __def) const
- bool retrieve_i (const char *__s, int i, interval &__v, const interval &__def) const
- bool retrieve_i (const char *__s, int i, std::string &__v, const std::string &__def) const
- bool retrieve_i (const char *__s, int i, num::Number &__v, const num::Number &__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const

- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const

10.69.1 Detailed Description

This class is the base class for all classes used for communicating with COCONUT modules. It is basically a map of strings (variable names) to basic_alltype values (the variable values).

10.69.2 Constructor & Destructor Documentation

10.69.2.1 coco::datamap::datamap ( ) [inline]

Standard Constructor

Definition at line 66 of file datamap.h.

10.69.2.2 coco::datamap::datamap (const std::string & __n, const basic_alltype & __v) [inline]

Constructor setting the variable __n to value __v.

Definition at line 68 of file datamap.h.

10.69.2.3 coco::datamap::datamap (const char * __n, const basic_alltype & __v) [inline]

Constructor setting the variable __n to value __v.

Definition at line 72 of file datamap.h.

10.69.2.4 coco::datamap::datamap (const datamap & __c) [inline]

Standard Copy Constructor

Definition at line 76 of file datamap.h.

10.69.2.5 virtual coco::datamap::~datamap ( ) [inline, virtual]

Standard Destructor

Definition at line 78 of file datamap.h.

10.69.3 Member Function Documentation

10.69.3.1 bool coco::datamap::defd (const std::string & __s) const

This method returns whether the variable with name __s is defined.

10.69.3.2 bool coco::datamap::defd (const char * __s) const

This method returns whether the variable with name __s is defined.

10.69.3.3 bool coco::datamap::defd (const std::string & __s, int i) const

This method returns whether the variable with name __s and index is defined.

10.69.3.4 bool coco::datamap::defd (const char * __s, int i) const

This method returns whether the variable with name __s and index is defined.

10.69.3.5 void coco::datamap::list (std::vector< std::string > & __v) const

The list method sets __v to the list of entry keys stored in this datamap.

10.69.3.6 void coco::datamap::remove (const std::string & __s)

The remove methods remove the variable with name __s from the datamap.

10.69.3.7 void coco::datamap::remove (const char * __s)

The remove methods remove the variable with name __s from the datamap.

10.69.3.8 void coco::datamap::remove (const std::string & __s, int i)

The remove methods remove the variable with name __s and index i from the datamap.

10.69.3.9 void coco::datamap::remove (const char * __s, int i)

The remove methods remove the variable with name __s and index i from the datamap.

10.69.3.10 bool coco::datamap::retrieve (const std::string & __s, bool & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.11 bool coco::datamap::retrieve (const std::string & __s, int & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.12 bool coco::datamap::retrieve (const std::string & __s, unsigned int & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.13 bool coco::datamap::retrieve (const std::string & __s, double & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.14 bool coco::datamap::retrieve (const std::string & __s, interval & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.15 bool coco::datamap::retrieve (const std::string & __s, std::string & __is) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.16 bool coco::datamap::retrieve (const std::string & __s, num::Number & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.17 bool coco::datamap::retrieve (const std::string & __s, const std::vector< bool > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.18 bool coco::datamap::retrieve (const std::string & __s, const std::vector< int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.19 bool coco::datamap::retrieve (const std::string & __s, const std::vector< unsigned int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.20 bool coco::datamap::retrieve (const std::string & __s, const std::vector< double > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.21 bool coco::datamap::retrieve (const std::string & __s, const std::vector< interval > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.22 bool coco::datamap::retrieve (const std::string & __s, const std::vector< std::string > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.23 bool coco::datamap::retrieve (const std::string & __s, const std::vector< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.24 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< bool > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.25 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.26 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< unsigned int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.27 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< double > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.28 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< interval > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.29 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< std::string > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.30 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.31 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< double > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.32 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.33 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< interval > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.34 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< std::string > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.35 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.36 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< double > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.37 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.38 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< interval > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.39 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< std::string > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.40 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.41 bool coco::datamap::retrieve (const char * __s, bool & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.42 bool coco::datamap::retrieve (const char * __s, int & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.43 bool coco::datamap::retrieve (const char * __s, unsigned int & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.44 bool coco::datamap::retrieve (const char * __s, double & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.45 bool coco::datamap::retrieve (const char * __s, interval & __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.46 bool coco::datamap::retrieve (const char * __s, std::string & __is) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.47 bool coco::datamap::retrieve (const char * __s, num::Number & __is) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.48 bool coco::datamap::retrieve (const char * __s, const std::vector< bool > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.49 bool coco::datamap::retrieve (const char * __s, const std::vector< unsigned int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.50 bool coco::datamap::retrieve (const char * __s, const std::vector< int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.51 bool coco::datamap::retrieve (const char * __s, const std::vector< double > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to

the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.52 bool coco::datamap::retrieve (const char * __s, const std::vector< interval > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.53 bool coco::datamap::retrieve (const char * __s, const std::vector< std::string > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.54 bool coco::datamap::retrieve (const char * __s, const std::vector< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.55 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< bool > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.56 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< unsigned int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.57 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.58 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< double > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.59 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< interval > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.60 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< std::string > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.61 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.62 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< double > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.63 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.64 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< interval > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.65 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< std::string > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.66 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.67 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< double > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.68 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< int > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.69 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< interval > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.70 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< std::string > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.71 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.72 bool coco::datamap::retrieve (const std::string & __s, bool & __v, bool __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.73 bool coco::datamap::retrieve (const std::string & __s, int & __v, int __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.74 bool coco::datamap::retrieve (const std::string & __s, unsigned int & __v, unsigned int __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.75 bool coco::datamap::retrieve (const std::string & __s, double & __v, double __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.76 bool coco::datamap::retrieve (const std::string & __s, interval & __v, const interval & __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.77 bool coco::datamap::retrieve (const std::string & __s, std::string & __is, const std::string & __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.78 bool coco::datamap::retrieve (const std::string & __s, num::Number & __is, const num::Number & __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.79 bool coco::datamap::retrieve (const std::string & __s, const std::vector< bool > *& __v, const std::vector< bool > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.80 bool coco::datamap::retrieve (const std::string & __s, const std::vector< int > *& __v, const std::vector< int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.81 bool coco::datamap::retrieve (const std::string & __s, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.82 bool coco::datamap::retrieve (const std::string & __s, const std::vector< double > *& __v, const std::vector< double > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.83 bool coco::datamap::retrieve (const std::string & __s, const std::vector< interval > *& __v, const std::vector< interval > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.84 bool coco::datamap::retrieve (const std::string & __s, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.85 bool coco::datamap::retrieve (const std::string & __s, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.86 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.87 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< int > *& *__v*, const vmtl::sparse_vector< int > * *__def*) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.88 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< unsigned int > *& *__v*, const vmtl::sparse_vector< unsigned int > * *__def*) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.89 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< double > *& *__v*, const vmtl::sparse_vector< double > * *__def*) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.90 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< interval > *& *__v*, const vmtl::sparse_vector< interval > * *__def*) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.91 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< std::string > *& *__v*, const vmtl::sparse_vector< std::string > * *__def*) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.92 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< num::Number > *& *__v*, const vmtl::sparse_vector< num::Number > * *__def*) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.93 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.94 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.95 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.96 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.97 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.98 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.99 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.100 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.101 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.102 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.103 bool coco::datamap::retrieve (const char * __s, bool & __v, bool __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.104 bool coco::datamap::retrieve (const char * __s, int & __v, int __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.105 bool coco::datamap::retrieve (const char * __s, unsigned int & __v, unsigned int __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.106 bool coco::datamap::retrieve (const char * __s, double & __v, double __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.107 bool coco::datamap::retrieve (const char * __s, interval & __v, const interval & __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.108 bool coco::datamap::retrieve (const char * __s, std::string & __v, const std::string & __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.109 bool coco::datamap::retrieve (const char * __s, num::Number & __v, const num::Number & __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.110 bool coco::datamap::retrieve (const char * __s, const std::vector< bool > *& __v, const std::vector< bool > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.111 bool coco::datamap::retrieve (const char * __s, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.112 bool coco::datamap::retrieve (const char * __s, const std::vector< int > *& __v, const std::vector< int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.113 bool coco::datamap::retrieve (const char * __s, const std::vector< double > *& __v, const std::vector< double > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.114 bool coco::datamap::retrieve (const char * __s, const std::vector< interval > *& __v, const std::vector< interval > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.115 bool coco::datamap::retrieve (const char * __s, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.116 bool coco::datamap::retrieve (const char * __s, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.117 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.118 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.119 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.120 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.121 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.122 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.123 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.124 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.125 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.126 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.127 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.128 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.129 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.130 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.131 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.132 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.133 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.134 bool coco::datamap::retrieve_i (const std::string & __s, int i, bool & __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.135 bool coco::datamap::retrieve_i (const std::string & __s, int i, int & __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.136 bool coco::datamap::retrieve_i (const std::string & __s, int i, unsigned int & __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.137 bool coco::datamap::retrieve_i (const std::string & __s, int i, double & __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.138 bool coco::datamap::retrieve_i (const std::string & __s, int i, interval & __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.139 bool coco::datamap::retrieve_i (const std::string & __s, int i, std::string & __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.140 bool coco::datamap::retrieve_i (const std::string & __s, int i, num::Number & __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.141 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< bool > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.142 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.143 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< unsigned int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.144 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< double > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.145 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< interval > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.146 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< std::string > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.147 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.148 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< bool > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.149 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.150 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< unsigned int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.151 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< double > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.152 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< interval > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.153 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< std::string > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.154 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.155 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< double > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.156 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.157 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< interval > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.158 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< std::string > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.159 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.160 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< double > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.161 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.162 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< interval > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.163 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< std::string > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.164 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< num::Number > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.165 bool coco::datamap::retrieve_i (const char * __s, int i, bool & __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.166 bool coco::datamap::retrieve_i (const char * __s, int i, int & __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.167 bool coco::datamap::retrieve_i (const char * __s, int i, unsigned int & __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.168 bool coco::datamap::retrieve_i (const char * __s, int i, double & __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.169 bool coco::datamap::retrieve_i (const char * __s, int i, interval & __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.170 bool coco::datamap::retrieve_i (const char * __s, int i, std::string & __is) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.171 bool coco::datamap::retrieve_i (const char * __s, int i, num::Number & __is) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.172 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< bool > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.173 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< int > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.174 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< unsigned int > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.175 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< double > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.176 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< interval > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.177 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< std::string > *& __v) const

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.178 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.179 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< bool > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.180 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.181 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< unsigned int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.182 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< double > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.183 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< interval > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.184 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< std::string > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.185 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.186 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< double > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.187 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.188 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< interval > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.189 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< std::string > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.190 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.191 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< double > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.192 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< int > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.193 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< interval > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.194 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< std::string > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.195 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< num::Number > *& __v) const

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.196 bool coco::datamap::retrieve_i (const std::string & __s, int i, bool & __v, bool __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.197 bool coco::datamap::retrieve_i (const std::string & __s, int i, int & __v, int __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.198 bool coco::datamap::retrieve_i (const std::string & __s, int i, unsigned int & __v, unsigned int __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.199 bool coco::datamap::retrieve_i (const std::string & __s, int i, double & __v, double __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.200 bool coco::datamap::retrieve_i (const std::string & __s, int i, interval & __v, const interval & __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.201 bool coco::datamap::retrieve_i (const std::string & __s, int i, std::string & __is, const std::string & __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.202 bool coco::datamap::retrieve_i (const std::string & __s, int i, num::Number & __is, const num::Number & __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.203 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< bool > *& __v, const std::vector< bool > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.204 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< int > *& __v, const std::vector< int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.205 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.206 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< double > *& __v, const std::vector< double > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.207 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< interval > *& __v, const std::vector< interval > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.208 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.209 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.210 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.211 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.212 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.213 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.214 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.215 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.216 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.217 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.218 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.219 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.220 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.221 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.222 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.223 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.224 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.225 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.226 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.227 bool coco::datamap::retrieve_i (const char * __s, int i, bool & __v, bool __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.228 bool coco::datamap::retrieve_i (const char * __s, int i, int & __v, int __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.229 bool coco::datamap::retrieve_i (const char * __s, int i, unsigned int & __v, unsigned int __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.230 bool coco::datamap::retrieve_i (const char * __s, int i, double & __v, double __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.231 bool coco::datamap::retrieve_i (const char * __s, int i, interval & __v, const interval & __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.232 bool coco::datamap::retrieve_i (const char * __s, int i, std::string & __v, const std::string & __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.233 bool coco::datamap::retrieve_i (const char * __s, int i, num::Number & __v, const num::Number & __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.234 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< bool > *& __v, const std::vector< bool > * __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.235 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< int > *& __v, const std::vector< int > * __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.236 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.237 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< double > *& __v, const std::vector< double > * __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.238 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< interval > *& __v, const std::vector< interval > * __def) const

The retrieve methods assign the value of the variable ___s with index i to * the variable ___v. It is returned whether the variable is set. If * the variable is not set, the value of ___def is assigned to ___v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.239 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.240 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.241 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.242 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.243 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.244 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.245 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.246 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.247 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.248 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.249 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.250 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.251 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.252 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.253 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.254 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.255 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.256 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.257 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

10.69.3.258 const basic_alltype& coco::datamap::sfind (const std::string & __s) const

This method returns the value (as basic_alltype) of the variable __s. If the variable is not set, an empty basic_alltype is returned.

10.69.3.259 const basic_alltype& coco::datamap::sfind (const char * __s) const

This method returns the value (as basic_alltype) of the variable __s. If the variable is not set, an empty basic_alltype is returned.

10.69.3.260 const basic_alltype& coco::datamap::sfind (const std::string & __s, int i) const

This method returns the value (as basic_alltype) of the variable __s with index i. If the variable is not set, an empty basic_alltype is returned.

10.69.3.261 const basic_alltype& coco::datamap::sfind (const char * __s, int i) const

This method returns the value (as basic_alltype) of the variable __s with index i. If the variable is not set, an empty basic_alltype is returned.

10.69.3.262 bool coco::datamap::sinsert (const std::string & __s, const basic_alltype & __h, bool replace)

This method inserts the variable __s with value __h into the datamap. If replace is true, the value of an already defined variable __s would be replaced, if it is false the new value would not be set. The method returns whether the variable __s was indeed set to __h.

10.69.3.263 bool coco::datamap::sinsert (const char * __s, const basic_alltype & __h, bool replace)

This method inserts the variable __s with value __h into the datamap. If replace is true, the value of an already defined variable __s would be replaced, if it is false the new value would not be set. The method returns whether the variable __s was indeed set to __h.

10.69.3.264 bool coco::datamap::sinsert (const std::string & __s, int i, const basic_alltype & __h, bool replace)

This method inserts the variable __s with index i with value __h into the datamap. If replace is true, the value of an already defined variable __s with index i would be replaced, if it is false the new value would not be set. The method returns whether the variable __s with index i was indeed set to __h.

10.69.3.265 bool coco::datamap::sinsert (const char * __s, int i, const basic_alltype & __h, bool replace)

This method inserts the variable __s with index i with value __h into the datamap. If replace is true, the value of an already defined variable __s with index i would be replaced, if it is false the new value would not be set. The method returns whether the variable __s with index i was indeed set to __h.

10.69.3.266 bool coco::datamap::which (const std::string & __s, std::vector< int > & __idx) const

The which methods sets __idx to the list of indices defined for the variable with name __s. The methods return whether the variable __s is defined.

10.69.3.267 bool coco::datamap::which (const char * __s, std::vector< int > & __idx) const

The which methods sets ___idx to the list of indices defined for the variable with name ___s. The methods return whether the variable ___s is defined.

The documentation for this class was generated from the following file:

- datamap.h

10.70 coco::dbccmp_false Struct Reference

`#include <dbcompare.hpp>`

Public Member Functions

- dbccmp_false ()
- ~dbccmp_false ()
- bool operator() (const _TR &x, const _TR &y)

10.70.1 Constructor & Destructor Documentation

10.70.1.1 coco::dbccmp_false::dbccmp_false ( ) [inline]

Definition at line 231 of file dbcompare.hpp.

10.70.1.2 coco::dbccmp_false::~dbccmp_false ( ) [inline]

Definition at line 232 of file dbcompare.hpp.

10.70.2 Member Function Documentation

10.70.2.1 bool coco::dbccmp_false::operator() (const _TR & x, const _TR & y) [inline]

Definition at line 233 of file dbcompare.hpp.

The documentation for this struct was generated from the following file:

- dbcompare.hpp

10.71 coco::dbccmp_true Struct Reference

`#include <dbcompare.hpp>`

Public Member Functions

- dbccmp_true ()
- ~dbccmp_true ()
- bool operator() (const _TR &x, const _TR &y)

10.71.1 Constructor & Destructor Documentation

10.71.1.1 coco::dbccmp_true::dbccmp_true ( ) [inline]

Definition at line 222 of file dbcompare.hpp.

10.71.1.2 coco::dbccmp_true::~dbccmp_true ( ) [inline]

Definition at line 223 of file dbcompare.hpp.

10.71.2 Member Function Documentation

10.71.2.1 bool coco::dbccmp_true::operator() (const _TR & *x*, const _TR & *y*) [inline]

Definition at line 224 of file dbcompare.hpp.

The documentation for this struct was generated from the following file:

- dbcompare.hpp

10.72 coco::dbccmps_absgt Class Reference

`#include <dbcompare.hpp>`

Public Member Functions

- dbccmps_absgt (double _rel=1.0, double _abs=0.0)
- ~dbccmps_absgt ()
- bool operator() (const _TR &x, const _TR &y)

10.72.1 Constructor & Destructor Documentation

10.72.1.1 coco::dbccmps_absgt::dbccmps_absgt (double *_rel* = `1.0`, double *_abs* = `0.0`) [inline]

Definition at line 144 of file dbcompare.hpp.

10.72.1.2 coco::dbccmps_absgt::~dbccmps_absgt ( ) [inline]

Definition at line 146 of file dbcompare.hpp.

10.72.2 Member Function Documentation

10.72.2.1 bool coco::dbccmps_absgt::operator() (const _TR & *x*, const _TR & *y*) [inline]

Definition at line 148 of file dbcompare.hpp.

The documentation for this class was generated from the following file:

- dbcompare.hpp

10.73 coco::dbccmps_abslt Class Reference

`#include <dbcompare.hpp>`

Public Member Functions

- dbccmps_abslt (double _rel=1.0, double _abs=0.0)
- ~dbccmps_abslt ()
- bool operator() (const _TR &x, const _TR &y)

10.73.1 Constructor & Destructor Documentation

10.73.1.1 coco::dbccmps_abslt::dbccmps_abslt (double *_rel* = `1.0`, double *_abs* = `0.0`) [inline]

Definition at line 175 of file dbcompare.hpp.

10.73.1.2 coco::dbccmps_abslt::~dbccmps_abslt ( ) [inline]

Definition at line 177 of file dbcompare.hpp.

10.73.2 Member Function Documentation

10.73.2.1 bool coco::dbccmps_abslt::operator() (const _TR & *x*, const _TR & *y*) [inline]

Definition at line 179 of file dbcompare.hpp.

The documentation for this class was generated from the following file:

- dbcompare.hpp

10.74 coco::dbccmps_false Class Reference

`#include <dbcompare.hpp>`

Public Member Functions

- dbccmps_false (double _rel=1.0, double _abs=0.0)
- ~dbccmps_false ()
- bool operator() (const _TR &x, const _TR &y)

10.74.1 Constructor & Destructor Documentation

10.74.1.1 coco::dbccmps_false::dbccmps_false (double *_rel* = `1.0`, double *_abs* = `0.0`) [inline]

Definition at line 208 of file dbcompare.hpp.

10.74.1.2 coco::dbccmps_false::~dbccmps_false ( ) [inline]

Definition at line 210 of file dbcompare.hpp.

10.74.2 Member Function Documentation

10.74.2.1 bool coco::dbccmps_false::operator() (const _TR & *x*, const _TR & *y*) [inline]

Definition at line 212 of file dbcompare.hpp.

The documentation for this class was generated from the following file:

- dbcompare.hpp

10.75 coco::dbccmps_gt Class Reference

```
#include <dbcompare.hpp>
```

Public Member Functions

- dbccmps_gt (double _rel=1.0, double _abs=0.0)
- ~dbccmps_gt ()
- bool operator() (const _TR &x, const _TR &y)

10.75.1 Constructor & Destructor Documentation

10.75.1.1 coco::dbccmps_gt::dbccmps_gt (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 129 of file dbcompare.hpp.

10.75.1.2 coco::dbccmps_gt::~dbccmps_gt ( ) [inline]

Definition at line 131 of file dbcompare.hpp.

10.75.2 Member Function Documentation

10.75.2.1 bool coco::dbccmps_gt::operator() (const _TR & *x*, const _TR & *y*) [inline]

Definition at line 133 of file dbcompare.hpp.

The documentation for this class was generated from the following file:

- dbcompare.hpp

10.76 coco::dbccmps_lt Class Reference

```
#include <dbcompare.hpp>
```

Public Member Functions

- dbccmps_lt (double _rel=1.0, double _abs=0.0)
- ~dbccmps_lt ()
- bool operator() (const _TR &x, const _TR &y)

10.76.1 Constructor & Destructor Documentation

10.76.1.1 coco::dbccmps_lt::dbccmps_lt (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 160 of file dbcompare.hpp.

10.76.1.2 coco::dbccmps_lt::~dbccmps_lt ( ) [inline]

Definition at line 162 of file dbcompare.hpp.

10.76.2 Member Function Documentation

10.76.2.1 bool coco::dbccmps_lt::operator() (const _TR & *x*, const _TR & *y*) [inline]

Definition at line 164 of file dbcompare.hpp.

The documentation for this class was generated from the following file:

- dbcompare.hpp

10.77 coco::dbccmps_true Class Reference

`#include <dbcompare.hpp>`

Public Member Functions

- dbccmps_true (double _rel=1.0, double _abs=0.0)
- ~dbccmps_true ()
- bool operator() (const _TR &x, const _TR &y)

10.77.1 Constructor & Destructor Documentation

10.77.1.1 coco::dbccmps_true::dbccmps_true (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 191 of file dbcompare.hpp.

10.77.1.2 coco::dbccmps_true::~dbccmps_true ( ) [inline]

Definition at line 193 of file dbcompare.hpp.

10.77.2 Member Function Documentation

10.77.2.1 bool coco::dbccmps_true::operator() (const _TR & *x*, const _TR & *y*) [inline]

Definition at line 195 of file dbcompare.hpp.

The documentation for this class was generated from the following file:

- dbcompare.hpp

10.78 coco::coco::dbt_row Class Reference

This type is used to hold one row in some table of the search database.

Public Member Functions

- dbt_row ()
- template<class _TC >
  dbt_row (const std::string &nm, const _TC &cont)
- template<class _TC >
  dbt_row (const char *nm, const _TC &cont)
- dbt_row (const dbt_row &dr)
- ~dbt_row ()
- const _Base & row () const
- _Base & row ()
- template<class _TC >
  void add (const std::string &nm, const _TC &cont)
- template<class _TC >
  void add (const char *nm, const _TC &cont)
- dbt_row ()
- template<class _TC >
  dbt_row (const std::string &nm, const _TC &cont)
- template<class _TC >
  dbt_row (const char *nm, const _TC &cont)
- dbt_row (const dbt_row &dr)
- ~dbt_row ()
- const _Base & row () const
- _Base & row ()
- template<class _TC >
  void add (const std::string &nm, const _TC &cont)
- template<class _TC >
  void add (const char *nm, const _TC &cont)

10.78.1 Detailed Description

Variables of this type are used to assemble a row as a collection of column values for communication with the search database.

Definition at line 43 of file search_graph.cc.

10.78.2 Constructor & Destructor Documentation

10.78.2.1 coco::coco::dbt_row::dbt_row ( ) [inline]

Standard Constructor

Definition at line 52 of file search_graph.cc.

10.78.2.2 template<class _TC > coco::coco::dbt_row::dbt_row (const std::string & *nm*, const _TC & *cont*) [inline]

Constructor initializing the dbt_row with one column of name `nm` containing `cont`.

Definition at line 56 of file search_graph.cc.

10.78.2.3 template<class _TC > coco::coco::dbt_row::dbt_row (const char * *nm*, const _TC & *cont*) [inline]

Constructor initializing the dbt_row with one column of name `nm` containing `cont`.

Definition at line 60 of file search_graph.cc.

10.78.2.4 coco::coco::dbt_row::dbt_row (const dbt_row & *dr*) [inline]

Standard Copy Constructor

Definition at line 63 of file search_graph.cc.

10.78.2.5 coco::coco::dbt_row::~dbt_row ( ) [inline]

Standard Destructor

Definition at line 66 of file search_graph.cc.

10.78.2.6 coco::coco::dbt_row::dbt_row ( ) [inline]

Standard Constructor

Definition at line 52 of file search_graph.cc.

10.78.2.7 template<class _TC > coco::coco::dbt_row::dbt_row (const std::string & *nm*, const _TC & *cont*) [inline]

Constructor initializing the dbt_row with one column of name `nm` containing `cont`.

Definition at line 56 of file search_graph.cc.

10.78.2.8 template<class _TC > coco::coco::dbt_row::dbt_row (const char * *nm*, const _TC & *cont*) [inline]

Constructor initializing the dbt_row with one column of name `nm` containing `cont`.

Definition at line 60 of file search_graph.cc.

10.78.2.9 coco::coco::dbt_row::dbt_row (const dbt_row & *dr*) [inline]

Standard Copy Constructor

Definition at line 63 of file search_graph.cc.

10.78.2.10 coco::coco::dbt_row::~dbt_row ( ) [inline]

Standard Destructor

Definition at line 66 of file search_graph.cc.

10.78.3 Member Function Documentation

10.78.3.1 template<class _TC > void coco::coco::dbt_row::add (const std::string & *nm,* const _TC & *cont*) [inline]

This method adds a new column with name `nm` containing `cont`.

Definition at line 83 of file dbbasic.h.

10.78.3.2 template<class _TC > void coco::coco::dbt_row::add (const std::string & *nm,* const _TC & *cont*)

This method adds a new column with name `nm` containing `cont`.

10.78.3.3 template<class _TC > void coco::coco::dbt_row::add (const char * *nm,* const _TC & *cont*) [inline]

This method adds a new column with name `nm` containing `cont`.

Definition at line 89 of file dbbasic.h.

10.78.3.4 template<class _TC > void coco::coco::dbt_row::add (const char * *nm,* const _TC & *cont*)

This method adds a new column with name `nm` containing `cont`.

10.78.3.5 const _Base& coco::coco::dbt_row::row ( ) const [inline]

Return the row contents of the dbt_row.

Definition at line 69 of file search_graph.cc.

10.78.3.6 const _Base& coco::coco::dbt_row::row ( ) const [inline]

Return the row contents of the dbt_row.

Definition at line 69 of file search_graph.cc.

10.78.3.7 _Base& coco::coco::dbt_row::row ( ) [inline]

Return the row contents of the dbt_row.

Definition at line 72 of file search_graph.cc.

10.78.3.8 _Base& coco::coco::dbt_row::row ( ) [inline]

Return the row contents of the dbt_row.

Definition at line 72 of file search_graph.cc.

The documentation for this class was generated from the following file:

- dbbasic.h

10.79 coco::dbt_row Class Reference

This type is used to hold one row in some table of the search database.

`#include <dbbasic.h>`

Public Member Functions

- dbt_row ()
- template<class _TC >
  dbt_row (const std::string &nm, const _TC &cont)
- template<class _TC >
  dbt_row (const char *nm, const _TC &cont)
- dbt_row (const dbt_row &dr)
- ~dbt_row ()
- const _Base & row () const
- _Base & row ()
- template<class _TC >
  void add (const std::string &nm, const _TC &cont)
- template<class _TC >
  void add (const char *nm, const _TC &cont)

10.79.1 Detailed Description

Variables of this type are used to assemble a row as a collection of column values for communication with the search database.

10.79.2 Constructor & Destructor Documentation

10.79.2.1 coco::dbt_row::dbt_row ( ) [inline]

Standard Constructor

Definition at line 51 of file dbbasic.h.

10.79.2.2 template<class _TC > coco::dbt_row::dbt_row (const std::string & *nm*, const _TC & *cont*) [inline]

Constructor initializing the dbt_row with one column of name `nm` containing `cont`.

Definition at line 55 of file dbbasic.h.

10.79.2.3 template<class _TC > coco::dbt_row::dbt_row (const char * *nm*, const _TC & *cont*) [inline]

Constructor initializing the dbt_row with one column of name `nm` containing `cont`.

Definition at line 59 of file dbbasic.h.

10.79.2.4 coco::dbt_row::dbt_row (const dbt_row & *dr*) [inline]

Standard Copy Constructor

Definition at line 62 of file dbbasic.h.

10.79.2.5 coco::dbt_row::~dbt_row ( ) [inline]

Standard Destructor

Definition at line 65 of file dbbasic.h.

10.79.3 Member Function Documentation

10.79.3.1 template<class _TC > void coco::dbt_row::add (const std::string & *nm*, const _TC & *cont*)

This method adds a new column with name `nm` containing `cont`.

10.79.3.2 template<class _TC > void coco::dbt_row::add (const char * *nm*, const _TC & *cont*)

This method adds a new column with name `nm` containing `cont`.

10.79.3.3 const _Base& coco::dbt_row::row ( ) const [inline]

Return the row contents of the dbt_row.

Definition at line 68 of file dbbasic.h.

10.79.3.4 _Base& coco::dbt_row::row ( ) [inline]

Return the row contents of the dbt_row.

Definition at line 71 of file dbbasic.h.

The documentation for this class was generated from the following file:

- dbbasic.h

10.80 coco::dd1f_interval_eval Class Reference

Forward function range evaluation.

`#include <dagd1func_ie.h>`

Inheritance diagram for coco::dd1f_interval_eval:

Collaboration diagram for coco::dd1f_interval_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- dd1f_interval_eval (const std::vector< interval > &__x, const variable_indicator &__v, const model &__m, std::vector< interval > *__c, bool do_i=true, const std::map< unsigned int, d1func * > *_pi=NULL, unsigned int __tn=0)
- dd1f_interval_eval (const dd1f_interval_eval &__v)
- ~dd1f_interval_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_box (const std::vector< interval > &__x, const variable_indicator &__v)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const interval &__rval)
- int update (const expression_node &__data, const interval &__rval)
- interval calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.80.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural interval extension.

10.80.2 Member Typedef Documentation

10.80.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.80.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.80.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.80.3 Constructor & Destructor Documentation

10.80.3.1 coco::dd1f_interval_eval::dd1f_interval_eval (const std::vector< interval > & __x, const variable_indicator & __v, const model & __m, std::vector< interval > * __c, bool do_i = true, const std::map< unsigned int, d1func * > * _pi = NULL, unsigned int __tn = 0) [inline]

Constructor: ___x is the box over which the function range shall be evaluated, ___v is a variable indicator specifying the variables that have changed value since the last evaluation, ___m specifies the DAG, and ___c the cache (may be NULL). The boolean do_i determines whether the natural interval extension will use known node ranges for reducing the ranges.

Definition at line 143 of file dagd1func_ie.h.

10.80.3.2 coco::dd1f_interval_eval::dd1f_interval_eval (const dd1f_interval_eval & __v) [inline]

Standard Copy Constructor

Definition at line 159 of file dagd1func_ie.h.

10.80.3.3 coco::dd1f_interval_eval::~dd1f_interval_eval ( ) [inline]

Standard Destructor

Definition at line 162 of file dagd1func_ie.h.

10.80.4 Member Function Documentation

10.80.4.1 void coco::dd1f_interval_eval::calculate (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 254 of file dagd1func_ie.h.

10.80.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.80.4.3 interval coco::dd1f_interval_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< dd1f_interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 767 of file dagd1func_ie.h.

10.80.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.80.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.80.4.6 void coco::dd1f_interval_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< dd1f_interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 179 of file dagd1func_ie.h.

10.80.4.7 int coco::dd1f_interval_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 181 of file dagd1func_ie.h.

10.80.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.80.4.9 bool coco::dd1f_interval_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the range for this node is already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< dd1f_interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 88 of file dagd1func_ie.h.

10.80.4.10 void coco::dd1f_interval_eval::new_box (const std::vector< interval > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation box to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 171 of file dagd1func_ie.h.

10.80.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.80.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.80.4.13 void coco::dd1f_interval_eval::retrieve_from_cache (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 265 of file dagd1func_ie.h.

10.80.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.80.4.15 expression_const_walker coco::dd1f_interval_eval::short_cut_to (const expression_node & *__data*) [inline]

NOP version, not needed

Definition at line 165 of file dagd1func_ie.h.

10.80.4.16 int coco::dd1f_interval_eval::update (const interval & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 280 of file dagd1func_ie.h.

10.80.4.17 int coco::dd1f_interval_eval::update (const expression_node & *__data,* const interval & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 286 of file dagd1func_ie.h.

10.80.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data,* const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.80.4.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.80.4.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.80.4.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.80.4.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.80.4.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

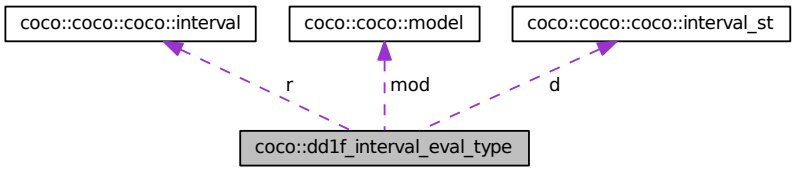
- dagd1func_ie.h

10.81 coco::dd1f_interval_eval_type Struct Reference

Visitor data for dd1f_interval_eval.

`#include <dagd1func_ie.h>`

Collaboration diagram for coco::dd1f_interval_eval_type:



Public Attributes

- const std::vector< interval > * x
- std::vector< interval > * cache
- const model * mod
- const std::map< unsigned int, d1func * > * p_infos
- unsigned int tn
- union {
  void * p
  interval_st d
  int info
  } u
- interval r
- unsigned int n
- bool do_intersect

10.81.1 Detailed Description

This class is the visitor data type for the dd1f_interval_eval evaluator.

10.81.2 Member Data Documentation

10.81.2.1 std::vector<interval>* coco::dd1f_interval_eval_type::cache

the function range cache

Definition at line 59 of file dagd1func_ie.h.

10.81.2.2 interval_st coco::dd1f_interval_eval_type::d

Definition at line 63 of file dagd1func_ie.h.

10.81.2.3 bool coco::dd1f_interval_eval_type::do_intersect

compute an intersection with the node range?

Definition at line 67 of file dagd1func_ie.h.

10.81.2.4 int coco::dd1f_interval_eval_type::info

Definition at line 63 of file dagd1func_ie.h.

10.81.2.5 const model* coco::dd1f_interval_eval_type::mod

the DAG

Definition at line 60 of file dagd1func_ie.h.

10.81.2.6 unsigned int coco::dd1f_interval_eval_type::n

children counter

Definition at line 66 of file dagd1func_ie.h.

10.81.2.7 void* coco::dd1f_interval_eval_type::p

Definition at line 63 of file dagd1func_ie.h.

10.81.2.8 const std::map<unsigned int,d1func*>* coco::dd1f_interval_eval_type::p_infos

the short-cut infos

Definition at line 61 of file dagd1func_ie.h.

10.81.2.9 interval coco::dd1f_interval_eval_type::r

return value

Definition at line 65 of file dagd1func_ie.h.

10.81.2.10 unsigned int coco::dd1f_interval_eval_type::tn

the top node number

Definition at line 62 of file dagd1func_ie.h.

10.81.2.11 union { ... } coco::dd1f_interval_eval_type::u

additional data for complex nodes

10.81.2.12 const std::vector<interval>* coco::dd1f_interval_eval_type::x

the box

Definition at line 58 of file dagd1func_ie.h.

The documentation for this struct was generated from the following file:

- dagd1func_ie.h

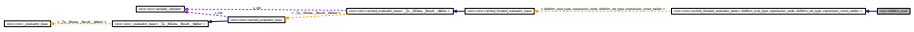
10.82 coco::defdom_eval Class Reference

Forward function range evaluation.

`#include <defdom_evaluator.h>`

Inheritance diagram for coco::defdom_eval:

Collaboration diagram for coco::defdom_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- defdom_eval (const std::vector< interval > &__x, const variable_indicator &__v, const model &_-_m, defdom_map &__p, std::vector< interval > *__c)
- defdom_eval (const defdom_eval &__v)
- ~defdom_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_box (const std::vector< interval > &__x, const variable_indicator &__v)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const defdom_ret_type &__rval)
- int update (const expression_node &__data, const defdom_ret_type &__rval)
- defdom_ret_type calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.82.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural interval extension.

10.82.2 Member Typedef Documentation

10.82.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.82.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.82.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.82.3 Constructor & Destructor Documentation

10.82.3.1 coco::defdom_eval::defdom_eval (const std::vector< interval > & __x, const variable_indicator & __v, const model & __m, defdom_map & __p, std::vector< interval > * __c) [inline]

Constructor: __x is the box over which the function range shall be evaluated, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies the DAG, and __c

the cache (may be NULL). The boolean do_i determines whether the natural interval extension will use known node ranges for reducing the ranges.

Definition at line 216 of file defdom_evaluator.h.

10.82.3.2 coco::defdom_eval::defdom_eval (const defdom_eval & __v) [inline]

Standard Copy Constructor

Definition at line 230 of file defdom_evaluator.h.

10.82.3.3 coco::defdom_eval::~defdom_eval ( ) [inline]

Standard Destructor

Definition at line 233 of file defdom_evaluator.h.

10.82.4 Member Function Documentation

10.82.4.1 void coco::defdom_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 321 of file defdom_evaluator.h.

10.82.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.82.4.3 defdom_ret_type coco::defdom_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< defdom_eval_type, expression_node, defdom_ret_type, expression_const_walker >.

Definition at line 973 of file defdom_evaluator.h.

10.82.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.82.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.82.4.6 void coco::defdom_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< defdom_eval_type, expression_node, defdom_ret_type, expression_const_walker >.

Definition at line 250 of file defdom_evaluator.h.

10.82.4.7 int coco::defdom_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 252 of file defdom_evaluator.h.

10.82.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.82.4.9 bool coco::defdom_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the range for this node is already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< defdom_eval_type, expression_node, defdom_ret_type, expression_const_walker >.

Definition at line 117 of file defdom_evaluator.h.

10.82.4.10 void coco::defdom_eval::new_box (const std::vector< interval > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation box to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 242 of file defdom_evaluator.h.

10.82.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.82.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.82.4.13 void coco::defdom_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 332 of file defdom_evaluator.h.

10.82.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.82.4.15 expression_const_walker coco::defdom_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 236 of file defdom_evaluator.h.

10.82.4.16 int coco::defdom_eval::update (const defdom_ret_type & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 352 of file defdom_evaluator.h.

10.82.4.17 int coco::defdom_eval::update (const expression_node & __data, const defdom_ret_type & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 358 of file defdom_evaluator.h.

10.82.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.82.4.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.82.4.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.82.4.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.82.4.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.82.4.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

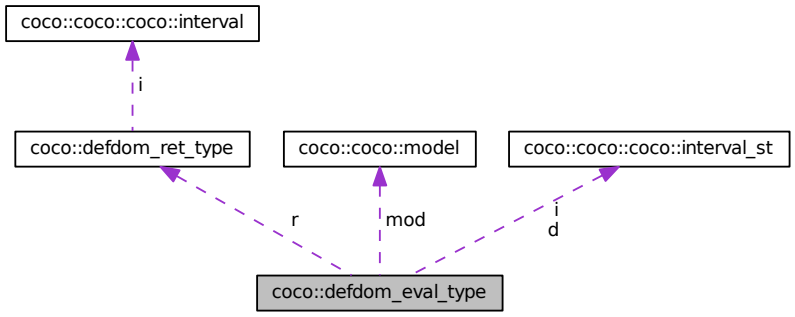
- defdom_evaluator.h

10.83 coco::defdom_eval_type Struct Reference

Visitor data for defdom_eval.

`#include <defdom_evaluator.h>`

Collaboration diagram for coco::defdom_eval_type:



Public Attributes

- const std::vector< interval > * x
- std::vector< interval > * cache
- const model * mod
- union {
  void * p
  interval_st d
  int info
  struct {
  int info
  interval_st i
  unsigned int n
  } ifhelp

} u

- defdom_ret_type r
- defdom_map ∗ prob_nodes

10.83.1 Detailed Description

This class is the visitor data type for the defdom_eval evaluator.

10.83.2 Member Data Documentation

10.83.2.1 std::vector<interval>∗ coco::defdom_eval_type::cache

the function range cache

Definition at line 89 of file defdom_evaluator.h.

10.83.2.2 interval_st coco::defdom_eval_type::d

Definition at line 91 of file defdom_evaluator.h.

10.83.2.3 interval_st coco::defdom_eval_type::i

Definition at line 92 of file defdom_evaluator.h.

10.83.2.4 struct { ... } coco::defdom_eval_type::ifhelp

10.83.2.5 int coco::defdom_eval_type::info

Definition at line 91 of file defdom_evaluator.h.

10.83.2.6 const model∗ coco::defdom_eval_type::mod

the DAG

Definition at line 90 of file defdom_evaluator.h.

10.83.2.7 unsigned int coco::defdom_eval_type::n

children counter

Definition at line 92 of file defdom_evaluator.h.

10.83.2.8 void∗ coco::defdom_eval_type::p

Definition at line 91 of file defdom_evaluator.h.

10.83.2.9 defdom_map∗ coco::defdom_eval_type::prob_nodes

Definition at line 96 of file defdom_evaluator.h.

10.83.2.10 defdom_ret_type coco::defdom_eval_type::r

return value

Definition at line 94 of file defdom_evaluator.h.

10.83.2.11 union { ... } coco::defdom_eval_type::u

additional data for complex nodes

10.83.2.12 const std::vector<interval>* coco::defdom_eval_type::x

the box

Definition at line 88 of file defdom_evaluator.h.

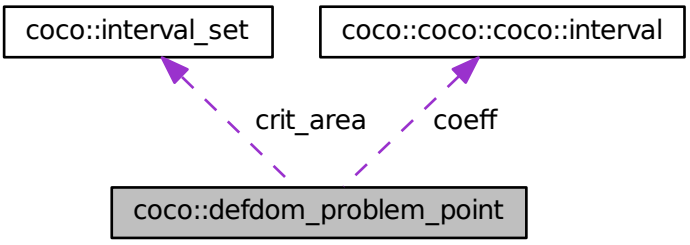
The documentation for this struct was generated from the following file:

- defdom_evaluator.h

10.84 coco::defdom_problem_point Struct Reference

`#include <defdom_evaluator.h>`

Collaboration diagram for coco::defdom_problem_point:



Public Member Functions

- defdom_problem_point (unsigned int n1, const interval_set &ca, int o, bool ieq=false, unsigned int n2=0, const interval &cf=0.)

Public Attributes

- bool is_equality

- unsigned int node1
- unsigned int node2
- interval_set crit_area
- int order
- interval coeff

10.84.1 Constructor & Destructor Documentation

10.84.1.1 coco::defdom_problem_point::defdom_problem_point (unsigned int *n1*, const interval_set & *ca*, int *o*, bool *ieq* = `false`, unsigned int *n2* = `0`, const interval & *cf* = `0.`) [inline]

Definition at line 69 of file defdom_evaluator.h.

10.84.2 Member Data Documentation

10.84.2.1 interval coco::defdom_problem_point::coeff

Definition at line 67 of file defdom_evaluator.h.

10.84.2.2 interval_set coco::defdom_problem_point::crit_area

Definition at line 65 of file defdom_evaluator.h.

10.84.2.3 bool coco::defdom_problem_point::is_equality

Definition at line 62 of file defdom_evaluator.h.

10.84.2.4 unsigned int coco::defdom_problem_point::node1

Definition at line 63 of file defdom_evaluator.h.

10.84.2.5 unsigned int coco::defdom_problem_point::node2

Definition at line 64 of file defdom_evaluator.h.

10.84.2.6 int coco::defdom_problem_point::order

Definition at line 66 of file defdom_evaluator.h.

The documentation for this struct was generated from the following file:

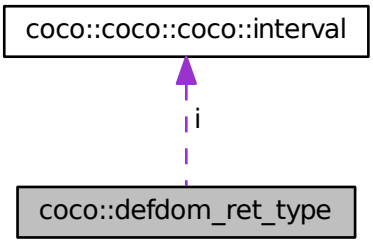
- defdom_evaluator.h

10.85 coco::defdom_ret_type Struct Reference

Return type for defdom_eval.

```
#include <defdom_evaluator.h>
```

Collaboration diagram for coco::defdom_ret_type:



Public Attributes

- unsigned int n
- bool t
- interval i

10.85.1 Detailed Description

This class is the return data type for the defdom_eval evaluator.

10.85.2 Member Data Documentation

10.85.2.1 interval coco::defdom_ret_type::i

Definition at line 57 of file defdom_evaluator.h.

10.85.2.2 unsigned int coco::defdom_ret_type::n

Definition at line 55 of file defdom_evaluator.h.

10.85.2.3 bool coco::defdom_ret_type::t

Definition at line 56 of file defdom_evaluator.h.

The documentation for this struct was generated from the following file:

- defdom_evaluator.h

10.86 coco::delta Class Reference

The delta class (updates to work nodes)

`#include <api_deltabase.h>`

Public Member Functions

- delta ()
- delta (const delta_base &__d)
- delta (const delta &__d)
- ~delta ()
- const std::string & get_action () const
- const delta_base * get_base () const
- bool apply (work_node &_x, const delta_id &_d) const
- bool apply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- void convert (work_node &_x)
- void unkeep ()
- delta_id store (work_node &_x, const certificate &_c)
- delta & operator= (const delta &_d)
- bool operator== (const delta &_c) const
- bool operator!= (const delta &_c) const

Friends

- class delta_base
- class ie_return_type
- std::ostream & operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.

10.86.1 Detailed Description

The delta class is used to store deltas (updates to search_nodes in the search_graph). It is designed as a wrapper class in order to overcome the problem that copy constructors cannot be overloaded. Deltas are stored in the search database table `deltas`.

10.86.2 Constructor & Destructor Documentation

10.86.2.1 coco::delta::delta ( ) [inline]

Standard Constructor

Definition at line 72 of file api_deltabase.h.

10.86.2.2 coco::delta::delta (const delta_base & __d)

Constructor, which constructs a delta from a delta_base. The clone operation for the delta_base will be called in that process.

10.86.2.3 coco::delta::delta (const delta & _d)

Copy Constructor, which constructs a new delta from an existing delta. The clone operation for the delta_base will be called in that process, effectively overloading the copy constructor for the delta_base.

10.86.2.4 coco::delta::~delta ( )

Destructor, which frees the _d using `delete`

10.86.3 Member Function Documentation

10.86.3.1 bool coco::delta::apply (work_node & _x, const delta_id & _d) const

Apply the delta with delta_id _d to work node _x

10.86.3.2 bool coco::delta::apply3 (work_node & _x, const work_node & _y, const delta_id & _d) const

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x.

10.86.3.3 void coco::delta::convert (work_node & _x)

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

10.86.3.4 const std::string& coco::delta::get_action ( ) const

Retrieve the action information (the delta type) for this delta.

10.86.3.5 const delta_base* coco::delta::get_base ( ) const

Return the delta_base stored in this wrapper

10.86.3.6 bool coco::delta::operator!= (const delta & _c) const

10.86.3.7 delta& coco::delta::operator= (const delta & _d)

Assignment operator, which uses the clone operation for the delta_base, effectively overloading the assignment operator for the delta_base.

10.86.3.8 bool coco::delta::operator== (const delta & _c) const

Comparison operators

10.86.3.9 delta_id coco::delta::store (work_node & _x, const certificate & _c)

Store the delta along with certificate _c in work_node _x. This work_node then **keeps** the delta. The created delta_id of the delta is returned.

10.86.3.10 void coco::delta::unkeep ( )

This method is called when the delta is permanently applied and it signals the delta that certain data structures should be kept intact, even if the delta is destroyed.

10.86.4 Friends And Related Function Documentation

10.86.4.1 friend class delta_base [friend]

Definition at line 127 of file api_deltabase.h.

10.86.4.2 friend class ie_return_type [friend]

Definition at line 128 of file api_deltabase.h.

10.86.4.3 std::ostream& operator<< (std::ostream & *o*, const delta & *t*) [friend]

Stream output operator for deltas, which calls the get_action method of the delta.

Definition at line 137 of file api_deltabase.h.

The documentation for this class was generated from the following file:

- api_deltabase.h

10.87 coco::coco::delta Class Reference

The delta class (updates to work nodes)

Public Member Functions

- delta ()
- delta (const delta_base &__d)
- delta (const delta &__d)
- ~delta ()
- const std::string & get_action () const
- const delta_base * get_base () const
- bool apply (work_node &_x, const delta_id &_d) const
- bool apply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- void convert (work_node &_x)
- void unkeep ()
- delta_id store (work_node &_x, const certificate &_c)
- delta & operator= (const delta &_d)
- bool operator== (const delta &_c) const
- bool operator!= (const delta &_c) const
- delta ()
- delta (const delta_base &__d)
- delta (const delta &__d)
- ~delta ()

- const std::string & get_action () const
- const delta_base * get_base () const
- bool apply (work_node &_x, const delta_id &_d) const
- bool apply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- void convert (work_node &_x)
- void unkeep ()
- delta_id store (work_node &_x, const certificate &_c)
- delta & operator= (const delta &_d)
- bool operator== (const delta &_c) const
- bool operator!= (const delta &_c) const
- delta ()
- delta (const delta_base &__d)
- delta (const delta &__d)
- ~delta ()
- const std::string & get_action () const
- const delta_base * get_base () const
- bool apply (work_node &_x, const delta_id &_d) const
- bool apply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- void convert (work_node &_x)
- void unkeep ()
- delta_id store (work_node &_x, const certificate &_c)
- delta & operator= (const delta &_d)
- bool operator== (const delta &_c) const
- bool operator!= (const delta &_c) const

Friends

- class delta_base
- class ie_return_type
- std::ostream & operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.
- std::ostream & operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.
- std::ostream & operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.

10.87.1 Detailed Description

The delta class is used to store deltas (updates to search_nodes in the search_graph). It is designed as a wrapper class in order to overcome the problem that copy constructors cannot be overloaded. Deltas are stored in the search database table `deltas`.

Definition at line 64 of file search_graph.cc.

10.87.2 Constructor & Destructor Documentation

10.87.2.1 coco::coco::delta::delta ( ) [inline]

Standard Constructor

Definition at line 72 of file search_graph.cc.

10.87.2.2 coco::coco::delta::delta (const delta_base & __d) [inline]

Constructor, which constructs a delta from a delta_base. The clone operation for the delta_base will be called in that process.

Definition at line 112 of file api_delta.h.

10.87.2.3 coco::coco::delta::delta (const delta & __d) [inline]

Copy Constructor, which constructs a new delta from an existing delta. The clone operation for the delta_base will be called in that process, effectively overloading the copy constructor for the delta_base.

Definition at line 114 of file api_delta.h.

10.87.2.4 coco::coco::delta::~delta ( ) [inline]

Destructor, which frees the _d using `delete`

Definition at line 120 of file api_delta.h.

10.87.2.5 coco::coco::delta::delta ( ) [inline]

Standard Constructor

Definition at line 72 of file search_graph.cc.

10.87.2.6 coco::coco::delta::delta (const delta_base & __d)

Constructor, which constructs a delta from a delta_base. The clone operation for the delta_base will be called in that process.

10.87.2.7 coco::coco::delta::delta (const delta & __d)

Copy Constructor, which constructs a new delta from an existing delta. The clone operation for the delta_base will be called in that process, effectively overloading the copy constructor for the delta_base.

10.87.2.8 coco::coco::delta::~delta ( )

Destructor, which frees the _d using `delete`

10.87.2.9 coco::coco::delta::delta ( ) [inline]

Standard Constructor

Definition at line 72 of file search_graph.cc.

10.87.2.10 coco::coco::delta::delta (const delta_base & __d)

Constructor, which constructs a delta from a delta_base. The clone operation for the delta_base will be called in that process.

10.87.2.11 coco::coco::delta::delta (const delta & __d)

Copy Constructor, which constructs a new delta from an existing delta. The clone operation for the delta_base will be called in that process, effectively overloading the copy constructor for the delta_base.

10.87.2.12 coco::coco::delta::~delta ( )

Destructor, which frees the _d using `delete`

10.87.3 Member Function Documentation

10.87.3.1 bool coco::coco::delta::apply (work_node & _x, const delta_id & _d) const [inline]

Apply the delta with delta_id _d to work node _x

Definition at line 139 of file api_delta.h.

10.87.3.2 bool coco::coco::delta::apply (work_node & _x, const delta_id & _d) const

Apply the delta with delta_id _d to work node _x

10.87.3.3 bool coco::coco::delta::apply (work_node & _x, const delta_id & _d) const

Apply the delta with delta_id _d to work node _x

10.87.3.4 bool coco::coco::delta::apply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [inline]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x.

Definition at line 160 of file api_delta.h.

10.87.3.5 bool coco::coco::delta::apply3 (work_node & _x, const work_node & _y, const delta_id & _d) const

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x.

10.87.3.6 bool coco::coco::delta::apply3 (work_node & _x, const work_node & _y, const delta_id & _d) const

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x.

10.87.3.7 void coco::coco::delta::convert (work_node & _x) [inline]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 182 of file api_delta.h.

10.87.3.8 void coco::coco::delta::convert (work_node & _x)

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

10.87.3.9 void coco::coco::delta::convert (work_node & _x)

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

10.87.3.10 const std::string& coco::coco::delta::get_action ( ) const

Retrieve the action information (the delta type) for this delta.

10.87.3.11 const std::string& coco::coco::delta::get_action ( ) const

Retrieve the action information (the delta type) for this delta.

10.87.3.12 const std::string & coco::coco::delta::get_action ( ) const [inline]

Retrieve the action information (the delta type) for this delta.

Definition at line 136 of file api_delta.h.

10.87.3.13 const delta_base * coco::coco::delta::get_base ( ) const [inline]

Return the delta_base stored in this wrapper

Definition at line 137 of file api_delta.h.

10.87.3.14 const delta_base* coco::coco::delta::get_base ( ) const

Return the delta_base stored in this wrapper

10.87.3.15 const delta_base* coco::coco::delta::get_base ( ) const

Return the delta_base stored in this wrapper

10.87.3.16 bool coco::coco::delta::operator!= (const delta & _c) const [inline]

Definition at line 195 of file api_delta.h.

10.87.3.17 bool coco::coco::delta::operator!= (const delta & _c) const

10.87.3.18 bool coco::coco::delta::operator!= (const delta & _c) const

10.87.3.19 delta & coco::coco::delta::operator= (const delta & _d) [inline]

Assignment operator, which uses the clone operation for the delta_base, effectively overloading the assignment operator for the delta_base.

Definition at line 126 of file api_delta.h.

10.87.3.20 delta& coco::coco::delta::operator= (const delta & _d)

Assignment operator, which uses the clone operation for the delta_base, effectively overloading the assignment operator for the delta_base.

10.87.3.21 delta& coco::coco::delta::operator= (const delta & _d)

Assignment operator, which uses the clone operation for the delta_base, effectively overloading the assignment operator for the delta_base.

10.87.3.22 bool coco::coco::delta::operator== (const delta & _c) const [inline]

Comparison operators

Definition at line 192 of file api_delta.h.

10.87.3.23 bool coco::coco::delta::operator== (const delta & _c) const

Comparison operators

10.87.3.24 bool coco::coco::delta::operator== (const delta & _c) const

Comparison operators

10.87.3.25 delta_id coco::coco::delta::store (work_node & _x, const certificate & _c)

Store the delta along with certificate _c in work_node _x. This work_node then **keeps** the delta. The created delta_id of the delta is returned.

10.87.3.26 delta_id coco::coco::delta::store (work_node & _x, const certificate & _c)

Store the delta along with certificate _c in work_node _x. This work_node then **keeps** the delta. The created delta_id of the delta is returned.

10.87.3.27 delta_id coco::delta::store (work_node & _x, const certificate & _c)

Store the delta along with certificate _c in work_node _x. This work_node then **keeps** the delta. The created delta_id of the delta is returned.

Definition at line 37 of file delta.cc.

10.87.3.28 void coco::coco::delta::unkeep ( ) [inline]

This method is called when the delta is permanently applied and it signals the delta that certain data structures should be kept intact, even if the delta is destroyed.

Definition at line 190 of file api_delta.h.

10.87.3.29 void coco::coco::delta::unkeep ( )

This method is called when the delta is permanently applied and it signals the delta that certain data structures should be kept intact, even if the delta is destroyed.

10.87.3.30 void coco::coco::delta::unkeep ( )

This method is called when the delta is permanently applied and it signals the delta that certain data structures should be kept intact, even if the delta is destroyed.

10.87.4 Friends And Related Function Documentation

10.87.4.1 delta_base [friend]

Definition at line 127 of file search_graph.cc.

10.87.4.2 ie_return_type [friend]

Definition at line 128 of file search_graph.cc.

10.87.4.3 std::ostream& operator<< (std::ostream & *o*, const delta & *t*) [friend]

Stream output operator for deltas, which calls the get_action method of the delta.

Definition at line 137 of file search_graph.cc.

10.87.4.4 std::ostream& operator<< (std::ostream & *o*, const delta & *t*) [friend]

Stream output operator for deltas, which calls the get_action method of the delta.

Definition at line 137 of file search_graph.cc.

10.87.4.5 std::ostream& operator<< (std::ostream & *o*, const delta & *t*) [friend]

Stream output operator for deltas, which calls the get_action method of the delta.

Definition at line 137 of file search_graph.cc.

The documentation for this class was generated from the following files:

- api_deltabase.h
- api_delta.h
- delta.cc

10.88 coco::delta_base Class Reference

Base class for the deltas.

`#include <api_deltabase.h>`

Public Member Functions

- delta_base ()
- delta_base (const std::string &a)
- delta_base (const char *a)
- delta_base (const delta_base &__d)
- virtual delta_base * new_copy () const PURE_VIRTUALvirtual void destroy_copy(delta_base *__d) const PURE_VIRTUALvirtual~delta_base()
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)

- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Protected Attributes

- std::string _action

10.88.1 Detailed Description

Base class for the deltas, which is wrapped by the delta class since copy constructors cannot directly be overloaded.

10.88.2 Constructor & Destructor Documentation

10.88.2.1 coco::delta_base::delta_base ( ) [inline]

Standard Constructor

Definition at line 158 of file api_deltabase.h.

10.88.2.2 coco::delta_base::delta_base (const std::string & *a*) [inline]

Constructor, explicitly setting the action to `a`

Definition at line 160 of file api_deltabase.h.

10.88.2.3 coco::delta_base::delta_base (const char * *a*) [inline]

Constructor, explicitly setting the action to `a`

Definition at line 162 of file api_deltabase.h.

10.88.2.4 coco::delta_base::delta_base (const delta_base & *__d*) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 164 of file api_deltabase.h.

10.88.3 Member Function Documentation

10.88.3.1 virtual bool coco::delta_base::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_di*, size_t & *delta_size*) const [inline, virtual]

Apply the delta with delta_id `_d` to work node `_x`. In this process the undo information for this delta is stored in `_u`.

Definition at line 198 of file api_deltabase.h.

10.88.3.2 virtual bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [virtual]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.88.3.3 virtual void coco::delta_base::convert (work_node & *_x*, delta_base *& *_d*) [inline, virtual]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file api_deltabase.h.

10.88.3.4 const std::string& coco::delta_base::get_action ( ) const [inline]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.88.3.5 delta coco::delta_base::make_delta (const std::string & *a*) [inline]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file api_deltabase.h.

10.88.3.6 virtual delta_base* coco::delta_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Definition at line 167 of file api_deltabase.h.

10.88.3.7 virtual void coco::delta_base::unkeep ( ) [inline, virtual]

Perform this operation when the delta is released (unkept) from a work_node.

Definition at line 194 of file api_deltabase.h.

10.88.4 Member Data Documentation

10.88.4.1 std::string coco::delta_base::_action [protected]

The action (type) of this delta

Definition at line 154 of file api_deltabase.h.

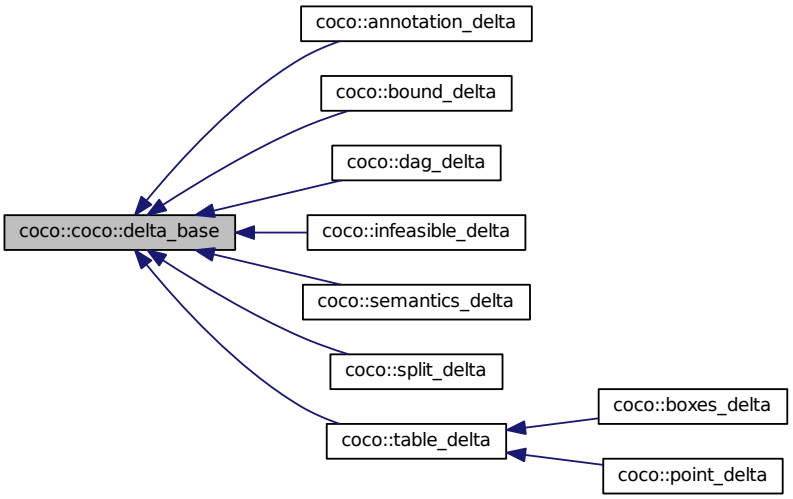
The documentation for this class was generated from the following file:

- api_deltabase.h

10.89 coco::coco::delta_base Class Reference

Base class for the deltas.

Inheritance diagram for coco::coco::delta_base:



Public Member Functions

- delta_base ()
- delta_base (const std::string &a)
- delta_base (const char ∗a)
- delta_base (const delta_base &__d)
- virtual delta_base ∗ new_copy () const PURE_VIRTUALvirtual void destroy_copy(delta_base ∗__d) const PURE_VIRTUALvirtual∼delta_base()
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base ∗&_d)
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base ∗&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base ∗&_u, const delta_id &_d, size_t &delta_size) const
- delta_base ()
- delta_base (const std::string &a)
- delta_base (const char ∗a)
- delta_base (const delta_base &__d)
- virtual delta_base ∗ new_copy () const PURE_VIRTUALvirtual void destroy_copy(delta_base ∗__d) const PURE_VIRTUALvirtual∼delta_base()
- delta make_delta (const std::string &a)

- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- delta_base ()
- delta_base (const std::string &a)
- delta_base (const char *a)
- delta_base (const delta_base &__d)
- virtual delta_base * new_copy () const PURE_VIRTUALvirtual void destroy_copy(delta_base *__d) const PURE_VIRTUALvirtual~delta_base()
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Protected Attributes

- std::string _action

10.89.1 Detailed Description

Base class for the deltas, which is wrapped by the delta class since copy constructors cannot directly be overloaded.

Definition at line 150 of file search_graph.cc.

10.89.2 Constructor & Destructor Documentation

10.89.2.1 coco::coco::delta_base::delta_base ( ) [inline]

Standard Constructor

Definition at line 158 of file search_graph.cc.

10.89.2.2 coco::coco::delta_base::delta_base (const std::string & *a*) [inline]

Constructor, explicitly setting the action to `a`

Definition at line 160 of file search_graph.cc.

10.89.2.3 coco::coco::delta_base::delta_base (const char * *a*) [inline]

Constructor, explicitly setting the action to `a`

Definition at line 162 of file search_graph.cc.

10.89.2.4 coco::coco::delta_base::delta_base (const delta_base & __d) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 164 of file search_graph.cc.

10.89.2.5 coco::coco::delta_base::delta_base ( ) [inline]

Standard Constructor

Definition at line 158 of file search_graph.cc.

10.89.2.6 coco::coco::delta_base::delta_base (const std::string & *a*) [inline]

Constructor, explicitly setting the action to `a`

Definition at line 160 of file search_graph.cc.

10.89.2.7 coco::coco::delta_base::delta_base (const char * *a*) [inline]

Constructor, explicitly setting the action to `a`

Definition at line 162 of file search_graph.cc.

10.89.2.8 coco::coco::delta_base::delta_base (const delta_base & __d) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 164 of file search_graph.cc.

10.89.2.9 coco::coco::delta_base::delta_base ( ) [inline]

Standard Constructor

Definition at line 158 of file search_graph.cc.

10.89.2.10 coco::coco::delta_base::delta_base (const std::string & *a*) [inline]

Constructor, explicitly setting the action to `a`

Definition at line 160 of file search_graph.cc.

10.89.2.11 coco::coco::delta_base::delta_base (const char * *a*) [inline]

Constructor, explicitly setting the action to `a`

Definition at line 162 of file search_graph.cc.

10.89.2.12 coco::coco::delta_base::delta_base (const delta_base & __d) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 164 of file search_graph.cc.

10.89.3 Member Function Documentation

10.89.3.1 virtual bool coco::coco::delta_base::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_di*, size_t & *delta_size*) const [inline, virtual]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.89.3.2 virtual bool coco::coco::delta_base::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_di*, size_t & *delta_size*) const [inline, virtual]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.89.3.3 virtual bool coco::coco::delta_base::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_di*, size_t & *delta_size*) const [inline, virtual]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.89.3.4 virtual bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [virtual]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.89.3.5 bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [inline, virtual]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.89.3.6 virtual bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [virtual]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.89.3.7 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.89.3.8 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.89.3.9 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.89.3.10 const std::string& coco::coco::delta_base::get_action ( ) const [inline]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.89.3.11 const std::string& coco::coco::delta_base::get_action ( ) const [inline]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.89.3.12 const std::string& coco::coco::delta_base::get_action ( ) const [inline]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.89.3.13 delta coco::coco::delta_base::make_delta (const std::string & a) [inline]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.89.3.14 delta coco::coco::delta_base::make_delta (const std::string & a) [inline]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.89.3.15 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.89.3.16 virtual delta_base* coco::coco::delta_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::semantics_delta, coco::dag_delta, coco::split_delta, coco::annotation_delta, coco::bound_delta, coco::table_delta, coco::infeasible_delta, coco::boxes_delta, and coco::point_delta.

Definition at line 167 of file search_graph.cc.

10.89.3.17 virtual delta_base* coco::coco::delta_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::semantics_delta, coco::dag_delta, coco::split_delta, coco::annotation_delta, coco::bound_delta, coco::table_delta, coco::infeasible_delta, coco::boxes_delta, and coco::point_delta.

Definition at line 167 of file search_graph.cc.

10.89.3.18 virtual delta_base* coco::coco::delta_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::semantics_delta, coco::dag_delta, coco::split_delta, coco::annotation_delta, coco::bound_delta, coco::table_delta, coco::infeasible_delta, coco::boxes_delta, and coco::point_delta.

Definition at line 167 of file search_graph.cc.

10.89.3.19 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.89.3.20 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.89.3.21 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.89.4 Member Data Documentation

10.89.4.1 std::string coco::coco::delta_base::_action [protected]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

The documentation for this class was generated from the following files:

- api_deltabase.h
- api_delta.h

10.90 coco::coco::delta_get_action Class Reference

Stored procedure class for computing the delta action specifier.

Public Types

- typedef vdbl::context context
- typedef std::string return_type

Public Member Functions

- delta_get_action (vdbl::colid _c)
- delta_get_action (const delta_get_action &_i)
- virtual ~delta_get_action ()
- const std::string & operator() () const
- const std::string & def () const
- void setcontext (const context *c, const vdbl::row *r)

10.90.1 Detailed Description

A database method returning the delta action specifier, used as a stored procedure.

10.90.2 Member Typedef Documentation

10.90.2.1 typedef vdbl::context coco::coco::delta_get_action::context

The evaluation context class

Definition at line 66 of file search_graph.cc.

10.90.2.2 typedef std::string coco::coco::delta_get_action::return_type

The return type of this stored procedure

Definition at line 68 of file search_graph.cc.

10.90.3 Constructor & Destructor Documentation

10.90.3.1 coco::coco::delta_get_action::delta_get_action (vdbl::colid _c) [inline]

Constructor, which sets the column id from _c

Definition at line 71 of file search_graph.cc.

10.90.3.2 coco::coco::delta_get_action::delta_get_action (const delta_get_action & _i) [inline]

Standard Copy Constructor

Definition at line 73 of file search_graph.cc.

10.90.3.3 virtual coco::coco::delta_get_action::~delta_get_action ( ) [inline, virtual]

Standard Destructor

Definition at line 76 of file search_graph.cc.

10.90.4 Member Function Documentation

10.90.4.1 const std::string& coco::coco::delta_get_action::def ( ) const [inline]

Default value of the procedure

Definition at line 81 of file search_graph.cc.

10.90.4.2 const std::string & coco::coco::delta_get_action::operator() ( ) const [inline]

Result of the procedure

Definition at line 98 of file api_delta.h.

10.90.4.3 void coco::coco::delta_get_action::setcontext (const context * c, const vdbl::row * r) [inline]

Set the evaluation context for row r and context c (ignored!)

Definition at line 83 of file search_graph.cc.

The documentation for this class was generated from the following file:

- api_delta.h

10.91 coco::delta_get_action Class Reference

Stored procedure class for computing the delta action specifier.

```
#include <api_delta.h>
```

Public Types

- typedef vdbl::context context
- typedef std::string return_type

Public Member Functions

- delta_get_action (vdbl::colid _c)
- delta_get_action (const delta_get_action &_i)
- virtual ~delta_get_action ()
- const std::string & operator() () const
- const std::string & def () const
- void setcontext (const context *c, const vdbl::row *r)

10.91.1 Detailed Description

A database method returning the delta action specifier, used as a stored procedure.

10.91.2 Member Typedef Documentation

10.91.2.1 typedef vdbl::context coco::delta_get_action::context

The evaluation context class

Definition at line 66 of file api_delta.h.

10.91.2.2 typedef std::string coco::delta_get_action::return_type

The return type of this stored procedure

Definition at line 68 of file api_delta.h.

10.91.3 Constructor & Destructor Documentation

10.91.3.1 coco::delta_get_action::delta_get_action (vdbl::colid _c) [inline]

Constructor, which sets the column id from _c

Definition at line 71 of file api_delta.h.

10.91.3.2 coco::delta_get_action::delta_get_action (const delta_get_action & _i) [inline]

Standard Copy Constructor

Definition at line 73 of file api_delta.h.

10.91.3.3 virtual coco::delta_get_action::~delta_get_action ( ) [inline, virtual]

Standard Destructor

Definition at line 76 of file api_delta.h.

10.91.4 Member Function Documentation

10.91.4.1 const std::string& coco::delta_get_action::def ( ) const [inline]

Default value of the procedure

Definition at line 81 of file api_delta.h.

10.91.4.2 const std::string& coco::delta_get_action::operator() ( ) const

Result of the procedure

10.91.4.3 void coco::delta_get_action::setcontext (const context * c, const vdbl::row * r) [inline]

Set the evaluation context for row `r` and context `c` (ignored!)

Definition at line 83 of file api_delta.h.

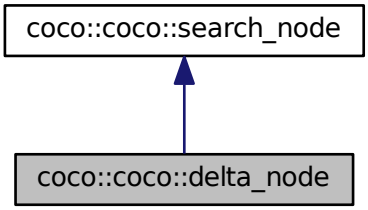
The documentation for this class was generated from the following file:

- api_delta.h

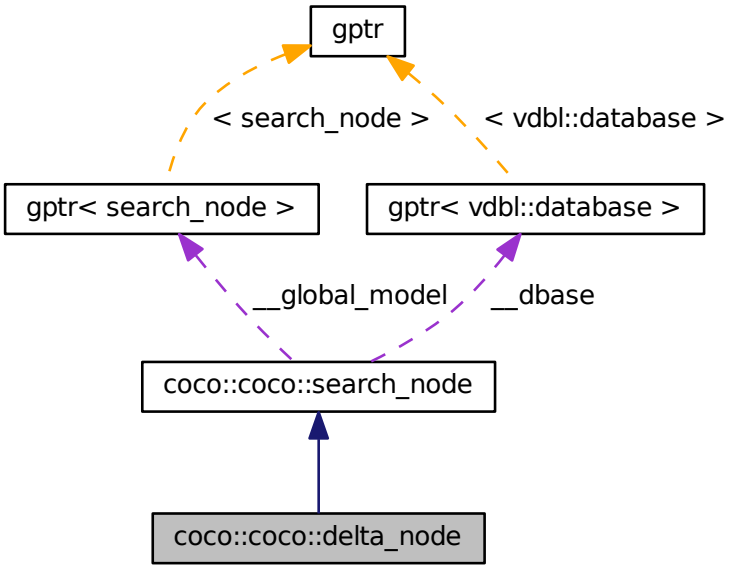
10.92 coco::coco::delta_node Class Reference

Class holding the delta nodes in the search graph.

Inheritance diagram for coco::coco::delta_node:



Collaboration diagram for coco::coco::delta_node:



Public Member Functions

- delta_node (const search_node_id &_i, const vdbl::userid &_dui, std::vector< delta_id > &__d, gptr< search_node > &_gm, gptr< vdbl::database > &_db, search_node_relation _snr=snr_reduction)
- virtual ~delta_node ()
- delta_node & operator= (const delta_node &__w)
- bool is_delta () const
- unsigned int n_deltas () const
- delta_id get_delta_id (unsigned int i) const
- delta get_delta (unsigned int i)
- const delta & get_delta (unsigned int i) const
- template<class _Iterator >
  void add_deltas (_Iterator _f, _Iterator _l)
- template<class _SeqContainer >
  void get_deltas (_SeqContainer &_c)
- vdbl::userid get_dbuserid () const
- gptr< search_node > * global_model () const
- gptr< vdbl::database > * database () const
- search_node_id get_id () const
- void set_id (const search_node_id &i)
- vdbl::rowid get_rowid () const
- void set_rowid (const vdbl::rowid &i)

- void keep (const annotation &_an)
- void keep (const std::vector< annotation > &_anv)
- void unkeep (const annotation &_an)
- void unkeep (const std::vector< annotation > &_anv)

Protected Member Functions

- search_node_relation parent_relation () const

Protected Attributes

- gptr< search_node > * __global_model
- gptr< vdbl::database > * __dbase
- vdbl::userid _dbuser
- search_node_relation _snr
- search_node_id _id
- std::vector< annotation > _keep
- vdbl::rowid _rid

Friends

- class search_graph

10.92.1 Detailed Description

This is a class of nodes stored in the search graph. It represents a delta node, i.e., a node which only stores the changes with respect to the parent model.

See also

delta

10.92.2 Constructor & Destructor Documentation

10.92.2.1 coco::coco::delta_node::delta_node (const search_node_id & *_i*, const vdbl::userid & *_dui*, std::vector< delta_id > & *__d*, gptr< search_node > & *_gm*, gptr< vdbl::database > & *_db*, search_node_relation *_snr* = `snr_reduction`) `[inline]`

This constructor generates a new delta node (a reduction node by default) with search_node_id `i` and search_node_relation `__snr` holding the delta ids stored in `__d`. The parameters `_gm`, `_db`, and `_dui` initialize the global model, search database, and the database user id, respectively.

Definition at line 220 of file search_graph.cc.

10.92.2.2 virtual coco::coco::delta_node::~delta_node ( ) `[inline, virtual]`

Standard Destructor

Definition at line 228 of file search_graph.cc.

10.92.3 Member Function Documentation

10.92.3.1 template<class _Iterator > void coco::coco::delta_node::add_deltas (_Iterator _f, _Iterator _l) [inline]

This method inserts all deltas in the collection of delta ids, which starts at iterator _f and ends before iterator _l to the list of delta ids in this delta node.

Definition at line 254 of file search_graph.cc.

10.92.3.2 gptr<vdbl::database>* coco::coco::search_node::database ( ) const [inline, inherited]

This is the accessor method for the search database.

Definition at line 157 of file search_graph.cc.

10.92.3.3 vdbl::userid coco::coco::search_node::get_dbuserid ( ) const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 151 of file search_graph.cc.

10.92.3.4 delta coco::delta_node::get_delta (unsigned int *i*) [inline]

A call to this method returns a copy of the ith delta.

Definition at line 395 of file search_node.hpp.

10.92.3.5 const delta & coco::delta_node::get_delta (unsigned int *i*) const [inline]

A call to this method returns a const reference to the ith delta.

Definition at line 411 of file search_node.hpp.

10.92.3.6 delta_id coco::coco::delta_node::get_delta_id (unsigned int *i*) const [inline]

A call to this method returns the delta id of the ith delta.

Definition at line 242 of file search_graph.cc.

10.92.3.7 template<class _SeqContainer > void coco::coco::delta_node::get_deltas (_SeqContainer & _c) [inline]

This method extracts all deltas from the delta node and adds it to the end of the container passed.

Definition at line 260 of file search_graph.cc.

10.92.3.8 search_node_id coco::coco::search_node::get_id ( ) const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 160 of file search_graph.cc.

10.92.3.9 vdbl::rowid coco::coco::search_node::get_rowid ( ) const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 166 of file search_graph.cc.

10.92.3.10 gptr<search_node>* coco::coco::search_node::global_model ( ) const [inline, inherited]

This is the accessor method for the global model.

Definition at line 154 of file search_graph.cc.

10.92.3.11 bool coco::coco::delta_node::is_delta ( ) const [inline, virtual]

This method is called to determine whether a search node stores only deltas (delta_node) or a full model (full_node). In case of a delta_node it returns `true`.

Reimplemented from coco::coco::search_node.

Definition at line 236 of file search_graph.cc.

10.92.3.12 void coco::coco::search_node::keep (const annotation & _an) [inline, inherited]

A call to this method informs the search node to be the keeper of the annotation _an.

Definition at line 173 of file search_graph.cc.

10.92.3.13 void coco::coco::search_node::keep (const std::vector< annotation > & _anv) [inline, inherited]

A call to this method informs the search node to be the keeper of all the annotations in _anv.

Definition at line 177 of file search_graph.cc.

10.92.3.14 unsigned int coco::coco::delta_node::n_deltas ( ) const [inline]

This method returns the number of deltas stored in this delta node.

Definition at line 239 of file search_graph.cc.

10.92.3.15 delta_node & coco::delta_node::operator= (const delta_node & __w) [inline]

Standard Assignment Operator

Definition at line 198 of file search_node.hpp.

10.92.3.16 search_node_relation coco::coco::search_node::parent_relation ( ) const [inline, protected, inherited]

A call to this method returns the relation of this node to its parent in the search graph.

Definition at line 110 of file search_graph.cc.

10.92.3.17 void coco::coco::search_node::set_id (const search_node_id & *i*) [inline, inherited]

This is the set method for the search node id of this node.

Definition at line 163 of file search_graph.cc.

10.92.3.18 void coco::coco::search_node::set_rowid (const vdbl::rowid & *i*) [inline, inherited]

This is the set method for the row id of this node.

Definition at line 169 of file search_graph.cc.

10.92.3.19 void coco::search_node::unkeep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of the annotation _an.

Definition at line 82 of file search_node.hpp.

10.92.3.20 void coco::search_node::unkeep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of all the annotations in _anv.

Definition at line 93 of file search_node.hpp.

10.92.4 Friends And Related Function Documentation

10.92.4.1 friend class search_graph [friend, inherited]

Definition at line 188 of file search_graph.cc.

10.92.5 Member Data Documentation

10.92.5.1 gptr<vdbl::database>* coco::coco::search_node::__dbase [protected, inherited]

This is a pointer to a global pointer to the search database associated to this search graph.

Definition at line 89 of file search_graph.cc.

10.92.5.2 gptr<search_node>* coco::coco::search_node::__global_model [protected, inherited]

This is a pointer to a global pointer to the top of the search graph. It is NULL for a standalone node or the top of the graph.

Definition at line 86 of file search_graph.cc.

10.92.5.3 vdbl::userid coco::coco::search_node::_dbuser [protected, inherited]

This member stores the database user id under which this search is performed. For the in-memory database this is irrelevant.

Definition at line 92 of file search_graph.cc.

10.92.5.4 search_node_id coco::coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 97 of file search_graph.cc.

10.92.5.5 std::vector<annotation> coco::coco::search_node::_keep [protected, inherited]

The _keep member is a list of all those annotations which are kept by this search node. If the node is destroyed all kept annotations are removed from the search database.

Definition at line 101 of file search_graph.cc.

10.92.5.6 vdbl::rowid coco::coco::search_node::_rid [protected, inherited]

This row id specifies the row in the search info table where the hook information for this search_node is stored

Definition at line 105 of file search_graph.cc.

10.92.5.7 search_node_relation coco::coco::search_node::_snr [protected, inherited]

This member specifies the relation of the search node to its parent(s) in the search graph.

Definition at line 95 of file search_graph.cc.

The documentation for this class was generated from the following files:

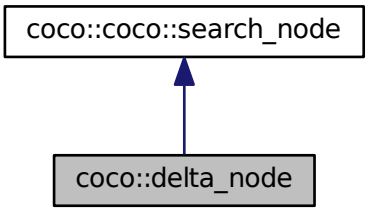
- search_node.h
- search_node.hpp

10.93 coco::delta_node Class Reference

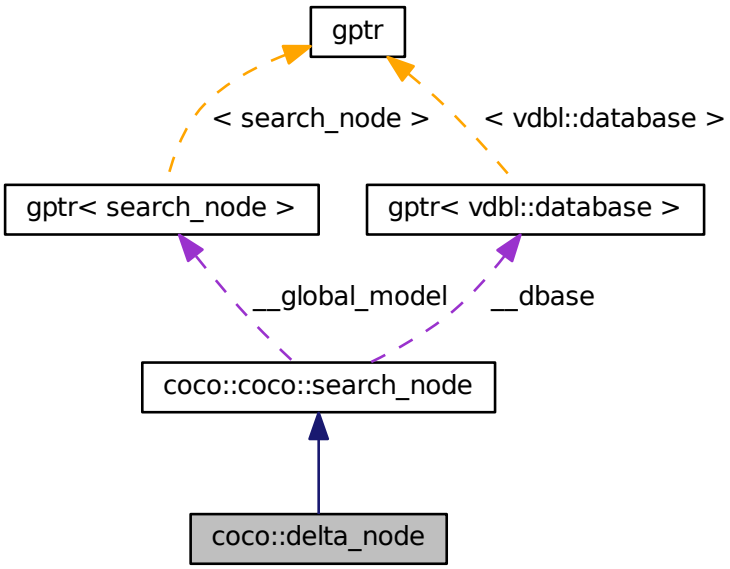
Class holding the delta nodes in the search graph.

`#include <search_node.h>`

Inheritance diagram for coco::delta_node:



Collaboration diagram for coco::delta_node:



Public Member Functions

- delta_node (const search_node_id &_i, const vdbl::userid &_dui, std::vector< delta_id > &__d, gptr< search_node > &_gm, gptr< vdbl::database > &_db, search_node_relation _snr=snr_reduction)

- virtual ~delta_node ()
- delta_node & operator= (const delta_node &__w)
- bool is_delta () const
- unsigned int n_deltas () const
- delta_id get_delta_id (unsigned int i) const
- delta get_delta (unsigned int i)
- const delta & get_delta (unsigned int i) const
- template<class _Iterator >
  void add_deltas (_Iterator _f, _Iterator _l)
- template<class _SeqContainer >
  void get_deltas (_SeqContainer &_c)
- vdbl::userid get_dbuserid () const
- gptr< search_node > * global_model () const
- gptr< vdbl::database > * database () const
- search_node_id get_id () const
- void set_id (const search_node_id &i)
- vdbl::rowid get_rowid () const
- void set_rowid (const vdbl::rowid &i)
- void keep (const annotation &_an)
- void keep (const std::vector< annotation > &_anv)
- void unkeep (const annotation &_an)
- void unkeep (const std::vector< annotation > &_anv)

Protected Member Functions

- search_node_relation parent_relation () const

Protected Attributes

- gptr< search_node > * __global_model
- gptr< vdbl::database > * __dbase
- vdbl::userid _dbuser
- search_node_relation _snr
- search_node_id _id
- std::vector< annotation > _keep
- vdbl::rowid _rid

Friends

- class search_graph

10.93.1 Detailed Description

This is a class of nodes stored in the search graph. It represents a delta node, i.e., a node which only stores the changes with respect to the parent model.

See also

delta

10.93.2 Constructor & Destructor Documentation

10.93.2.1 coco::delta_node::delta_node (const search_node_id & *_i*, const vdbl::userid & *_dui*, std::vector< delta_id > & *__d*, gptr< search_node > & *_gm*, gptr< vdbl::database > & *_db*, search_node_relation *_snr* = snr_reduction) [inline]

This constructor generates a new delta node (a reduction node by default) with search_node_id i and search_node_relation ___snr holding the delta ids stored in ___d. The parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively.

Definition at line 220 of file search_node.h.

10.93.2.2 virtual coco::delta_node::~delta_node ( ) [inline, virtual]

Standard Destructor

Definition at line 228 of file search_node.h.

10.93.3 Member Function Documentation

10.93.3.1 template<class _Iterator > void coco::delta_node::add_deltas (_Iterator *_f*, _Iterator *_l*) [inline]

This method inserts all deltas in the collection of delta ids, which starts at iterator _f and ends before iterator _l to the list of delta ids in this delta node.

Definition at line 254 of file search_node.h.

10.93.3.2 gptr<vdbl::database>* coco::coco::search_node::database ( ) const [inline, inherited]

This is the accessor method for the search database.

Definition at line 157 of file search_graph.cc.

10.93.3.3 vdbl::userid coco::coco::search_node::get_dbuserid ( ) const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 151 of file search_graph.cc.

10.93.3.4 delta coco::delta_node::get_delta (unsigned int *i*)

A call to this method returns a copy of the ith delta.

10.93.3.5 const delta& coco::delta_node::get_delta (unsigned int *i*) const

A call to this method returns a const reference to the ith delta.

10.93.3.6 delta_id coco::delta_node::get_delta_id (unsigned int *i*) const [inline]

A call to this method returns the delta id of the ith delta.

Definition at line 242 of file search_node.h.

10.93.3.7 template<class _SeqContainer > void coco::delta_node::get_deltas (_SeqContainer & _c) [inline]

This method extracts all deltas from the delta node and adds it to the end of the container passed.

Definition at line 260 of file search_node.h.

10.93.3.8 search_node_id coco::coco::search_node::get_id ( ) const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 160 of file search_graph.cc.

10.93.3.9 vdbl::rowid coco::coco::search_node::get_rowid ( ) const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 166 of file search_graph.cc.

10.93.3.10 gptr<search_node>* coco::coco::search_node::global_model ( ) const [inline, inherited]

This is the accessor method for the global model.

Definition at line 154 of file search_graph.cc.

10.93.3.11 bool coco::delta_node::is_delta ( ) const [inline, virtual]

This method is called to determine whether a search node stores only deltas (delta_node) or a full model (full_node). In case of a delta_node it returns `true`.

Reimplemented from coco::coco::search_node.

Definition at line 236 of file search_node.h.

10.93.3.12 void coco::coco::search_node::keep (const annotation & _an) [inline, inherited]

A call to this method informs the search node to be the keeper of the annotation _an.

Definition at line 173 of file search_graph.cc.

10.93.3.13 void coco::coco::search_node::keep (const std::vector< annotation > & _anv) [inline, inherited]

A call to this method informs the search node to be the keeper of all the annotations in _anv.

Definition at line 177 of file search_graph.cc.

10.93.3.14 unsigned int coco::delta_node::n_deltas ( ) const [inline]

This method returns the number of deltas stored in this delta node.

Definition at line 239 of file search_node.h.

10.93.3.15 delta_node& coco::delta_node::operator= (const delta_node & __w)

Standard Assignment Operator

10.93.3.16 search_node_relation coco::coco::search_node::parent_relation ( ) const [inline, protected, inherited]

A call to this method returns the relation of this node to its parent in the search graph.

Definition at line 110 of file search_graph.cc.

10.93.3.17 void coco::coco::search_node::set_id (const search_node_id & *i*) [inline, inherited]

This is the set method for the search node id of this node.

Definition at line 163 of file search_graph.cc.

10.93.3.18 void coco::coco::search_node::set_rowid (const vdbl::rowid & *i*) [inline, inherited]

This is the set method for the row id of this node.

Definition at line 169 of file search_graph.cc.

10.93.3.19 void coco::search_node::unkeep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of the annotation _an.

Definition at line 82 of file search_node.hpp.

10.93.3.20 void coco::search_node::unkeep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of all the annotations in _anv.

Definition at line 93 of file search_node.hpp.

10.93.4 Friends And Related Function Documentation

10.93.4.1 friend class search_graph [friend, inherited]

Definition at line 188 of file search_graph.cc.

10.93.5 Member Data Documentation

10.93.5.1 gptr<vdbl::database>* coco::coco::search_node::__dbase [protected, inherited]

This is a pointer to a global pointer to the search database associated to this search graph.

Definition at line 89 of file search_graph.cc.

10.93.5.2 gptr<search_node>* coco::coco::search_node::__global_model [protected, inherited]

This is a pointer to a global pointer to the top of the search graph. It is NULL for a standalone node or the top of the graph.

Definition at line 86 of file search_graph.cc.

10.93.5.3 vdbl::userid coco::coco::search_node::_dbuser [protected, inherited]

This member stores the database user id under which this search is performed. For the in-memory database this is irrelevant.

Definition at line 92 of file search_graph.cc.

10.93.5.4 search_node_id coco::coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 97 of file search_graph.cc.

10.93.5.5 std::vector<annotation> coco::coco::search_node::_keep [protected, inherited]

The _keep member is a list of all those annotations which are kept by this search node. If the node is destroyed all kept annotations are removed from the search database.

Definition at line 101 of file search_graph.cc.

10.93.5.6 vdbl::rowid coco::coco::search_node::_rid [protected, inherited]

This row id specifies the row in the search info table where the hook information for this search_node is stored

Definition at line 105 of file search_graph.cc.

10.93.5.7 search_node_relation coco::coco::search_node::_snr [protected, inherited]

This member specifies the relation of the search node to its parent(s) in the search graph.

Definition at line 95 of file search_graph.cc.

The documentation for this class was generated from the following file:

- search_node.h

10.94 coco::der_eval Class Reference

Backward gradient evaluation with prepared derivative data.

`#include <der_evaluator.h>`

Inheritance diagram for coco::der_eval:

Collaboration diagram for coco::der_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- der_eval (std::vector< std::vector< double > > &__der_data, variable_indicator &__v, const model &__m, std::vector< std::vector< double > > *__d, std::vector< double > &__grad)
- der_eval (const der_eval &__d)
- ~der_eval ()
- void new_point (std::vector< std::vector< double > > &__der_data, const variable_indicator &__v)
- void new_result (std::vector< double > &__grad)
- void set_mult (double scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.94.1 Detailed Description

This class is a cached backward evaluator performing a gradient evaluation with available partial derivative information.

10.94.2 Member Typedef Documentation

10.94.2.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.94.2.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.94.2.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.94.3 Constructor & Destructor Documentation

10.94.3.1 coco::der_eval::der_eval (std::vector< std::vector< double > > & __der_data, variable_indicator & __v, const model & __m, std::vector< std::vector< double > > * __d, std::vector< double > & __grad) [inline]

Constructor: __der_data contains the partial derivative information, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies the DAG, and __d the cache (may be NULL). Eventually, __grad is a reference to the result vector. This vector must have the correct length and has to be initialized with zero, since all components are actually computed by adding to the components of __grad.

Definition at line 1093 of file der_evaluator.h.

10.94.3.2 coco::der_eval::der_eval (const der_eval & __d) [inline]

Standard Copy Constructor

Definition at line 1107 of file der_evaluator.h.

10.94.3.3 coco::der_eval::~der_eval ( ) [inline]

Standard Destructor

Definition at line 1110 of file der_evaluator.h.

10.94.4 Member Function Documentation

10.94.4.1 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.94.4.2 int coco::der_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1148 of file der_evaluator.h.

10.94.4.3 bool coco::der_eval::calculate_value (bool eval_all) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1241 of file der_evaluator.h.

10.94.4.4 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.94.4.5 void coco::der_eval::cleanup (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1197 of file der_evaluator.h.

10.94.4.6 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.94.4.7 void coco::der_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1142 of file der_evaluator.h.

10.94.4.8 bool coco::der_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the gradient for this node is already available.

Reimplemented from coco::coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1072 of file der_evaluator.h.

10.94.4.9 void coco::der_eval::new_point (std::vector< std::vector< double > > & __der_data, const variable_indicator & __v) [inline]

This method changes the evaluation point to new derivative data ___der. The parameter ___v specifies which variables have changed value since the last evaluation.

Definition at line 1115 of file der_evaluator.h.

10.94.4.10 void coco::der_eval::new_result (std::vector< double > & __grad) [inline]

This method changes the result vector to ___grad.

Definition at line 1123 of file der_evaluator.h.

10.94.4.11 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.94.4.12 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.94.4.13 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.94.4.14 void coco::der_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1208 of file der_evaluator.h.

10.94.4.15 void coco::der_eval::set_mult (double scal) [inline]

This method causes the gradient to be multiplied by `scal`.

Definition at line 1129 of file der_evaluator.h.

10.94.4.16 expression_const_walker coco::der_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 1136 of file der_evaluator.h.

10.94.4.17 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.94.4.18 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.94.4.19 int coco::der_eval::update (const bool & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1215 of file der_evaluator.h.

10.94.4.20 int coco::der_eval::update (const expression_node & *__data*, const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1222 of file der_evaluator.h.

10.94.4.21 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 892 of file search_graph.cc.

10.94.4.22 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.94.4.23 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.94.4.24 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

The documentation for this class was generated from the following file:

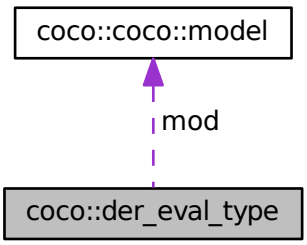
- der_evaluator.h

10.95 coco::der_eval_type Struct Reference

Visitor data for der_eval.

```
#include <der_evaluator.h>
```

Collaboration diagram for coco::der_eval_type:



Public Attributes

- std::vector< std::vector < double > > * d_data
- std::vector< std::vector < double > > * d_cache
- std::vector< double > * grad_vec
- const model * mod
- double mult
- double mult_trans
- unsigned int child_n

10.95.1 Detailed Description

This class is the visitor data type for the der_eval evaluator.

10.95.2 Member Data Documentation

10.95.2.1 unsigned int coco::der_eval_type::child_n

children counter

Definition at line 1051 of file der_evaluator.h.

10.95.2.2 std::vector<std::vector<double> >* coco::der_eval_type::d_cache

derivative cache

Definition at line 1046 of file der_evaluator.h.

10.95.2.3 std::vector<std::vector<double> >* coco::der_eval_type::d_data

partial derivative data

Definition at line 1045 of file der_evaluator.h.

10.95.2.4 std::vector<double>* coco::der_eval_type::grad_vec

the gradient vector

Definition at line 1047 of file der_evaluator.h.

10.95.2.5 const model* coco::der_eval_type::mod

the DAG

Definition at line 1048 of file der_evaluator.h.

10.95.2.6 double coco::der_eval_type::mult

the current value on the path

Definition at line 1049 of file der_evaluator.h.

10.95.2.7 double coco::der_eval_type::mult_trans

transfer variable for mult

Definition at line 1050 of file der_evaluator.h.

The documentation for this struct was generated from the following file:

- der_evaluator.h

10.96 coco::model::detect_0chain_visitor Class Reference

`#include <model.hpp>`

Inheritance diagram for coco::model::detect_0chain_visitor:

Collaboration diagram for coco::model::detect_0chain_visitor:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- detect_0chain_visitor (std::vector< unsigned int > &__t, std::vector< unsigned int > &__d, std::vector< unsigned int > &__nn, unsigned int &d)
- detect_0chain_visitor (const detect_0chain_visitor &__x)
- ~detect_0chain_visitor ()
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- bool is_cached (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- int update (const std::pair< unsigned int, unsigned int > &__rval)
- int update (const expression_node &__data, const std::pair< unsigned int, unsigned int > &__rval)
- std::pair< unsigned int, unsigned int > calculate_value (bool eval_all)

10.96.1 Detailed Description

The detect_0chain_visitor is used to find 0-chains, i.e. univariate functions within the DAG.

10.96.2 Member Typedef Documentation

10.96.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.96.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.96.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.96.3 Constructor & Destructor Documentation

10.96.3.1 coco::model::detect_0chain_visitor::detect_0chain_visitor (std::vector< unsigned int > & __t, std::vector< unsigned int > & __d, std::vector< unsigned int > & __nn, unsigned int & d) [inline]

Constructor, which initializes the vector of types with __t, the vector of depths with __d, the vector of chain bases with __nn, and the counter with d.

Definition at line 2334 of file model.hpp.

10.96.3.2 coco::model::detect_0chain_visitor::detect_0chain_visitor (const detect_0chain_visitor & __x) [inline]

Standard Copy Constructor

Definition at line 2347 of file model.hpp.

10.96.3.3 coco::model::detect_0chain_visitor::~detect_0chain_visitor ( ) [inline]

Standard Destructor

Definition at line 2351 of file model.hpp.

10.96.4 Member Function Documentation

10.96.4.1 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.96.4.2 void coco::model::detect_0chain_visitor::calculate (const expression_node & __data) [inline]

This is a method which has to be defined for a cached forward evaluator.

See also

cached_forward_evaluator_base.

Definition at line 2383 of file model.hpp.

10.96.4.3 std::pair<unsigned int,unsigned int> coco::model::detect_0chain_visitor::calculate_value (bool *eval_all*) [inline, virtual]

This is a method which has to be defined for a cached forward evaluator.

See also

cached_forward_evaluator_base.

Reimplemented from coco::coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression-_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2412 of file model.hpp.

10.96.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.96.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.96.4.6 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.96.4.7 void coco::model::detect_0chain_visitor::initialize ( ) [inline, virtual]

This is a method which has to be defined for a cached forward evaluator.

See also

cached_forward_evaluator_base.

Reimplemented from coco::coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression-_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2356 of file model.hpp.

10.96.4.8 int coco::model::detect_0chain_visitor::initialize (const expression_node & *__data*) [inline]

This is a method which has to be defined for a cached forward evaluator.

See also

cached_forward_evaluator_base.

Definition at line 2365 of file model.hpp.

10.96.4.9 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & *__data*) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.96.4.10 bool coco::model::detect_0chain_visitor::is_cached (const expression_node & *__data*) [inline]

This is a method which has to be defined for a cached forward evaluator.

See also

cached_forward_evaluator_base.

Definition at line 2358 of file model.hpp.

10.96.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.96.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.96.4.13 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.96.4.14 void coco::model::detect_0chain_visitor::retrieve_from_cache (const expression_node & __data) [inline]

This is a method which has to be defined for a cached forward evaluator.

See also

cached_forward_evaluator_base.

Definition at line 2361 of file model.hpp.

10.96.4.15 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.96.4.16 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The ___data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.96.4.17 int coco::model::detect_0chain_visitor::update (const std::pair< unsigned int, unsigned int > & __rval) [inline]

This is a method which has to be defined for a cached forward evaluator.

See also

cached_forward_evaluator_base.

Definition at line 2389 of file model.hpp.

10.96.4.18 int coco::model::detect_0chain_visitor::update (const expression_node & __data, const std::pair< unsigned int, unsigned int > & __rval) [inline]

This is a method which has to be defined for a cached forward evaluator.

See also

cached_forward_evaluator_base.

Definition at line 2391 of file model.hpp.

10.96.4.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.96.4.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.96.4.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.96.4.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- model.hpp

10.97 coco::model::detect_0chain_visitor_st Struct Reference

`#include <model.hpp>`

Public Attributes

- std::pair< unsigned int, unsigned int > r
- unsigned int * f
- std::vector< unsigned int > * t

- std::vector< unsigned int > * d
- std::vector< unsigned int > * nn
- bool new_number

10.97.1 Detailed Description

This struct is the visitor data of the model::detect_0chain_visitor.

10.97.2 Member Data Documentation

10.97.2.1 std::vector<unsigned int>* coco::model::detect_0chain_visitor_st::d

vector of node depths

Definition at line 2313 of file model.hpp.

10.97.2.2 unsigned int* coco::model::detect_0chain_visitor_st::f

counter

Definition at line 2311 of file model.hpp.

10.97.2.3 bool coco::model::detect_0chain_visitor_st::new_number

is this a new 0-chain?

Definition at line 2315 of file model.hpp.

10.97.2.4 std::vector<unsigned int>* coco::model::detect_0chain_visitor_st::nn

vector of 0-chain bases

Definition at line 2314 of file model.hpp.

10.97.2.5 std::pair<unsigned int,unsigned int> coco::model::detect_0chain_visitor_st::r

return value (node number, depth)

Definition at line 2310 of file model.hpp.

10.97.2.6 std::vector<unsigned int>* coco::model::detect_0chain_visitor_st::t

vector of node types

Definition at line 2312 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

10.98 coco::dfunc_eval Class Reference

`#include <dfunc_evaluator.h>`

Inheritance diagram for coco::dfunc_eval:

Collaboration diagram for coco::dfunc_eval:

Public Types

- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- dfunc_eval (const _T &__x, const variable_indicator &__v, const model &__m, std::vector< retval > *__c)
- dfunc_eval (const dfunc_eval &__v)
- ~dfunc_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_point (const _T &__x, const variable_indicator &__v)
- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const retval &__rval)
- int update (const expression_node &__data, const retval &__rval)
- retval calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.98.1 Member Typedef Documentation

10.98.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.98.1.2 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.98.2 Constructor & Destructor Documentation

10.98.2.1 coco::dfunc_eval::dfunc_eval (const _T & __x, const variable_indicator & __v, const model & __m, std::vector< retval > * __c) [inline]

Definition at line 135 of file dfunc_evaluator.h.

10.98.2.2 coco::dfunc_eval::dfunc_eval (const dfunc_eval & __v) [inline]

Definition at line 146 of file dfunc_evaluator.h.

10.98.2.3 coco::dfunc_eval::~dfunc_eval ( ) [inline]

Definition at line 148 of file dfunc_evaluator.h.

10.98.3 Member Function Documentation

10.98.3.1 void coco::dfunc_eval::calculate (const expression_node & __data) [inline]

Definition at line 238 of file dfunc_evaluator.h.

10.98.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.98.3.3 retval coco::dfunc_eval::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 655 of file dfunc_evaluator.h.

10.98.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.98.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.98.3.6 void coco::dfunc_eval::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 160 of file dfunc_evaluator.h.

10.98.3.7 int coco::dfunc_eval::initialize (const expression_node & *__data*) [inline]

Definition at line 162 of file dfunc_evaluator.h.

10.98.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The `__data` parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.98.3.9 bool coco::dfunc_eval::is_cached (const node_data_type & *__data*) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 80 of file dfunc_evaluator.h.

10.98.3.10 void coco::dfunc_eval::new_point (const _T & *__x*, const variable_indicator & *__v*) [inline]

Definition at line 154 of file dfunc_evaluator.h.

10.98.3.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.98.3.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.98.3.13 void coco::dfunc_eval::retrieve_from_cache (const expression_node & *__data*) [inline]

Definition at line 249 of file dfunc_evaluator.h.

10.98.3.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.98.3.15 expression_const_walker coco::dfunc_eval::short_cut_to (const expression_node & *__data*) [inline]

Definition at line 150 of file dfunc_evaluator.h.

10.98.3.16 int coco::dfunc_eval::update (const retval & *__rval*) [inline]

Definition at line 254 of file dfunc_evaluator.h.

10.98.3.17 int coco::dfunc_eval::update (const expression_node & *__data*, const retval & *__rval*) [inline]

Definition at line 260 of file dfunc_evaluator.h.

10.98.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.98.3.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.98.3.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.98.3.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.98.3.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.98.3.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

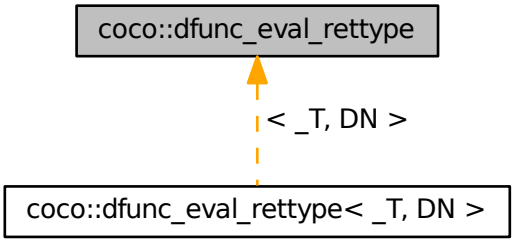
The documentation for this class was generated from the following file:

- dfunc_evaluator.h

10.99 coco::dfunc_eval_rettype Struct Reference

`#include <dfunc_evaluator.h>`

Inheritance diagram for coco::dfunc_eval_rettype:



Public Attributes

- _T f [DN+1]

10.99.1 Member Data Documentation

10.99.1.1 _T coco::dfunc_eval_rettype::f[DN+1]

Definition at line 51 of file dfunc_evaluator.h.

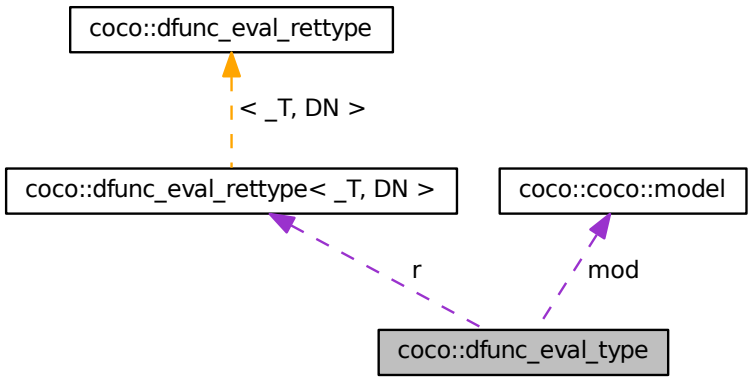
The documentation for this struct was generated from the following file:

- dfunc_evaluator.h

10.100 coco::dfunc_eval_type Struct Reference

`#include <dfunc_evaluator.h>`

Collaboration diagram for coco::dfunc_eval_type:



Public Attributes

- unsigned int _0chnbase
- _T x
- std::vector < dfunc_eval_rettype< _T, DN > > * cache
- const model * mod
- void * p
- _T d
- dfunc_eval_rettype< _T, DN > r
- unsigned int n

10.100.1 Member Data Documentation

10.100.1.1 unsigned int coco::dfunc_eval_type::_0chnbase

Definition at line 57 of file dfunc_evaluator.h.

10.100.1.2 std::vector<dfunc_eval_rettype<_T, DN> >* coco::dfunc_eval_type::cache

Definition at line 59 of file dfunc_evaluator.h.

10.100.1.3 _T coco::dfunc_eval_type::d

Definition at line 62 of file dfunc_evaluator.h.

10.100.1.4 const model* coco::dfunc_eval_type::mod

Definition at line 60 of file dfunc_evaluator.h.

10.100.1.5 unsigned int coco::dfunc_eval_type::n

Definition at line 64 of file dfunc_evaluator.h.

10.100.1.6 void* coco::dfunc_eval_type::p

Definition at line 61 of file dfunc_evaluator.h.

10.100.1.7 dfunc_eval_rettype<_T, DN> coco::dfunc_eval_type::r

Definition at line 63 of file dfunc_evaluator.h.

10.100.1.8 _T coco::dfunc_eval_type::x

Definition at line 58 of file dfunc_evaluator.h.

The documentation for this struct was generated from the following file:

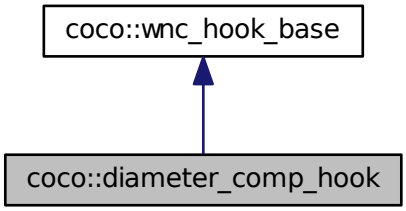
- dfunc_evaluator.h

10.101 coco::diameter_comp_hook Class Reference

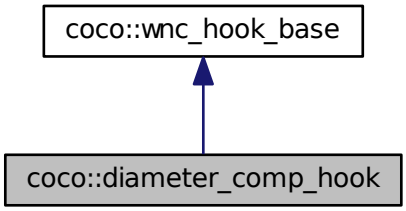
The log-volume computation hook (work node computation hook)

`#include <diameter_hook.h>`

Inheritance diagram for coco::diameter_comp_hook:



Collaboration diagram for coco::diameter_comp_hook:



Public Member Functions

- diameter_comp_hook ()
- virtual ~diameter_comp_hook ()
- diameter_comp_hook * new_copy () const
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *a=NULL, std::list< certificate > *b=NULL) const
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)
- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.101.1 Detailed Description

This class is a work node computation hook (see work_node_comp_hook), which calculates the diameter and the minimal width of the work node (variables only) and keeps it in columns “diameter” and “min width”.

10.101.2 Constructor & Destructor Documentation

10.101.2.1 coco::diameter_comp_hook::diameter_comp_hook ( ) [inline]

Standard Constructor

Definition at line 46 of file diameter_hook.h.

10.101.2.2 virtual coco::diameter_comp_hook::~diameter_comp_hook ( ) [inline, virtual]

Standard Destructor

Definition at line 49 of file diameter_hook.h.

10.101.3 Member Function Documentation

10.101.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected, inherited]

Remove all columns, whose identifiers are stored in ___managed_cols from table stable.

Definition at line 34 of file comp_hook.cc.

10.101.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 238 of file comp_hook.h.

10.101.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 135 of file comp_hook.h.

10.101.3.4 bool coco::diameter_comp_hook::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon unregistering this hook, destroy the column "diameter" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 83 of file diameter_hook.h.

10.101.3.5 bool coco::diameter_comp_hook::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the column "diameter" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 76 of file diameter_hook.h.

10.101.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline, inherited]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.101.3.7 diameter_comp_hook* coco::diameter_comp_hook::new_copy ( ) const [inline, virtual]

Clone Operation

Implements coco::wnc_hook_base.

Definition at line 52 of file diameter_hook.h.

10.101.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_id of work_node `wn`.

Definition at line 43 of file comp_hook.cc.

10.101.3.9 void coco::diameter_comp_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *a* = `NULL`, std::list< certificate > * *b* = `NULL`) const [inline, virtual]

The evaluation operator of this computation hook. It stores the log-volume of the work_node `wn` in dbt_row `dbr`.

Implements coco::wnc_hook_base.

Definition at line 56 of file diameter_hook.h.

10.101.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_relation of work_node `wn`.

Definition at line 46 of file comp_hook.cc.

The documentation for this class was generated from the following file:

- diameter_hook.h

10.102 coco::diffI Class Reference

`#include <diffI.h>`

Public Member Functions

- diffI (void)
- diffI (double a)
- diffI (double a, double b)
- diffI (int n)
- diffI (double a, int n)
- diffI (double a, double b, int n)
- diffI (const I &a, int n)
- diffI (const diffI &a)

- virtual ~diffI ()
- diffI & operator= (const diffI &a)
- diffI & operator= (const I &a)
- diffI & operator= (double a)
- diffI & operator+= (const diffI &b)
- diffI & operator+= (double b)
- diffI & operator+= (const I &b)
- diffI & operator-= (const diffI &b)
- diffI & operator-= (double b)
- diffI & operator-= (const I &b)
- diffI & operator*= (const diffI &b)
- diffI & operator*= (double b)
- diffI & operator*= (const I &b)
- diffI & operator/= (const diffI &b)
- diffI & operator/= (double b)
- diffI & operator/= (const I &b)
- I & operator[ ] (unsigned int pos) const
- I operator() (unsigned int pos) const
- I & setVal (unsigned int pos, const I &val)
- void setDeg (int n)
- void shift (unsigned int n)
- diffI getShifted (unsigned int n) const

Static Public Member Functions

- static diffI getVar (const double &x)
- static diffI getVar (const double &x, const double &y)
- static diffI getVar (const double &x, int n)
- static diffI getVar (const double &x, const double &y, int n)
- static diffI getVar (const I &x, int n)

Public Attributes

- int MAXDEG

10.102.1 Constructor & Destructor Documentation

10.102.1.1 coco::diffI::diffI (void) [inline]

Definition at line 123 of file diffI.h.

10.102.1.2 coco::diffI::diffI (double *a*) [inline]

Definition at line 137 of file diffI.h.

10.102.1.3 coco::diffI::diffI (double *a*, double *b*) [inline]

Definition at line 145 of file diffI.h.

10.102.1.4 coco::diffI::diffI (int *n*) [inline]

Definition at line 130 of file diffI.h.

10.102.1.5 coco::diffI::diffI (double *a*, int *n*) [inline]

Definition at line 153 of file diffI.h.

10.102.1.6 coco::diffI::diffI (double *a*, double *b*, int *n*) [inline]

Definition at line 161 of file diffI.h.

10.102.1.7 coco::diffI::diffI (const I & *a*, int *n*) [inline]

Definition at line 169 of file diffI.h.

10.102.1.8 coco::diffI::diffI (const diffI & *a*) [inline]

Definition at line 177 of file diffI.h.

10.102.1.9 coco::diffI::~diffI ( ) [inline, virtual]

Definition at line 280 of file diffI.h.

10.102.2 Member Function Documentation

10.102.2.1 diffI coco::diffI::getShifted (unsigned int *n*) const [inline]

Definition at line 294 of file diffI.h.

10.102.2.2 diffI coco::diffI::getVar (const double & *x*) [inline, static]

Definition at line 239 of file diffI.h.

10.102.2.3 diffI coco::diffI::getVar (const double & *x*, const double & *y*) [inline, static]

Definition at line 247 of file diffI.h.

10.102.2.4 diffI coco::diffI::getVar (const double & *x*, int *n*) [inline, static]

Definition at line 255 of file diffI.h.

10.102.2.5 diffI coco::diffI::getVar (const double & *x*, const double & *y*, int *n*) [inline, static]

Definition at line 263 of file diffI.h.

10.102.2.6 diffI coco::diffI::getVar (const I & *x*, int *n*) [inline, static]

Definition at line 271 of file diffI.h.

10.102.2.7 interval coco::diffI::operator() (unsigned int *pos*) const

Definition at line 983 of file diffI.cc.

10.102.2.8 diffI & coco::diffI::operator*= (const diffI & *b*)

Definition at line 441 of file diffI.cc.

10.102.2.9 diffI & coco::diffI::operator*= (double *b*)

Definition at line 104 of file diffI.cc.

10.102.2.10 diffI & coco::diffI::operator*= (const I & *b*)

Definition at line 118 of file diffI.cc.

10.102.2.11 diffI & coco::diffI::operator+= (const diffI & *b*)

Definition at line 20 of file diffI.cc.

10.102.2.12 diffI & coco::diffI::operator+= (double *b*)

Definition at line 38 of file diffI.cc.

10.102.2.13 diffI & coco::diffI::operator+= (const I & *b*)

Definition at line 50 of file diffI.cc.

10.102.2.14 diffI & coco::diffI::operator-= (const diffI & *b*)

Definition at line 62 of file diffI.cc.

10.102.2.15 diffI & coco::diffI::operator-= (double *b*)

Definition at line 80 of file diffI.cc.

10.102.2.16 diffI & coco::diffI::operator-= (const I & *b*)

Definition at line 92 of file diffI.cc.

10.102.2.17 diffI & coco::diffI::operator/= (const diffI & *b*)

Definition at line 468 of file diffI.cc.

10.102.2.18 diffI & coco::diffI::operator/= (double *b*)

Definition at line 132 of file diffI.cc.

10.102.2.19 diffI & coco::diffI::operator/= (const I & *b*)

Definition at line 146 of file diffI.cc.

10.102.2.20 diffI & coco::diffI::operator= (const diffI & *a*) [inline]

Definition at line 184 of file diffI.h.

10.102.2.21 diffI & coco::diffI::operator= (const I & *a*) [inline]

Definition at line 205 of file diffI.h.

10.102.2.22 diffI & coco::diffI::operator= (double *a*) [inline]

Definition at line 218 of file diffI.h.

10.102.2.23 interval & coco::diffI::operator[ ] (unsigned int *pos*) const [inline]

Definition at line 231 of file diffI.h.

10.102.2.24 void coco::diffI::setDeg (int *n*)

Definition at line 417 of file diffI.cc.

10.102.2.25 interval & coco::diffI::setVal (unsigned int *pos*, const I & *val*)

Definition at line 1001 of file diffI.cc.

10.102.2.26 void coco::diffI::shift (unsigned int *n*) [inline]

Definition at line 286 of file diffI.h.

10.102.3 Member Data Documentation

10.102.3.1 int coco::diffI::MAXDEG

Definition at line 28 of file diffI.h.

The documentation for this class was generated from the following files:

- diffI.h
- diffI.cc

10.103 coco::diffNumber Class Reference

`#include <diffNumber.h>`

Public Member Functions

- diffNumber (void)
- diffNumber (double a)
- diffNumber (int n)
- diffNumber (double a, int n)

- diffNumber (const diffNumber &a)
- virtual ~diffNumber ()
- diffNumber & operator= (const diffNumber &a)
- double & operator[ ] (unsigned int pos) const
- double operator() (unsigned int pos) const
- void setDeg (int n)
- double setVal (unsigned int n, double v)
- void shift (unsigned int n)
- diffNumber getShifted (unsigned int n) const

Static Public Member Functions

- static diffNumber getVar (const double &x)
- static diffNumber getVar (const double &x, int n)

Public Attributes

- int MAXDEG

10.103.1 Constructor & Destructor Documentation

10.103.1.1 coco::diffNumber::diffNumber (void) [inline]

Definition at line 90 of file diffNumber.h.

10.103.1.2 coco::diffNumber::diffNumber (double *a*) [inline]

Definition at line 107 of file diffNumber.h.

10.103.1.3 coco::diffNumber::diffNumber (int *n*) [inline]

Definition at line 99 of file diffNumber.h.

10.103.1.4 coco::diffNumber::diffNumber (double *a*, int *n*) [inline]

Definition at line 116 of file diffNumber.h.

10.103.1.5 coco::diffNumber::diffNumber (const diffNumber & *a*) [inline]

Definition at line 125 of file diffNumber.h.

10.103.1.6 coco::diffNumber::~diffNumber ( ) [inline, virtual]

Definition at line 282 of file diffNumber.h.

10.103.2 Member Function Documentation

10.103.2.1 diffNumber coco::diffNumber::getShifted (unsigned int *n*) const [inline]

Definition at line 296 of file diffNumber.h.

10.103.2.2 diffNumber coco::diffNumber::getVar (const double & *x*) [inline, static]

Definition at line 163 of file diffNumber.h.

10.103.2.3 diffNumber coco::diffNumber::getVar (const double & *x*, int *n*) [inline, static]

Definition at line 171 of file diffNumber.h.

10.103.2.4 double coco::diffNumber::operator() (unsigned int *pos*) const

Definition at line 445 of file diffNumber.cc.

10.103.2.5 diffNumber & coco::diffNumber::operator= (const diffNumber & *a*) [inline]

Definition at line 133 of file diffNumber.h.

10.103.2.6 double & coco::diffNumber::operator[ ] (unsigned int *pos*) const [inline]

Definition at line 155 of file diffNumber.h.

10.103.2.7 void coco::diffNumber::setDeg (int *n*)

Definition at line 11 of file diffNumber.cc.

10.103.2.8 double coco::diffNumber::setVal (unsigned int *n*, double *v*)

Definition at line 463 of file diffNumber.cc.

10.103.2.9 void coco::diffNumber::shift (unsigned int *n*) [inline]

Definition at line 288 of file diffNumber.h.

10.103.3 Member Data Documentation

10.103.3.1 int coco::diffNumber::MAXDEG

Definition at line 25 of file diffNumber.h.

The documentation for this class was generated from the following files:

- diffNumber.h
- diffNumber.cc

10.104 coco::double_to_uint64_u Union Reference

`#include <d1hashhelp.h>`

Public Attributes

- double d
- uint64_t u

10.104.1 Member Data Documentation

10.104.1.1 double coco::double_to_uint64_u::d

Definition at line 49 of file d1hashhelp.h.

10.104.1.2 uint64_t coco::double_to_uint64_u::u

Definition at line 50 of file d1hashhelp.h.

The documentation for this union was generated from the following file:

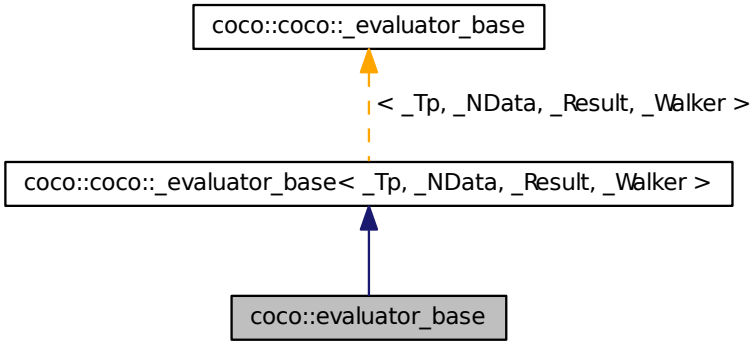
- d1hashhelp.h

10.105 coco::evaluator_base Class Reference

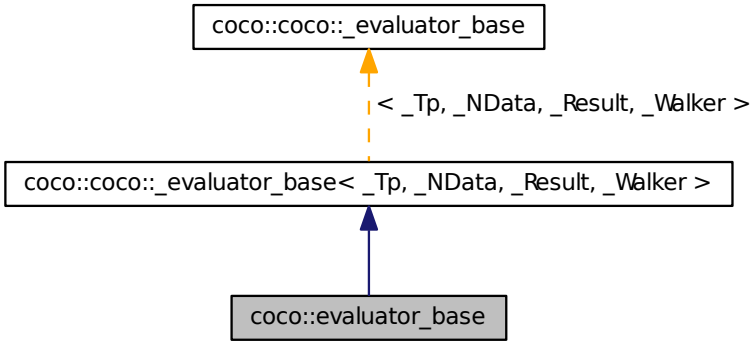
Base class of all (non-caching) evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::evaluator_base:



Collaboration diagram for coco::evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker
- typedef _Tp data_type

Public Member Functions

- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)
- virtual return_value vvalue ()
- virtual return_value value ()
- virtual int vcollect (const return_value &__cresult)
- virtual int collect (const node_data_type &__data, const return_value &__cresult)
- virtual void postorder (const node_data_type &__data)

Protected Attributes

- _Tp eval_data

10.105.1 Detailed Description

This class is the base class of all non-caching evaluators. Basically, it is a visitor to expression_node nodes in a DAG, based on _evaluator_base.

10.105.2 Member Typedef Documentation

10.105.2.1 typedef _Base::const_walker coco::evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 390 of file evaluator.h.

10.105.2.2 typedef _Tp coco::coco::_evaluator_base::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 306 of file search_graph.cc.

10.105.2.3 typedef _Base::node_data_type coco::evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 386 of file evaluator.h.

10.105.2.4 typedef _Base::return_value coco::evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 388 of file evaluator.h.

10.105.3 Member Function Documentation

10.105.3.1 virtual int coco::coco::_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__cresult*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 360 of file search_graph.cc.

10.105.3.2 virtual void coco::coco::_evaluator_base::postorder (const node_data_type & *__data*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited.

Definition at line 365 of file search_graph.cc.

10.105.3.3 virtual int coco::evaluator_base::preorder (const node_data_type & __data) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 401 of file evaluator.h.

10.105.3.4 virtual const_walker coco::evaluator_base::short_cut_to (const node_data_type & __data) [inline, virtual]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the const_walker returned is the target of the short-cut.

Definition at line 405 of file evaluator.h.

10.105.3.5 virtual return_value coco::coco::_evaluator_base::value ( ) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value.

Definition at line 337 of file search_graph.cc.

10.105.3.6 virtual int coco::coco::_evaluator_base::vcollect (const return_value & __cresult) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 349 of file search_graph.cc.

10.105.3.7 virtual return_value coco::coco::_evaluator_base::vvalue ( ) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value.

Definition at line 333 of file search_graph.cc.

10.105.4 Member Data Documentation

10.105.4.1 _Tp coco::coco::_evaluator_base::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 317 of file search_graph.cc.

The documentation for this class was generated from the following file:

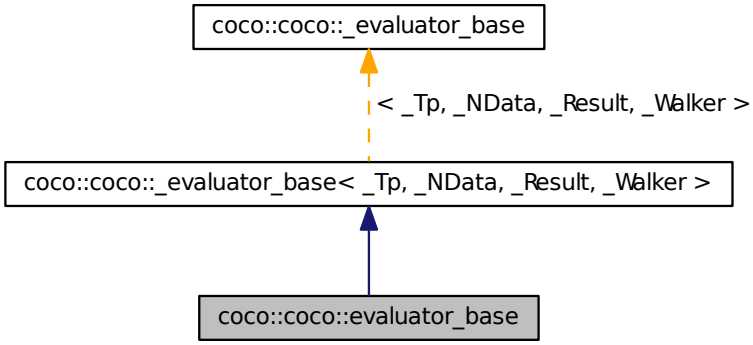
- evaluator.h

10.106 coco::coco::evaluator_base Class Reference

Base class of all (non-caching) evaluators.

Inheritance diagram for coco::coco::evaluator_base:

Collaboration diagram for coco::coco::evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker
- typedef _Tp data_type

Public Member Functions

- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)
- virtual return_value vvalue ()
- virtual return_value value ()
- virtual int vcollect (const return_value &__cresult)
- virtual int collect (const node_data_type &__data, const return_value &__cresult)
- virtual void postorder (const node_data_type &__data)

Protected Attributes

- _Tp eval_data

10.106.1 Detailed Description

This class is the base class of all non-caching evaluators. Basically, it is a visitor to expression_node nodes in a DAG, based on _evaluator_base.

10.106.2 Member Typedef Documentation

10.106.2.1 typedef _Base::const_walker coco::coco::evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::coco::backward_evaluator_base, coco::backward_evaluator_base, coco::coco::forward_evaluator_base, and coco::forward_evaluator_base.

Definition at line 391 of file search_graph.cc.

10.106.2.2 typedef _Tp coco::coco::_evaluator_base::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 306 of file search_graph.cc.

10.106.2.3 typedef _Base::node_data_type coco::coco::evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::coco::backward_evaluator_base, coco::backward_evaluator_base, coco::coco::forward_evaluator_base, and coco::forward_evaluator_base.

Definition at line 387 of file search_graph.cc.

10.106.2.4 typedef _Base::return_value coco::coco::evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::coco::backward_evaluator_base, coco::backward_evaluator_base, coco::coco::forward_evaluator_base, and coco::forward_evaluator_base.

Definition at line 389 of file search_graph.cc.

10.106.3 Member Function Documentation

10.106.3.1 virtual int coco::coco::_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__cresult*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 360 of file search_graph.cc.

10.106.3.2 virtual void coco::coco::_evaluator_base::postorder (const node_data_type & *__data*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited.

Definition at line 365 of file search_graph.cc.

10.106.3.3 virtual int coco::coco::evaluator_base::preorder (const node_data_type & *__data*) [inline, virtual]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 402 of file search_graph.cc.

10.106.3.4 virtual const_walker coco::coco::evaluator_base::short_cut_to (const node_data_type & *__data*) [inline, virtual]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 406 of file search_graph.cc.

10.106.3.5 virtual return_value coco::coco::_evaluator_base::value ( ) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value.

Definition at line 337 of file search_graph.cc.

10.106.3.6 virtual int coco::coco::_evaluator_base::vcollect (const return_value & *__cresult*) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect-:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 349 of file search_graph.cc.

10.106.3.7 virtual return_value coco::coco::_evaluator_base::vvalue ( ) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value.

Definition at line 333 of file search_graph.cc.

10.106.4 Member Data Documentation

10.106.4.1 _Tp coco::coco::_evaluator_base::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 317 of file search_graph.cc.

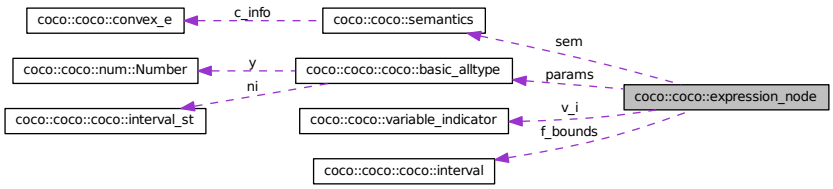
The documentation for this class was generated from the following file:

- evaluator.h

10.107 coco::coco::expression_node Class Reference

The base class for a node in the expression DAGs.

Collaboration diagram for coco::coco::expression_node:



Public Member Functions

- expression_node ()
- expression_node (int et, int nn)
- expression_node (const expression_node &__x)
- virtual ~expression_node ()
- expression_node & operator= (const expression_node &__x)
- bool operator< (const expression_node &__x) const
- void merge (const expression_node &__s)
- void set_bounds (rhs_t __i)
- void set_bounds (double __d=0.)
- void set_bounds (int __i)
- void set_bounds (double lo, double up)
- void add_is_var (unsigned int idx)
- void rm_is_var (unsigned int idx)
- bool is (unsigned int __tp) const
- const variable_indicator & var_indicator () const

- virtual double f_evaluate (int argnum, int idx, const std::vector< double > &x, const variable_-indicator &v_i, double fold, double fupd, std::vector< double > *cache_data) const
- virtual interval i_evaluate (int argnum, int idx, const std::vector< interval > &x, const variable_-indicator &v_i, interval fold, interval fupd, std::vector< interval > *cache_data) const
- virtual interval cp_evaluate (int argnum, int idx, const std::vector< interval > &node_range, const variable_indicator &v_i, interval fold, interval fupd, std::vector< interval > *cache_data) const

Public Attributes

- unsigned int node_num

  node number for indexing
- int operator_type
- unsigned int n_parents

  number of parents
- unsigned int n_children

  number of children

- std::vector< double > coeffs

  coefficients of the sub_expressions
- basic_alltype params

  additional expression info
- rhs_t f_bounds

  bounds on this node
- unsigned short is_var

  this node represents is_var variables
- std::vector< unsigned int > var_idx

  the variable indices corresponding to this node
- semantics sem
- variable_indicator v_i

  this node depends on which variables?
- evaluator_v * ev

Friends

- std::ostream & operator<< (std::ostream &o, const expression_node &__x)

10.107.1 Detailed Description

This class is the base class for all nodes in the expression DAGs used to specify the optimization problems to be solved.

10.107.2 Constructor & Destructor Documentation

10.107.2.1 coco::coco::expression_node::expression_node ( ) [inline]

The Standard Constructor

Definition at line 519 of file search_graph.cc.

10.107.2.2 coco::coco::expression_node::expression_node (int *et*, int *nn*) [inline]

Constructor, which builds a node of operator type `et` having node number `nn`.

Definition at line 526 of file search_graph.cc.

10.107.2.3 coco::coco::expression_node::expression_node (const expression_node & *__x*) [inline]

Standard Copy Constructor

Definition at line 533 of file search_graph.cc.

10.107.2.4 virtual coco::coco::expression_node::~expression_node ( ) [inline, virtual]

Standard Destructor

Definition at line 541 of file search_graph.cc.

10.107.3 Member Function Documentation

10.107.3.1 void coco::coco::expression_node::add_is_var (unsigned int *idx*) [inline]

The add_is_var method is used to add the variable number `idx` to the list of variables, which are represented by this node.

Definition at line 620 of file search_graph.cc.

10.107.3.2 virtual interval coco::coco::expression_node::cp_evaluate (int *argnum,* int *idx,* const std::vector< interval > & *node_range,* const variable_indicator & *v_i,* interval *fold,* interval *fupd,* std::vector< interval > * *cache_data*) const [inline, virtual]

This is an extra evaluation routine, which needs to be overloaded by a user defined node (with operator_type > 0).

Definition at line 663 of file search_graph.cc.

10.107.3.3 virtual double coco::coco::expression_node::f_evaluate (int *argnum,* int *idx,* const std::vector< double > & *x,* const variable_indicator & *v_i,* double *fold,* double *fupd,* std::vector< double > * *cache_data*) const [inline, virtual]

This is an extra evaluation routine, which needs to be overloaded by a user defined node (with operator_type > 0).

Definition at line 655 of file search_graph.cc.

10.107.3.4 virtual interval coco::coco::expression_node::i_evaluate (int *argnum,* int *idx,* const std::vector< interval > & *x,* const variable_indicator & *v_i,* interval *fold,* interval *fupd,* std::vector< interval > * *cache_data*) const [inline, virtual]

This is an extra evaluation routine, which needs to be overloaded by a user defined node (with operator_type > 0).

Definition at line 659 of file search_graph.cc.

10.107.3.5 bool coco::expression_node::is (unsigned int *__tp*) const

The method `is` is used to determine, whether the expression represented by this node has a certain type specified by `__tp`. For `__tp` one of the `ex_..` values or a logical combination of them can be used.

Definition at line 39 of file expression.cc.

10.107.3.6 void coco::coco::expression_node::merge (const expression_node & *__s*) [inline]

This method merges the node `__s` with this node. It is used mainly in the simplifier.

Definition at line 580 of file search_graph.cc.

10.107.3.7 bool coco::coco::expression_node::operator< (const expression_node & *__x*) const [inline]

This is the order operator for expression nodes.

This function compares two nodes' node numbers.

Definition at line 279 of file expression.hpp.

10.107.3.8 expression_node& coco::coco::expression_node::operator= (const expression_node & __x) [inline]

Standard Assignment Operator

Definition at line 549 of file search_graph.cc.

10.107.3.9 void coco::coco::expression_node::rm_is_var (unsigned int *idx*) [inline]

The add_is_var method is used to remove the variable number `idx` from the list of variables, which are represented by this node.

Definition at line 628 of file search_graph.cc.

10.107.3.10 void coco::coco::expression_node::set_bounds (rhs_t __i) [inline]

This method sets the node's restriction to the rhs_t `__i`.

Definition at line 593 of file search_graph.cc.

10.107.3.11 void coco::coco::expression_node::set_bounds (double __d = 0.) [inline]

This method sets the node's bounds to the equality restriction "== `__d`".

Definition at line 600 of file search_graph.cc.

10.107.3.12 void coco::coco::expression_node::set_bounds (int __i) [inline]

This method sets the node's bounds to the restriction "in `__i`".

Definition at line 606 of file search_graph.cc.

10.107.3.13 void coco::coco::expression_node::set_bounds (double *lo*, double *up*) [inline]

This method sets the node's bounds to the restriction "in `[lo, up]`".

Definition at line 613 of file search_graph.cc.

10.107.3.14 const variable_indicator& coco::coco::expression_node::var_indicator ( ) const [inline]

This method returns the variable indicator of this node.

Definition at line 650 of file search_graph.cc.

10.107.4 Friends And Related Function Documentation

10.107.4.1 std::ostream& operator<< (std::ostream & *o*, const expression_node & __x) [friend]

10.107.5 Member Data Documentation

10.107.5.1 std::vector<double> coco::coco::expression_node::coeffs

Definition at line 490 of file search_graph.cc.

10.107.5.2 evaluator_v* coco::coco::expression_node::ev

optional evaluators used instead of tree walks NULL means no evaluators, map if some or all are defined. This is for recursive_cached_walk.

Definition at line 505 of file search_graph.cc.

10.107.5.3 rhs_t coco::coco::expression_node::f_bounds

Definition at line 494 of file search_graph.cc.

10.107.5.4 unsigned short coco::coco::expression_node::is_var

Definition at line 496 of file search_graph.cc.

10.107.5.5 unsigned int coco::coco::expression_node::n_children

Definition at line 486 of file search_graph.cc.

10.107.5.6 unsigned int coco::coco::expression_node::n_parents

Definition at line 486 of file search_graph.cc.

10.107.5.7 unsigned int coco::coco::expression_node::node_num

Definition at line 475 of file search_graph.cc.

10.107.5.8 int coco::coco::expression_node::operator_type

this is a number describing the operation. negative numbers describe standard operations as defined above. the positive numbers describe other operations like elementary functions (exp, pow, sin,...) or more complicated functions like linear or general quadratic terms, user-defined functions, piecewise linear approximations,...

Definition at line 477 of file search_graph.cc.

10.107.5.9 basic_alltype coco::coco::expression_node::params

Definition at line 492 of file search_graph.cc.

10.107.5.10 semantics coco::coco::expression_node::sem

additional semantics information like integer (y/n), stochastic (y/n), karush-john (y/n) convexity,...

Definition at line 499 of file search_graph.cc.

10.107.5.11 variable_indicator coco::coco::expression_node::v_i

Definition at line 503 of file search_graph.cc.

10.107.5.12 std::vector<unsigned int> coco::coco::expression_node::var_idx

Definition at line 497 of file search_graph.cc.

The documentation for this class was generated from the following files:

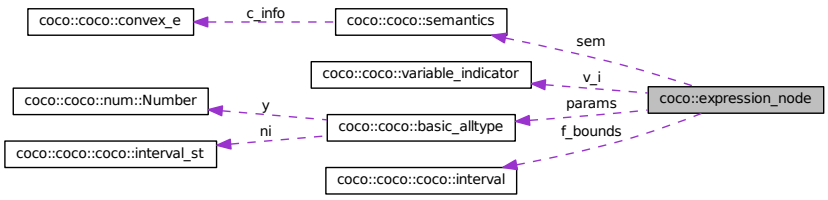
- expression.h
- expression.cc
- expression.hpp

10.108 coco::expression_node Class Reference

The base class for a node in the expression DAGs.

`#include <expression.h>`

Collaboration diagram for coco::expression_node:



Classes

- class children_compare
- class parents_compare
- class parents_compare_eq

Public Member Functions

- expression_node ()
- expression_node (int et, int nn)
- expression_node (const expression_node &__x)
- virtual ~expression_node ()
- expression_node & operator= (const expression_node &__x)
- bool operator< (const expression_node &__x) const
- void merge (const expression_node &__s)
- void set_bounds (rhs_t __i)
- void set_bounds (double __d=0.)
- void set_bounds (int __i)
- void set_bounds (double lo, double up)
- void add_is_var (unsigned int idx)

- void rm_is_var (unsigned int idx)
- bool is (unsigned int __tp) const
- const variable_indicator & var_indicator () const

- virtual double f_evaluate (int argnum, int idx, const std::vector< double > &x, const variable_indicator &v_i, double fold, double fupd, std::vector< double > *cache_data) const
- virtual interval i_evaluate (int argnum, int idx, const std::vector< interval > &x, const variable_indicator &v_i, interval fold, interval fupd, std::vector< interval > *cache_data) const
- virtual interval cp_evaluate (int argnum, int idx, const std::vector< interval > &node_range, const variable_indicator &v_i, interval fold, interval fupd, std::vector< interval > *cache_data) const

Public Attributes

- unsigned int node_num

  node number for indexing
- int operator_type
- unsigned int n_parents

  number of parents
- unsigned int n_children

  number of children
- std::vector< double > coeffs

  coefficients of the sub_expressions
- basic_alltype params

  additional expression info
- rhs_t f_bounds

  bounds on this node
- unsigned short is_var

  this node represents is_var variables
- std::vector< unsigned int > var_idx

  the variable indices corresponding to this node
- semantics sem
- variable_indicator v_i

  this node depends on which variables?
- evaluator_v * ev

Friends

- std::ostream & operator<< (std::ostream &o, const expression_node &__x)

10.108.1 Detailed Description

This class is the base class for all nodes in the expression DAGs used to specify the optimization problems to be solved.

10.108.2 Constructor & Destructor Documentation

10.108.2.1 coco::expression_node::expression_node ( ) [inline]

The Standard Constructor

Definition at line 519 of file expression.h.

10.108.2.2 coco::expression_node::expression_node (int *et*, int *nn*) [inline]

Constructor, which builds a node of operator type `et` having node number `nn`.

Definition at line 526 of file expression.h.

10.108.2.3 coco::expression_node::expression_node (const expression_node & *__x*) [inline]

Standard Copy Constructor

Definition at line 533 of file expression.h.

10.108.2.4 virtual coco::expression_node::~expression_node ( ) [inline, virtual]

Standard Destructor

Definition at line 541 of file expression.h.

10.108.3 Member Function Documentation

10.108.3.1 void coco::expression_node::add_is_var (unsigned int *idx*) [inline]

The add_is_var method is used to add the variable number `idx` to the list of variables, which are represented by this node.

Definition at line 620 of file expression.h.

10.108.3.2 virtual interval coco::expression_node::cp_evaluate (int *argnum*, int *idx*, const std::vector< interval > & *node_range*, const variable_indicator & *v_i*, interval *fold*, interval *fupd*, std::vector< interval > * *cache_data*) const [inline, virtual]

This is an extra evaluation routine, which needs to be overloaded by a user defined node (with operator_type > 0).

Definition at line 663 of file expression.h.

10.108.3.3 virtual double coco::expression_node::f_evaluate (int *argnum*, int *idx*, const std::vector< double > & *x*, const variable_indicator & *v_i*, double *fold*, double *fupd*, std::vector< double > * *cache_data*) const [inline, virtual]

This is an extra evaluation routine, which needs to be overloaded by a user defined node (with operator_type > 0).

Definition at line 655 of file expression.h.

10.108.3.4 virtual interval coco::expression_node::i_evaluate (int *argnum*, int *idx*, const std::vector< interval > & *x*, const variable_indicator & *v_i*, interval *fold*, interval *fupd*, std::vector< interval > * *cache_data*) const [inline, virtual]

This is an extra evaluation routine, which needs to be overloaded by a user defined node (with operator_type > 0).

Definition at line 659 of file expression.h.

10.108.3.5 bool coco::expression_node::is (unsigned int *__tp*) const

The method `is` is used to determine, whether the expression represented by this node has a certain type specified by `__tp`. For `__tp` one of the `ex_..` values or a logical combination of them can be used.

10.108.3.6 void coco::expression_node::merge (const expression_node & *__s*) [inline]

This method merges the node `__s` with this node. It is used mainly in the simplifier.

Definition at line 580 of file expression.h.

10.108.3.7 bool coco::expression_node::operator< (const expression_node & *__x*) const

This is the order operator for expression nodes.

10.108.3.8 expression_node& coco::expression_node::operator= (const expression_node & *__x*) [inline]

Standard Assignment Operator

Definition at line 549 of file expression.h.

10.108.3.9 void coco::expression_node::rm_is_var (unsigned int *idx*) [inline]

The add_is_var method is used to remove the variable number `idx` from the list of variables, which are represented by this node.

Definition at line 628 of file expression.h.

10.108.3.10 void coco::expression_node::set_bounds (rhs_t *__i*) [inline]

This method sets the node's restriction to the rhs_t `__i`.

Definition at line 593 of file expression.h.

10.108.3.11 void coco::expression_node::set_bounds (double *__d* = `0.`) [inline]

This method sets the node's bounds to the equality restriction "== `__d`".

Definition at line 600 of file expression.h.

10.108.3.12 void coco::expression_node::set_bounds (int *__i*) [inline]

This method sets the node's bounds to the restriction "in `__i`".

Definition at line 606 of file expression.h.

10.108.3.13 void coco::expression_node::set_bounds (double *lo*, double *up*) [inline]

This method sets the node's bounds to the restriction "in [lo, up]".

Definition at line 613 of file expression.h.

10.108.3.14 const variable_indicator& coco::expression_node::var_indicator ( ) const [inline]

This method returns the variable indicator of this node.

Definition at line 650 of file expression.h.

10.108.4 Friends And Related Function Documentation

10.108.4.1 std::ostream& operator<< (std::ostream & *o*, const expression_node & *__x*) [friend]

Definition at line 208 of file expression.cc.

10.108.5 Member Data Documentation

10.108.5.1 std::vector<double> coco::expression_node::coeffs

Definition at line 490 of file expression.h.

10.108.5.2 evaluator_v* coco::expression_node::ev

optional evaluators used instead of tree walks NULL means no evaluators, map if some or all are defined. This is for recursive_cached_walk.

Definition at line 505 of file expression.h.

10.108.5.3 rhs_t coco::expression_node::f_bounds

Definition at line 494 of file expression.h.

10.108.5.4 unsigned short coco::expression_node::is_var

Definition at line 496 of file expression.h.

10.108.5.5 unsigned int coco::expression_node::n_children

Definition at line 486 of file expression.h.

10.108.5.6 unsigned int coco::expression_node::n_parents

Definition at line 486 of file expression.h.

10.108.5.7 unsigned int coco::expression_node::node_num

Definition at line 475 of file expression.h.

10.108.5.8 int coco::expression_node::operator_type

this is a number describing the operation. negative numbers describe standard operations as defined above. the positive numbers describe other operations like elementary functions (exp, pow, sin,...) or more complicated functions like linear or general quadratic terms, user-defined functions, piecewise linear approximations,...

Definition at line 477 of file expression.h.

10.108.5.9 basic_alltype coco::expression_node::params

Definition at line 492 of file expression.h.

10.108.5.10 semantics coco::expression_node::sem

additional semantics information like integer (y/n), stochastic (y/n), karush-john (y/n) convexity,...

Definition at line 499 of file expression.h.

10.108.5.11 variable_indicator coco::expression_node::v_i

Definition at line 503 of file expression.h.

10.108.5.12 std::vector<unsigned int> coco::expression_node::var_idx

Definition at line 497 of file expression.h.

The documentation for this class was generated from the following file:

- expression.h

10.109 coco::expression_print_visitor Class Reference

`#include <expression.hpp>`

Public Member Functions

- expression_print_visitor (std::vector< bool > &__p, std::ostream &__o=std::cout)
- expression_print_visitor (const expression_print_visitor &__p)
- ~expression_print_visitor ()

- bool preorder (const expression_node &r)
- int vvalue ()
- int value ()
- void collect (const expression_node &r, int __r)

10.109.1 Detailed Description

This class is a visitor to expression_node nodes in a DAG, which can be used to print the DAG in .dag format.

10.109.2 Constructor & Destructor Documentation

10.109.2.1 coco::expression_print_visitor::expression_print_visitor (std::vector< bool > & __p, std::ostream & __o = std::cout) [inline]

Constructor, which initializes printed from __p and sets the ostream to __o (default is cout).

Definition at line 302 of file expression.hpp.

10.109.2.2 coco::expression_print_visitor::expression_print_visitor (const expression_print_visitor & __p) [inline]

Standard Copy Constructor

Definition at line 306 of file expression.hpp.

10.109.2.3 coco::expression_print_visitor::~expression_print_visitor ( ) [inline]

Standard Destructor

Definition at line 309 of file expression.hpp.

10.109.3 Member Function Documentation

10.109.3.1 void coco::expression_print_visitor::collect (const expression_node & r, int __r) [inline]

This function is needed by a VGTL preorder visitor.

Definition at line 330 of file expression.hpp.

10.109.3.2 bool coco::expression_print_visitor::preorder (const expression_node & r) [inline]

This function is needed by a VGTL preorder visitor.

Definition at line 313 of file expression.hpp.

10.109.3.3 int coco::expression_print_visitor::value ( ) [inline]

This function is needed by a VGTL preorder visitor.

Definition at line 329 of file expression.hpp.

10.109.3.4 int coco::expression_print_visitor::vvalue ( ) [inline]

This function is needed by a VGTL preorder visitor.

Definition at line 328 of file expression.hpp.

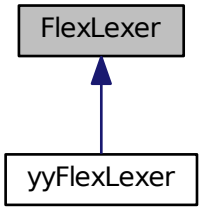
The documentation for this class was generated from the following file:

- expression.hpp

10.110 FlexLexer Class Reference

`#include <FlexLexer.h>`

Inheritance diagram for FlexLexer:



Public Member Functions

- virtual ~FlexLexer ()
- const char * YYText ()
- int YYLeng ()
- virtual void yy_switch_to_buffer (struct yy_buffer_state *new_buffer)=0
- virtual struct yy_buffer_state * yy_create_buffer (FLEX_STD istream *s, int size)=0
- virtual void yy_delete_buffer (struct yy_buffer_state *b)=0
- virtual void yyrestart (FLEX_STD istream *s)=0
- virtual int yylex ()=0
- int yylex (FLEX_STD istream *new_in, FLEX_STD ostream *new_out=0)
- virtual void switch_streams (FLEX_STD istream *new_in=0, FLEX_STD ostream *new_out=0)=0
- int lineno () const
- int debug () const
- void set_debug (int flag)

Protected Attributes

- char * yytext
- int yyleng
- int yylineno
- int yy_flex_debug

10.110.1 Constructor & Destructor Documentation

10.110.1.1 virtual FlexLexer::~FlexLexer ( ) [inline, virtual]

Definition at line 63 of file FlexLexer.h.

10.110.2 Member Function Documentation

10.110.2.1 int FlexLexer::debug ( ) const [inline]

Definition at line 91 of file FlexLexer.h.

10.110.2.2 int FlexLexer::lineno ( ) const [inline]

Definition at line 89 of file FlexLexer.h.

10.110.2.3 void FlexLexer::set_debug (int *flag*) [inline]

Definition at line 92 of file FlexLexer.h.

10.110.2.4 virtual void FlexLexer::switch_streams (FLEX_STD istream * *new_in* = 0, FLEX_STD ostream * *new_out* = 0) [pure virtual]

Implemented in yyFlexLexer.

10.110.2.5 virtual struct yy_buffer_state* FlexLexer::yy_create_buffer (FLEX_STD istream * *s*, int *size*) [read, pure virtual]

Implemented in yyFlexLexer.

10.110.2.6 virtual void FlexLexer::yy_delete_buffer (struct yy_buffer_state * *b*) [pure virtual]

Implemented in yyFlexLexer.

10.110.2.7 virtual void FlexLexer::yy_switch_to_buffer (struct yy_buffer_state * *new_buffer*) [pure virtual]

Implemented in yyFlexLexer.

10.110.2.8 int FlexLexer::YYLeng ( ) [inline]

Definition at line 66 of file FlexLexer.h.

10.110.2.9 virtual int FlexLexer::yylex ( ) [pure virtual]

Implemented in yyFlexLexer.

10.110.2.10 int FlexLexer::yylex (FLEX_STD istream * *new_in*, FLEX_STD ostream * *new_out* = 0) [inline]

Definition at line 78 of file FlexLexer.h.

10.110.2.11 virtual void FlexLexer::yyrestart (FLEX_STD istream * *s*) [pure virtual]

Implemented in yyFlexLexer.

10.110.2.12 const char* FlexLexer::YYText ( ) [inline]

Definition at line 65 of file FlexLexer.h.

10.110.3 Member Data Documentation

10.110.3.1 int FlexLexer::yy_flex_debug [protected]

Definition at line 98 of file FlexLexer.h.

10.110.3.2 int FlexLexer::yyleng [protected]

Definition at line 96 of file FlexLexer.h.

10.110.3.3 int FlexLexer::yylineno [protected]

Definition at line 97 of file FlexLexer.h.

10.110.3.4 char* FlexLexer::yytext [protected]

Definition at line 95 of file FlexLexer.h.

The documentation for this class was generated from the following file:

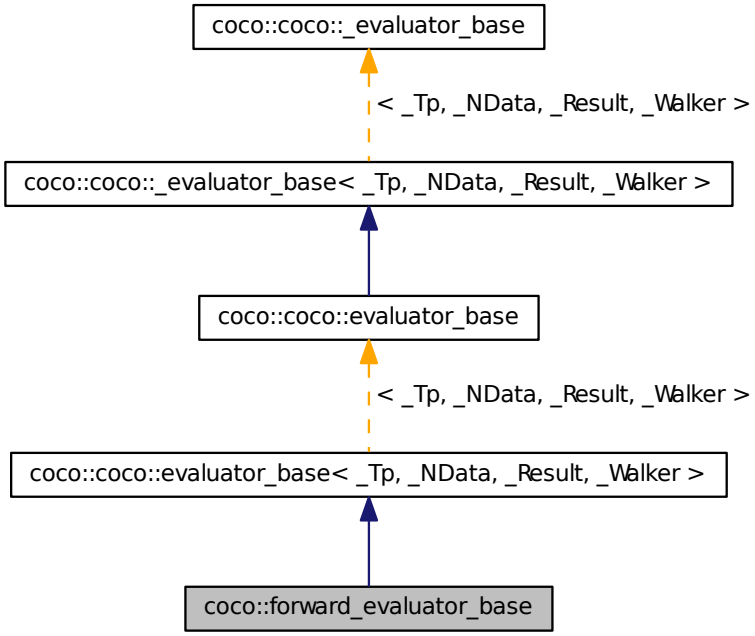
- FlexLexer.h

10.111 coco::forward_evaluator_base Class Reference

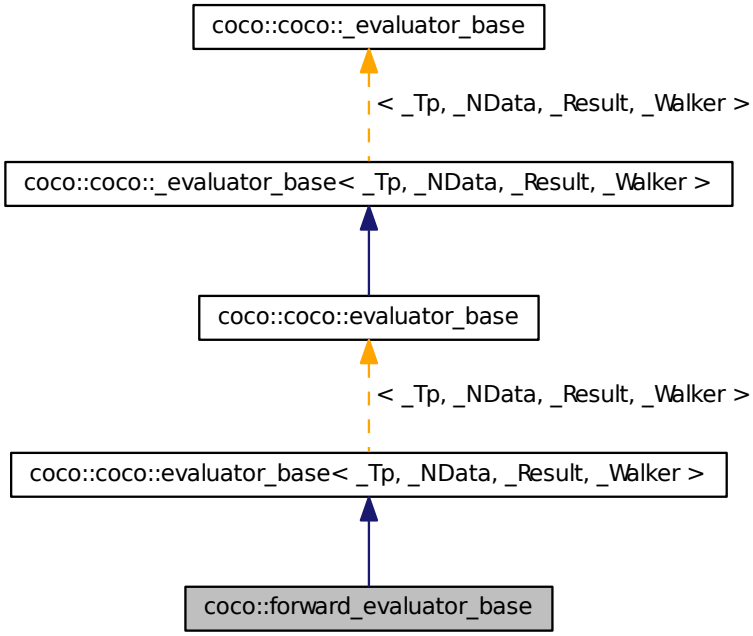
Base class of all (non-caching) forward evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::forward_evaluator_base:



Collaboration diagram for coco::forward_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual void initialize ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const return_value &__rval)

- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual return_value calculate_value (bool eval_all)
- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)

10.111.1 Detailed Description

This class is the base class of all non-caching forward evaluators. Basically, it is a visitor to expression_-node nodes in a DAG, based on evaluator_base.

10.111.2 Member Typedef Documentation

10.111.2.1 typedef _Base::const_walker coco::forward_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 489 of file evaluator.h.

10.111.2.2 typedef _Base::node_data_type coco::forward_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 485 of file evaluator.h.

10.111.2.3 typedef _Base::return_value coco::forward_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 487 of file evaluator.h.

10.111.3 Member Function Documentation

10.111.3.1 virtual void coco::forward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual]

This method is called right after all children of a node have been visited. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 547 of file evaluator.h.

10.111.3.2 virtual return_value coco::forward_evaluator_base::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Definition at line 577 of file evaluator.h.

10.111.3.3 virtual void coco::forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 551 of file evaluator.h.

10.111.3.4 int coco::forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 506 of file evaluator.h.

10.111.3.5 virtual void coco::forward_evaluator_base::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Definition at line 532 of file evaluator.h.

10.111.3.6 virtual int coco::forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 543 of file evaluator.h.

10.111.3.7 void coco::forward_evaluator_base::postorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 500 of file evaluator.h.

10.111.3.8 virtual int coco::coco::evaluator_base::preorder (const node_data_type & __data) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 402 of file search_graph.cc.

10.111.3.9 int coco::forward_evaluator_base::preorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It is translated into a call to initialize.

Definition at line 495 of file evaluator.h.

10.111.3.10 virtual const_walker coco::coco::evaluator_base::short_cut_to (const node_data_type & __data) [inline, virtual, inherited]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 406 of file search_graph.cc.

10.111.3.11 virtual int coco::forward_evaluator_base::update (const return_value & __rval) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 561 of file evaluator.h.

10.111.3.12 virtual int coco::forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 572 of file evaluator.h.

10.111.3.13 return_value coco::forward_evaluator_base::value ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 518 of file evaluator.h.

10.111.3.14 int coco::forward_evaluator_base::vcollect (const return_value & __rval) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 512 of file evaluator.h.

10.111.3.15 void coco::forward_evaluator_base::vinit ( ) [inline]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 527 of file evaluator.h.

10.111.3.16 return_value coco::forward_evaluator_base::vvalue ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 523 of file evaluator.h.

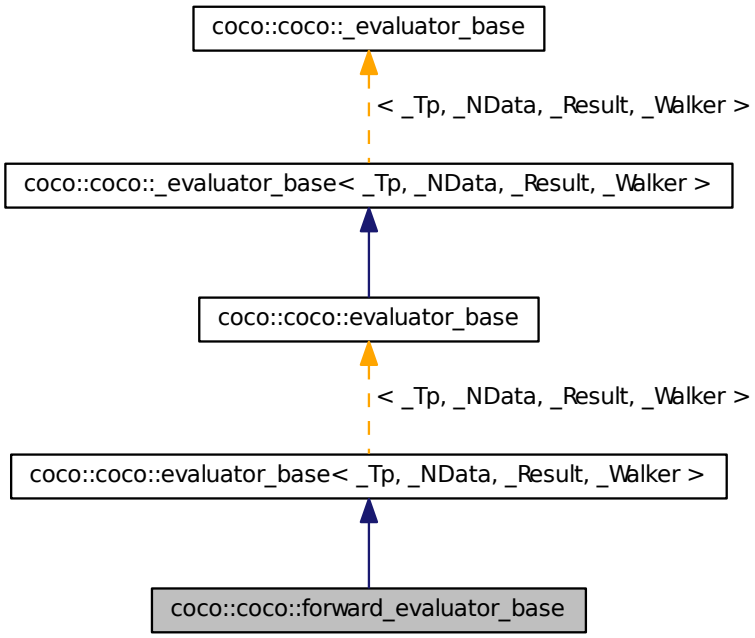
The documentation for this class was generated from the following file:

- evaluator.h

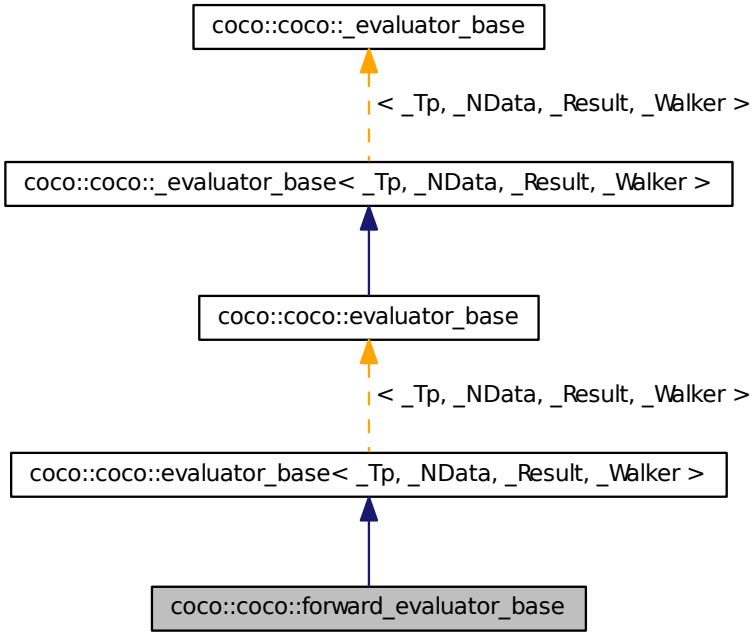
10.112 coco::coco::forward_evaluator_base Class Reference

Base class of all (non-caching) forward evaluators.

Inheritance diagram for coco::coco::forward_evaluator_base:



Collaboration diagram for coco::coco::forward_evaluator_base:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual void initialize ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const return_value &__rval)

- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual return_value calculate_value (bool eval_all)
- virtual int preorder (const node_data_type &__data)
- virtual const_walker short_cut_to (const node_data_type &__data)

10.112.1 Detailed Description

This class is the base class of all non-caching forward evaluators. Basically, it is a visitor to expression_-node nodes in a DAG, based on evaluator_base.

10.112.2 Member Typedef Documentation

10.112.2.1 typedef _Base::const_walker coco::coco::forward_evaluator_base::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 490 of file search_graph.cc.

10.112.2.2 typedef _Base::node_data_type coco::coco::forward_evaluator_base::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 486 of file search_graph.cc.

10.112.2.3 typedef _Base::return_value coco::coco::forward_evaluator_base::return_value

This type is the result type of the evaluator.

Reimplemented from coco::coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 488 of file search_graph.cc.

10.112.3 Member Function Documentation

10.112.3.1 virtual void coco::coco::forward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual]

This method is called right after all children of a node have been visited. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 548 of file search_graph.cc.

10.112.3.2 virtual return_value coco::coco::forward_evaluator_base::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Definition at line 578 of file search_graph.cc.

10.112.3.3 virtual void coco::coco::forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 552 of file search_graph.cc.

10.112.3.4 int coco::coco::forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 507 of file search_graph.cc.

10.112.3.5 virtual void coco::coco::forward_evaluator_base::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Definition at line 533 of file search_graph.cc.

10.112.3.6 virtual int coco::coco::forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 544 of file search_graph.cc.

10.112.3.7 void coco::coco::forward_evaluator_base::postorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 501 of file search_graph.cc.

10.112.3.8 virtual int coco::coco::evaluator_base::preorder (const node_data_type & __data) [inline, virtual, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. The return value of the method influences the further graph walk:

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 402 of file search_graph.cc.

10.112.3.9 int coco::coco::forward_evaluator_base::preorder (const node_data_type & __data) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It is translated into a call to initialize.

Definition at line 496 of file search_graph.cc.

10.112.3.10 virtual const_walker coco::coco::evaluator_base::short_cut_to (const node_data_type & __data) [inline, virtual, inherited]

The short_cut_to method is called whenever a short-cut is signalled during the graph walk, and the `const_walker` returned is the target of the short-cut.

Definition at line 406 of file search_graph.cc.

10.112.3.11 virtual int coco::coco::forward_evaluator_base::update (const return_value & __rval) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 562 of file search_graph.cc.

10.112.3.12 virtual int coco::coco::forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 573 of file search_graph.cc.

10.112.3.13 return_value coco::coco::forward_evaluator_base::value ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 519 of file search_graph.cc.

10.112.3.14 int coco::coco::forward_evaluator_base::vcollect (const return_value & *__rval*) [inline]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 513 of file search_graph.cc.

10.112.3.15 void coco::coco::forward_evaluator_base::vinit ( ) [inline]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 528 of file search_graph.cc.

10.112.3.16 return_value coco::coco::forward_evaluator_base::vvalue ( ) [inline]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 524 of file search_graph.cc.

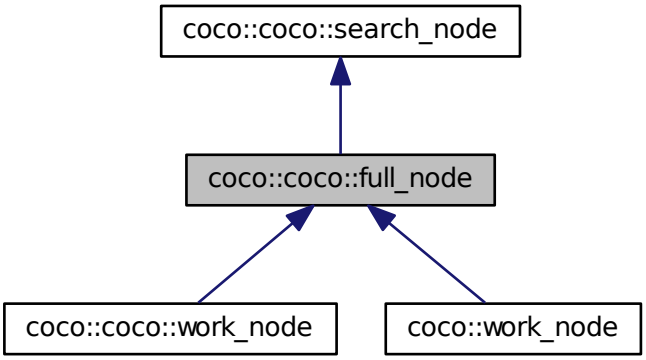
The documentation for this class was generated from the following file:

- evaluator.h

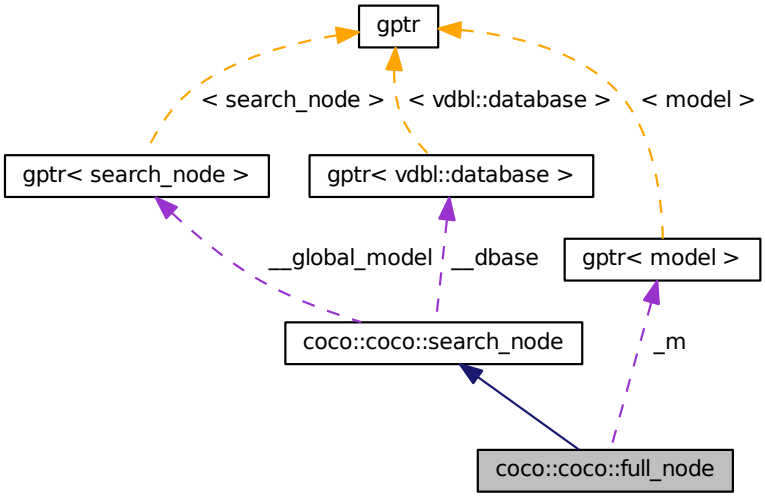
10.113 coco::coco::full_node Class Reference

Class holding the full nodes in the search graph.

Inheritance diagram for coco::coco::full_node:



Collaboration diagram for coco::coco::full_node:



Public Member Functions

- full_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__mod, gptr< search_node > &_gm, gptr< vdbl::database > &_db, search_node_relation _snr=snr_reduction)
- full_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__mod, gptr< search_node > &_gm, gptr< vdbl::database > &_db, const std::vector< annotation > &_a, search_node_relation _snr=snr_reduction)
- full_node (const full_node &_i)
- virtual ~full_node ()
- full_node & operator= (const full_node &__w)
- bool is_delta () const
- unsigned int n_annotations () const
- const annotation & get_annotation (unsigned int i) const
- const std::vector< annotation > & get_annotations () const
- const model * get_model () const
- const vdbl::database * get_database () const
- model * get_model_ptr () const
- vdbl::database * get_database_ptr () const
- vdbl::userid get_dbuserid () const
- gptr< search_node > * global_model () const
- gptr< vdbl::database > * database () const
- search_node_id get_id () const
- void set_id (const search_node_id &i)
- vdbl::rowid get_rowid () const
- void set_rowid (const vdbl::rowid &i)

- void keep (const annotation &_an)
- void keep (const std::vector< annotation > &_anv)
- void unkeep (const annotation &_an)
- void unkeep (const std::vector< annotation > &_anv)

Public Attributes

- std::vector< annotation > _ann

Protected Member Functions

- full_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__mod, gptr< search_node > *_gm, gptr< vdbl::database > &_db, search_node_relation _snr=snr_reduction)
- full_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__mod, gptr< search_node > *_gm, gptr< vdbl::database > &_db, const std::vector< annotation > &_a, search_node_relation _snr=snr_reduction)
- search_node_relation parent_relation () const

Protected Attributes

- gptr< model > * _m
- gptr< search_node > * __global_model
- gptr< vdbl::database > * __dbase
- vdbl::userid _dbuser
- search_node_relation _snr
- search_node_id _id
- std::vector< annotation > _keep
- vdbl::rowid _rid

Friends

- class delta_base
- class certificate_base
- class dag_delta
- class dag_undelta
- class search_graph

10.113.1 Detailed Description

This is a class of nodes stored in the search graph. It represents a full node, i.e., a node which stores a complete model not just changes to the parent model.

10.113.2 Constructor & Destructor Documentation

10.113.2.1 coco::coco::full_node::full_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __mod, gptr< search_node > * _gm, gptr< vdbl::database > & _db, search_node_relation _snr = snr_reduction) [inline, protected]

This constructor generates a new full node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameter ___mod is used to specify the full model, while the parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized as empty vector.

Definition at line 292 of file search_graph.cc.

10.113.2.2 coco::coco::full_node::full_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __mod, gptr< search_node > * _gm, gptr< vdbl::database > & _db, const std::vector< annotation > & _a, search_node_relation _snr = snr_reduction) [inline, protected]

This constructor generates a new full node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameter ___mod is used to specify the full model, while the parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized by the parameter _a.

Definition at line 305 of file search_graph.cc.

10.113.2.3 coco::coco::full_node::full_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __mod, gptr< search_node > & _gm, gptr< vdbl::database > & _db, search_node_relation _snr = snr_reduction) [inline]

This constructor generates a new full node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameter ___mod is used to specify the full model, while the parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized as empty vector.

Definition at line 320 of file search_graph.cc.

10.113.2.4 coco::coco::full_node::full_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __mod, gptr< search_node > & _gm, gptr< vdbl::database > & _db, const std::vector< annotation > & _a, search_node_relation _snr = snr_reduction) [inline]

This constructor generates a new full node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameter ___mod is used to specify the full model, while the parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized by the parameter _a.

Definition at line 333 of file search_graph.cc.

10.113.2.5 coco::coco::full_node::full_node (const full_node & _i) [inline]

Standard copy constructor

Definition at line 341 of file search_graph.cc.

10.113.2.6 virtual coco::coco::full_node::~full_node ( ) [inline, virtual]

Standard Destructor

Definition at line 346 of file search_graph.cc.

10.113.3 Member Function Documentation

10.113.3.1 gptr<vdbl::database>* coco::coco::search_node::database ( ) const [inline, inherited]

This is the accessor method for the search database.

Definition at line 157 of file search_graph.cc.

10.113.3.2 const annotation& coco::coco::full_node::get_annotation (unsigned int *i*) const [inline]

A call to this method returns a const reference to the `i`th annotation.

Definition at line 362 of file search_graph.cc.

10.113.3.3 const std::vector<annotation>& coco::coco::full_node::get_annotations ( ) const [inline]

A call to this method returns a const reference to the whole vector of annotations.

Definition at line 367 of file search_graph.cc.

10.113.3.4 const vdbl::database* coco::coco::full_node::get_database ( ) const [inline]

This method returns a const pointer pointing to a locally stored copy of the search database.

Bug This method needs to be replaced, once the COCONUT environment really gets distributed.

Definition at line 378 of file search_graph.cc.

10.113.3.5 vdbl::database* coco::coco::full_node::get_database_ptr ( ) const [inline]

This method returns a const pointer pointing to a locally stored copy of the search database.

Bug This method needs to be replaced, once the COCONUT environment really gets distributed.

Definition at line 389 of file search_graph.cc.

10.113.3.6 vdbl::userid coco::coco::search_node::get_dbuserid ( ) const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 151 of file search_graph.cc.

10.113.3.7 search_node_id coco::coco::search_node::get_id ( ) const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 160 of file search_graph.cc.

10.113.3.8 const model* coco::coco::full_node::get_model ( ) const [inline]

This method returns a const pointer pointing to a locally stored copy of the model stored in this full node.

Reimplemented in coco::coco::work_node, and coco::work_node.

Definition at line 372 of file search_graph.cc.

10.113.3.9 model* coco::coco::full_node::get_model_ptr ( ) const [inline]

This method returns a pointer pointing to a locally stored copy of the model stored in this full node.

Definition at line 383 of file search_graph.cc.

10.113.3.10 vdbl::rowid coco::coco::search_node::get_rowid ( ) const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 166 of file search_graph.cc.

10.113.3.11 gptr<search_node>* coco::coco::search_node::global_model ( ) const [inline, inherited]

This is the accessor method for the global model.

Definition at line 154 of file search_graph.cc.

10.113.3.12 bool coco::coco::full_node::is_delta ( ) const [inline, virtual]

This method is called to determine whether a search node stores only deltas (delta_node) or a full model (full_node). In case of a full_node it returns `false`.

Reimplemented from coco::coco::search_node.

Definition at line 354 of file search_graph.cc.

10.113.3.13 void coco::coco::search_node::keep (const annotation & _an) [inline, inherited]

A call to this method informs the search node to be the keeper of the annotation _an.

Definition at line 173 of file search_graph.cc.

10.113.3.14 void coco::coco::search_node::keep (const std::vector< annotation > & _anv) [inline, inherited]

A call to this method informs the search node to be the keeper of all the annotations in _anv.

Definition at line 177 of file search_graph.cc.

10.113.3.15 unsigned int coco::coco::full_node::n_annotations ( ) const [inline]

This method returns the number of annotations stored in this full node.

Definition at line 358 of file search_graph.cc.

10.113.3.16 full_node & coco::full_node::operator= (const full_node & *__w*) [inline]

Standard Assignment Operator

Reimplemented in coco::coco::work_node, and coco::work_node.

Definition at line 208 of file search_node.hpp.

10.113.3.17 search_node_relation coco::coco::search_node::parent_relation ( ) const [inline, protected, inherited]

A call to this method returns the relation of this node to its parent in the search graph.

Definition at line 110 of file search_graph.cc.

10.113.3.18 void coco::coco::search_node::set_id (const search_node_id & *i*) [inline, inherited]

This is the set method for the search node id of this node.

Definition at line 163 of file search_graph.cc.

10.113.3.19 void coco::coco::search_node::set_rowid (const vdbl::rowid & *i*) [inline, inherited]

This is the set method for the row id of this node.

Definition at line 169 of file search_graph.cc.

10.113.3.20 void coco::search_node::unkeep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of the annotation _an.

Definition at line 82 of file search_node.hpp.

10.113.3.21 void coco::search_node::unkeep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of all the annotations in _anv.

Definition at line 93 of file search_node.hpp.

10.113.4 Friends And Related Function Documentation

10.113.4.1 friend class certificate_base [friend]

Definition at line 393 of file search_graph.cc.

10.113.4.2 friend class dag_delta [friend]

Definition at line 394 of file search_graph.cc.

10.113.4.3 friend class dag_undelta [friend]

Definition at line 395 of file search_graph.cc.

10.113.4.4 friend class delta_base [friend]

Definition at line 392 of file search_graph.cc.

10.113.4.5 friend class search_graph [friend, inherited]

Definition at line 188 of file search_graph.cc.

10.113.5 Member Data Documentation

10.113.5.1 gptr<vdbl::database>* coco::coco::search_node::__dbase [protected, inherited]

This is a pointer to a global pointer to the search database associated to this search graph.

Definition at line 89 of file search_graph.cc.

10.113.5.2 gptr<search_node>* coco::coco::search_node::__global_model [protected, inherited]

This is a pointer to a global pointer to the top of the search graph. It is NULL for a standalone node or the top of the graph.

Definition at line 86 of file search_graph.cc.

10.113.5.3 std::vector<annotation> coco::coco::full_node::_ann

This is the vector of annotations active at this full node.

Definition at line 283 of file search_graph.cc.

10.113.5.4 vdbl::userid coco::coco::search_node::_dbuser [protected, inherited]

This member stores the database user id under which this search is performed. For the in-memory database this is irrelevant.

Definition at line 92 of file search_graph.cc.

10.113.5.5 search_node_id coco::coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 97 of file search_graph.cc.

10.113.5.6 std::vector<annotation> coco::coco::search_node::_keep [protected, inherited]

The _keep member is a list of all those annotations which are kept by this search node. If the node is destroyed all kept annotations are removed from the search database.

Definition at line 101 of file search_graph.cc.

10.113.5.7 gptr<model>* coco::coco::full_node::_m [protected]

This is a pointer to a global pointer to the model kept in this full node.

Definition at line 279 of file search_graph.cc.

10.113.5.8 vdbl::rowid coco::coco::search_node::_rid [protected, inherited]

This row id specifies the row in the search info table where the hook information for this search_node is stored

Definition at line 105 of file search_graph.cc.

10.113.5.9 search_node_relation coco::coco::search_node::_snr [protected, inherited]

This member specifies the relation of the search node to its parent(s) in the search graph.

Definition at line 95 of file search_graph.cc.

The documentation for this class was generated from the following files:

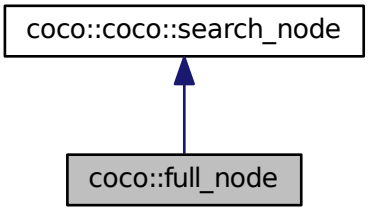
- search_node.h
- search_node.hpp

10.114 coco::full_node Class Reference

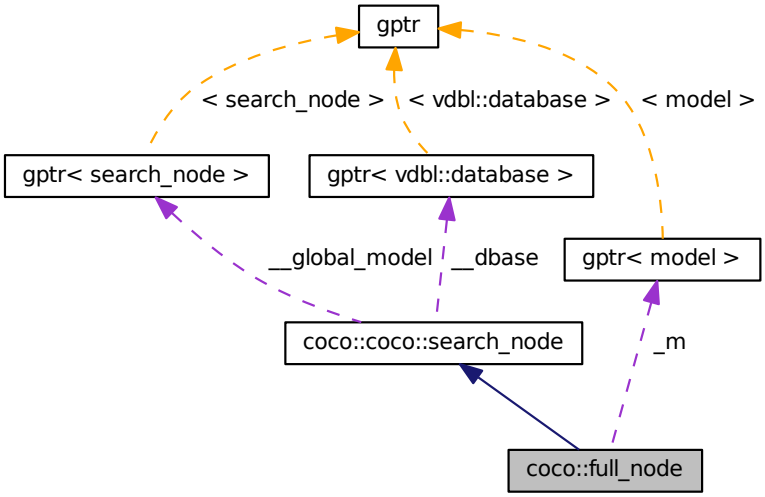
Class holding the full nodes in the search graph.

`#include <search_node.h>`

Inheritance diagram for coco::full_node:



Collaboration diagram for coco::full_node:



Public Member Functions

- full_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__mod, gptr< search_node > &_gm, gptr< vdbl::database > &_db, search_node_relation _snr=snr_reduction)
- full_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__mod, gptr< search_node > &_gm, gptr< vdbl::database > &_db, const std::vector< annotation > &_a, search_node_relation _snr=snr_reduction)
- full_node (const full_node &_i)
- virtual ~full_node ()
- full_node & operator= (const full_node &__w)
- bool is_delta () const
- unsigned int n_annotations () const
- const annotation & get_annotation (unsigned int i) const
- const std::vector< annotation > & get_annotations () const
- const model * get_model () const
- const vdbl::database * get_database () const
- model * get_model_ptr () const
- vdbl::database * get_database_ptr () const
- vdbl::userid get_dbuserid () const
- gptr< search_node > * global_model () const
- gptr< vdbl::database > * database () const
- search_node_id get_id () const
- void set_id (const search_node_id &i)
- vdbl::rowid get_rowid () const
- void set_rowid (const vdbl::rowid &i)

- void keep (const annotation &_an)
- void keep (const std::vector< annotation > &_anv)
- void unkeep (const annotation &_an)
- void unkeep (const std::vector< annotation > &_anv)

Public Attributes

- std::vector< annotation > _ann

Protected Member Functions

- full_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__mod, gptr< search_node > *_gm, gptr< vdbl::database > &_db, search_node_relation _snr=snr_reduction)
- full_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__mod, gptr< search_node > *_gm, gptr< vdbl::database > &_db, const std::vector< annotation > &_a, search_node_relation _snr=snr_reduction)
- search_node_relation parent_relation () const

Protected Attributes

- gptr< model > * _m
- gptr< search_node > * __global_model
- gptr< vdbl::database > * __dbase
- vdbl::userid _dbuser
- search_node_relation _snr
- search_node_id _id
- std::vector< annotation > _keep
- vdbl::rowid _rid

Friends

- class delta_base
- class certificate_base
- class dag_delta
- class dag_undelta
- class search_graph

10.114.1 Detailed Description

This is a class of nodes stored in the search graph. It represents a full node, i.e., a node which stores a complete model not just changes to the parent model.

10.114.2 Constructor & Destructor Documentation

10.114.2.1 coco::full_node::full_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __mod, gptr< search_node > * _gm, gptr< vdbl::database > & _db, search_node_relation _snr = snr_reduction) [inline, protected]

This constructor generates a new full node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameter ___mod is used to specify the full model, while the parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized as empty vector.

Definition at line 292 of file search_node.h.

10.114.2.2 coco::full_node::full_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __mod, gptr< search_node > * _gm, gptr< vdbl::database > & _db, const std::vector< annotation > & _a, search_node_relation _snr = snr_reduction) [inline, protected]

This constructor generates a new full node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameter ___mod is used to specify the full model, while the parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized by the parameter _a.

Definition at line 305 of file search_node.h.

10.114.2.3 coco::full_node::full_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __mod, gptr< search_node > & _gm, gptr< vdbl::database > & _db, search_node_relation _snr = snr_reduction) [inline]

This constructor generates a new full node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameter ___mod is used to specify the full model, while the parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized as empty vector.

Definition at line 320 of file search_node.h.

10.114.2.4 coco::full_node::full_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __mod, gptr< search_node > & _gm, gptr< vdbl::database > & _db, const std::vector< annotation > & _a, search_node_relation _snr = snr_reduction) [inline]

This constructor generates a new full node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameter ___mod is used to specify the full model, while the parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized by the parameter _a.

Definition at line 333 of file search_node.h.

10.114.2.5 coco::full_node::full_node (const full_node & _i) [inline]

Standard copy constructor

Definition at line 341 of file search_node.h.

10.114.2.6 virtual coco::full_node::~full_node ( ) [inline, virtual]

Standard Destructor

Definition at line 346 of file search_node.h.

10.114.3 Member Function Documentation

10.114.3.1 gptr<vdbl::database>* coco::coco::search_node::database ( ) const [inline, inherited]

This is the accessor method for the search database.

Definition at line 157 of file search_graph.cc.

10.114.3.2 const annotation& coco::full_node::get_annotation (unsigned int *i*) const [inline]

A call to this method returns a const reference to the `i`th annotation.

Definition at line 362 of file search_node.h.

10.114.3.3 const std::vector<annotation>& coco::full_node::get_annotations ( ) const [inline]

A call to this method returns a const reference to the whole vector of annotations.

Definition at line 367 of file search_node.h.

10.114.3.4 const vdbl::database* coco::full_node::get_database ( ) const [inline]

This method returns a const pointer pointing to a locally stored copy of the search database.

Bug This method needs to be replaced, once the COCONUT environment really gets distributed.

Definition at line 378 of file search_node.h.

10.114.3.5 vdbl::database* coco::full_node::get_database_ptr ( ) const [inline]

This method returns a const pointer pointing to a locally stored copy of the search database.

Bug This method needs to be replaced, once the COCONUT environment really gets distributed.

Definition at line 389 of file search_node.h.

10.114.3.6 vdbl::userid coco::coco::search_node::get_dbuserid ( ) const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 151 of file search_graph.cc.

10.114.3.7 search_node_id coco::coco::search_node::get_id ( ) const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 160 of file search_graph.cc.

10.114.3.8 const model* coco::full_node::get_model ( ) const [inline]

This method returns a const pointer pointing to a locally stored copy of the model stored in this full node.

Definition at line 372 of file search_node.h.

10.114.3.9 model* coco::full_node::get_model_ptr ( ) const [inline]

This method returns a pointer pointing to a locally stored copy of the model stored in this full node.

Definition at line 383 of file search_node.h.

10.114.3.10 vdbl::rowid coco::coco::search_node::get_rowid ( ) const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 166 of file search_graph.cc.

10.114.3.11 gptr<search_node>* coco::coco::search_node::global_model ( ) const [inline, inherited]

This is the accessor method for the global model.

Definition at line 154 of file search_graph.cc.

10.114.3.12 bool coco::full_node::is_delta ( ) const [inline, virtual]

This method is called to determine whether a search node stores only deltas (delta_node) or a full model (full_node). In case of a full_node it returns `false`.

Reimplemented from coco::coco::search_node.

Definition at line 354 of file search_node.h.

10.114.3.13 void coco::coco::search_node::keep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to be the keeper of the annotation _an.

Definition at line 173 of file search_graph.cc.

10.114.3.14 void coco::coco::search_node::keep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to be the keeper of all the annotations in _anv.

Definition at line 177 of file search_graph.cc.

10.114.3.15 unsigned int coco::full_node::n_annotations ( ) const [inline]

This method returns the number of annotations stored in this full node.

Definition at line 358 of file search_node.h.

10.114.3.16 full_node& coco::full_node::operator= (const full_node & *__w*)

Standard Assignment Operator

10.114.3.17 search_node_relation coco::coco::search_node::parent_relation ( ) const [inline, protected, inherited]

A call to this method returns the relation of this node to its parent in the search graph.

Definition at line 110 of file search_graph.cc.

10.114.3.18 void coco::coco::search_node::set_id (const search_node_id & *i*) [inline, inherited]

This is the set method for the search node id of this node.

Definition at line 163 of file search_graph.cc.

10.114.3.19 void coco::coco::search_node::set_rowid (const vdbl::rowid & *i*) [inline, inherited]

This is the set method for the row id of this node.

Definition at line 169 of file search_graph.cc.

10.114.3.20 void coco::search_node::unkeep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of the annotation _an.

Definition at line 82 of file search_node.hpp.

10.114.3.21 void coco::search_node::unkeep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of all the annotations in _anv.

Definition at line 93 of file search_node.hpp.

10.114.4 Friends And Related Function Documentation

10.114.4.1 friend class certificate_base [friend]

Definition at line 393 of file search_node.h.

10.114.4.2 friend class dag_delta [friend]

Definition at line 394 of file search_node.h.

10.114.4.3 friend class dag_undelta [friend]

Definition at line 395 of file search_node.h.

10.114.4.4 friend class delta_base [friend]

Definition at line 392 of file search_node.h.

10.114.4.5 friend class search_graph [friend, inherited]

Definition at line 188 of file search_graph.cc.

10.114.5 Member Data Documentation

10.114.5.1 gptr<vdbl::database>* coco::coco::search_node::__dbase [protected, inherited]

This is a pointer to a global pointer to the search database associated to this search graph.

Definition at line 89 of file search_graph.cc.

10.114.5.2 gptr<search_node>* coco::coco::search_node::__global_model [protected, inherited]

This is a pointer to a global pointer to the top of the search graph. It is NULL for a standalone node or the top of the graph.

Definition at line 86 of file search_graph.cc.

10.114.5.3 std::vector<annotation> coco::full_node::_ann

This is the vector of annotations active at this full node.

Definition at line 283 of file search_node.h.

10.114.5.4 vdbl::userid coco::coco::search_node::_dbuser [protected, inherited]

This member stores the database user id under which this search is performed. For the in-memory database this is irrelevant.

Definition at line 92 of file search_graph.cc.

10.114.5.5 search_node_id coco::coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 97 of file search_graph.cc.

10.114.5.6 std::vector<annotation> coco::coco::search_node::_keep [protected, inherited]

The _keep member is a list of all those annotations which are kept by this search node. If the node is destroyed all kept annotations are removed from the search database.

Definition at line 101 of file search_graph.cc.

10.114.5.7 gptr<model>* coco::full_node::_m [protected]

This is a pointer to a global pointer to the model kept in this full node.

Definition at line 279 of file search_node.h.

10.114.5.8 vdbl::rowid coco::coco::search_node::_rid [protected, inherited]

This row id specifies the row in the search info table where the hook information for this search_node is stored

Definition at line 105 of file search_graph.cc.

10.114.5.9 search_node_relation coco::coco::search_node::_snr [protected, inherited]

This member specifies the relation of the search node to its parent(s) in the search graph.

Definition at line 95 of file search_graph.cc.

The documentation for this class was generated from the following file:

- search_node.h

10.115 coco::func_d_eval Class Reference

Forward function evaluation with preparation of derivative data.

`#include <der_evaluator.h>`

Inheritance diagram for coco::func_d_eval:

Collaboration diagram for coco::func_d_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- func_d_eval (const std::vector< double > &__x, const variable_indicator &__v, const model &__m, std::vector< std::vector< double > > &__d, std::vector< double > *__c)
- func_d_eval (const func_d_eval &__x)
- ~func_d_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_point (const std::vector< double > &__x, const variable_indicator &__v)

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const double &__rval)
- int update (const expression_node &__data, const double &__rval)
- double calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.115.1 Detailed Description

This class is a cached forward evaluator performing a function evaluation and the preparation of partial derivative information for backwards differentiation.

10.115.2 Member Typedef Documentation

10.115.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.115.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.115.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.115.3 Constructor & Destructor Documentation

10.115.3.1 coco::func_d_eval::func_d_eval (const std::vector< double > & __x, const variable_indicator & __v, const model & __m, std::vector< std::vector< double > > & __d, std::vector< double > * __c) [inline]

Constructor: __x is the point where the derivative shall be evaluated, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies the DAG, __d the derivative data, and __c the cache (may be NULL).

Definition at line 243 of file der_evaluator.h.

10.115.3.2 coco::func_d_eval::func_d_eval (const func_d_eval & __x) [inline]

Standard Copy Constructor

Definition at line 258 of file der_evaluator.h.

10.115.3.3 coco::func_d_eval::~func_d_eval ( ) [inline]

Standard Destructor

Definition at line 261 of file der_evaluator.h.

10.115.4 Member Function Documentation

10.115.4.1 void coco::func_d_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 343 of file der_evaluator.h.

10.115.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.115.4.3 double coco::func_d_eval::calculate_value (bool eval_all) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 1031 of file der_evaluator.h.

10.115.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.115.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.115.4.6 void coco::func_d_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 278 of file der_evaluator.h.

10.115.4.7 int coco::func_d_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 280 of file der_evaluator.h.

10.115.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The ___data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.115.4.9 bool coco::func_d_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the value and derivative information for this node are already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 162 of file der_evaluator.h.

10.115.4.10 void coco::func_d_eval::new_point (const std::vector< double > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation point to ___x. The parameter ___v specifies which variables have changed value since the last evaluation.

Definition at line 270 of file der_evaluator.h.

10.115.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.115.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.115.4.13 void coco::func_d_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 353 of file der_evaluator.h.

10.115.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.115.4.15 expression_const_walker coco::func_d_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 264 of file der_evaluator.h.

10.115.4.16 int coco::func_d_eval::update (const double & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 384 of file der_evaluator.h.

10.115.4.17 int coco::func_d_eval::update (const expression_node & *__data*, const double & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 390 of file der_evaluator.h.

10.115.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.115.4.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.115.4.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.115.4.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.115.4.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.115.4.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

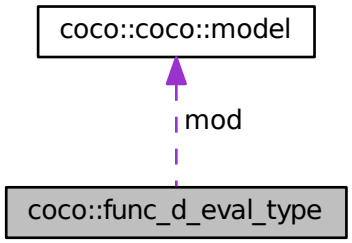
- der_evaluator.h

10.116 coco::func_d_eval_type Struct Reference

Visitor data for func_d_eval.

`#include <der_evaluator.h>`

Collaboration diagram for coco::func_d_eval_type:



Public Attributes

- const std::vector< double > * x
- std::vector< double > * f_cache
- std::vector< std::vector < double > > * d_data
- const model * mod
- union {
  void * p
  double d
  } u
- double r

- unsigned int n
- unsigned int info

10.116.1 Detailed Description

This class is the visitor data type for the func_d_eval evaluator.

10.116.2 Member Data Documentation

10.116.2.1 double coco::func_d_eval_type::d

Definition at line 137 of file der_evaluator.h.

10.116.2.2 std::vector<std::vector<double> >* coco::func_d_eval_type::d_data

the derivative data

Definition at line 135 of file der_evaluator.h.

10.116.2.3 std::vector<double>* coco::func_d_eval_type::f_cache

the function value cache

Definition at line 134 of file der_evaluator.h.

10.116.2.4 unsigned int coco::func_d_eval_type::info

needed for derivative of max, min

Definition at line 139 of file der_evaluator.h.

10.116.2.5 const model* coco::func_d_eval_type::mod

the DAG

Definition at line 136 of file der_evaluator.h.

10.116.2.6 unsigned int coco::func_d_eval_type::n

children counter

Definition at line 139 of file der_evaluator.h.

10.116.2.7 void* coco::func_d_eval_type::p

Definition at line 137 of file der_evaluator.h.

10.116.2.8 double coco::func_d_eval_type::r

return value

Definition at line 138 of file der_evaluator.h.

10.116.2.9 union { ... } coco::func_d_eval_type::u

additional data for complex nodes

10.116.2.10 const std::vector<double>* coco::func_d_eval_type::x

the point

Definition at line 133 of file der_evaluator.h.

The documentation for this struct was generated from the following file:

- der_evaluator.h

10.117 coco::func_eval Class Reference

Forward function evaluation.

`#include <func_evaluator.h>`

Inheritance diagram for coco::func_eval:

Collaboration diagram for coco::func_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- func_eval (const std::vector< double > &__x, const variable_indicator &__v, const model &__m, std::vector< double > *__c)
- func_eval (const func_eval &__v)
- ~func_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_point (const std::vector< double > &__x, const variable_indicator &__v)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)

- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const double &__rval)
- int update (const expression_node &__data, const double &__rval)
- double calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.117.1 Detailed Description

This class is a cached forward evaluator performing a function evaluation.

10.117.2 Member Typedef Documentation

10.117.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.117.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.117.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.117.3 Constructor & Destructor Documentation

10.117.3.1 coco::func_eval::func_eval (const std::vector< double > & __x, const variable_indicator & __v, const model & __m, std::vector< double > * __c) [inline]

Constructor: ___x is the point where the function shall be evaluated, ___v is a variable indicator specifying the variables that have changed value since the last evaluation, ___m specifies the DAG, and ___c the cache (may be NULL).

Definition at line 141 of file func_evaluator.h.

10.117.3.2 coco::func_eval::func_eval (const func_eval & __v) [inline]

Standard Copy Constructor

Definition at line 153 of file func_evaluator.h.

10.117.3.3 coco::func_eval::~func_eval ( ) [inline]

Standard Destructor

Definition at line 156 of file func_evaluator.h.

10.117.4 Member Function Documentation

10.117.4.1 void coco::func_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 251 of file func_evaluator.h.

10.117.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.117.4.3 double coco::func_eval::calculate_value (bool eval_all) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 735 of file func_evaluator.h.

10.117.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.117.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.117.4.6 void coco::func_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 173 of file func_evaluator.h.

10.117.4.7 int coco::func_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 175 of file func_evaluator.h.

10.117.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.117.4.9 bool coco::func_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the value for this node is already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 82 of file func_evaluator.h.

10.117.4.10 void coco::func_eval::new_point (const std::vector< double > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation point to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 165 of file func_evaluator.h.

10.117.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.117.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.117.4.13 void coco::func_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 260 of file func_evaluator.h.

10.117.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.117.4.15 expression_const_walker coco::func_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 159 of file func_evaluator.h.

10.117.4.16 int coco::func_eval::update (const double & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 278 of file func_evaluator.h.

10.117.4.17 int coco::func_eval::update (const expression_node & __data, const double & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 284 of file func_evaluator.h.

10.117.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.117.4.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.117.4.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.117.4.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.117.4.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.117.4.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

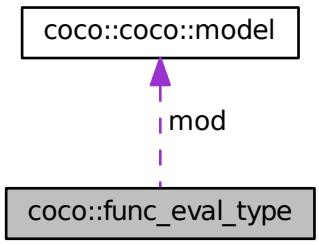
- func_evaluator.h

10.118 coco::func_eval_type Struct Reference

Visitor data for func_eval.

`#include <func_evaluator.h>`

Collaboration diagram for coco::func_eval_type:



Public Attributes

- const std::vector< double > * x
- std::vector< double > * cache
- const model * mod
- union {
  void * p
  double d
  } u
- double r
- unsigned int n

10.118.1 Detailed Description

This class is the visitor data type for the func_eval evaluator.

10.118.2 Member Data Documentation

10.118.2.1 std::vector<double>* coco::func_eval_type::cache

the function value cache

Definition at line 57 of file func_evaluator.h.

10.118.2.2 double coco::func_eval_type::d

Definition at line 59 of file func_evaluator.h.

10.118.2.3 const model* coco::func_eval_type::mod

the DAG

Definition at line 58 of file func_evaluator.h.

10.118.2.4 unsigned int coco::func_eval_type::n

children counter

Definition at line 61 of file func_evaluator.h.

10.118.2.5 void* coco::func_eval_type::p

Definition at line 59 of file func_evaluator.h.

10.118.2.6 double coco::func_eval_type::r

return value

Definition at line 60 of file func_evaluator.h.

10.118.2.7 union { ... } coco::func_eval_type::u

additional data for complex nodes

10.118.2.8 const std::vector<double>* coco::func_eval_type::x

the point

Definition at line 56 of file func_evaluator.h.

The documentation for this struct was generated from the following file:

- func_evaluator.h

10.119 coco::func_id_eval Class Reference

Forward function range evaluation with preparation of interval derivative data.

`#include <ider_evaluator.h>`

Inheritance diagram for coco::func_id_eval:

Collaboration diagram for coco::func_id_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- func_id_eval (const std::vector< interval > &__x, const std::vector< interval > &__rg, const variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > &__d, std::vector< interval > *__c, bool intersect=true, const std::vector< interval > *_center=NULL)
- func_id_eval (const func_id_eval &__x)
- ~func_id_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_box (const std::vector< interval > &__x, const variable_indicator &__v)
- void new_range (const std::vector< interval > &__rg, const variable_indicator &__v)
- void new_center (const std::vector< interval > *_center)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const interval &__rval)
- int update (const expression_node &__data, const interval &__rval)
- interval calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.119.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation using the natural interval extension of the function, and it prepares partial interval derivative information for backwards interval differentiation.

10.119.2 Member Typedef Documentation

10.119.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.119.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.119.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.119.3 Constructor & Destructor Documentation

10.119.3.1 coco::func_id_eval::func_id_eval (const std::vector< interval > & __x, const std::vector< interval > & __rg, const variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > & __d, std::vector< interval > * __c, bool *intersect* = true, const std::vector< interval > * *_center* = NULL) [inline]

Constructor: __x is the box over which the interval derivative shall be evaluated, __rg contains the ranges of all the DAG nodes, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies the DAG, __d the partial interval derivative data, and __c the cache (may be NULL).

Definition at line 248 of file ider_evaluator.h.

10.119.3.2 coco::func_id_eval::func_id_eval (const func_id_eval & __x) [inline]

Standard Copy Constructor

Definition at line 271 of file ider_evaluator.h.

10.119.3.3 coco::func_id_eval::~func_id_eval ( ) [inline]

Standard Destructor

Definition at line 274 of file ider_evaluator.h.

10.119.4 Member Function Documentation

10.119.4.1 void coco::func_id_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 376 of file ider_evaluator.h.

10.119.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.119.4.3 interval coco::func_id_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 1545 of file ider_evaluator.h.

10.119.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.119.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.119.4.6 void coco::func_id_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 306 of file ider_evaluator.h.

10.119.4.7 int coco::func_id_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 308 of file ider_evaluator.h.

10.119.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.119.4.9 bool coco::func_id_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the range and interval derivative information for this node are already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 168 of file ider_evaluator.h.

10.119.4.10 void coco::func_id_eval::new_box (const std::vector< interval > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation box to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 283 of file ider_evaluator.h.

10.119.4.11 void coco::func_id_eval::new_center (const std::vector< interval > * _center) [inline]

This method changes the center for centered forms to _center.

Definition at line 299 of file ider_evaluator.h.

10.119.4.12 void coco::func_id_eval::new_range (const std::vector< interval > & __rg, const variable_indicator & __v) [inline]

This method changes the node ranges to __rg. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 292 of file ider_evaluator.h.

10.119.4.13 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.119.4.14 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.119.4.15 void coco::func_id_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 389 of file ider_evaluator.h.

10.119.4.16 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.119.4.17 expression_const_walker coco::func_id_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 277 of file ider_evaluator.h.

10.119.4.18 int coco::func_id_eval::update (const interval & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 419 of file ider_evaluator.h.

10.119.4.19 int coco::func_id_eval::update (const expression_node & __data, const interval & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 425 of file ider_evaluator.h.

10.119.4.20 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.119.4.21 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.119.4.22 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.119.4.23 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.119.4.24 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.119.4.25 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

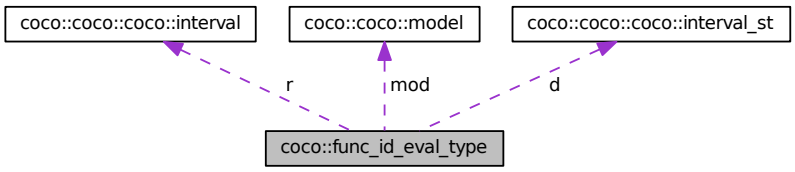
- ider_evaluator.h

10.120 coco::func_id_eval_type Struct Reference

Visitor data for func_id_eval.

`#include <ider_evaluator.h>`

Collaboration diagram for coco::func_id_eval_type:



Public Attributes

- const std::vector< interval > * x
- const std::vector< interval > * range
- std::vector< interval > * f
- std::vector< std::vector < interval > > * id_data
- const model * mod
- union {
  void * p
  interval_st d
  unsigned int info
  } u
- interval r
- unsigned int n
- bool intersect
- const std::vector< interval > * c

10.120.1 Detailed Description

This class is the visitor data type for the func_id_eval evaluator.

10.120.2 Member Data Documentation

10.120.2.1 const std::vector<interval>* coco::func_id_eval_type::c

the center for centered forms

Definition at line 145 of file ider_evaluator.h.

10.120.2.2 interval_st coco::func_id_eval_type::d

Definition at line 141 of file ider_evaluator.h.

10.120.2.3 std::vector<interval>* coco::func_id_eval_type::f

the function range cache

Definition at line 138 of file ider_evaluator.h.

10.120.2.4 std::vector<std::vector<interval> >* coco::func_id_eval_type::id_data

the interval derivative data

Definition at line 139 of file ider_evaluator.h.

10.120.2.5 unsigned int coco::func_id_eval_type::info

Definition at line 141 of file ider_evaluator.h.

10.120.2.6 bool coco::func_id_eval_type::intersect

intersect with ranges

Definition at line 144 of file ider_evaluator.h.

10.120.2.7 const model* coco::func_id_eval_type::mod

the DAG

Definition at line 140 of file ider_evaluator.h.

10.120.2.8 unsigned int coco::func_id_eval_type::n

children counter

Definition at line 143 of file ider_evaluator.h.

10.120.2.9 void* coco::func_id_eval_type::p

Definition at line 141 of file ider_evaluator.h.

10.120.2.10 interval coco::func_id_eval_type::r

return value

Definition at line 142 of file ider_evaluator.h.

10.120.2.11 const std::vector<interval>* coco::func_id_eval_type::range

the ranges of all nodes

Definition at line 137 of file ider_evaluator.h.

10.120.2.12 union { ... } coco::func_id_eval_type::u

additional data for complex nodes

10.120.2.13 const std::vector<interval>* coco::func_id_eval_type::x

the box

Definition at line 136 of file ider_evaluator.h.

The documentation for this struct was generated from the following file:

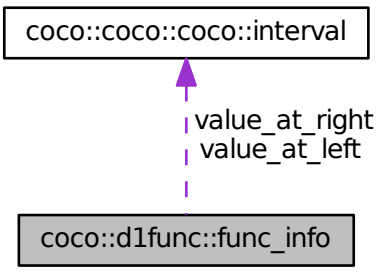
- ider_evaluator.h

10.121 coco::d1func::func_info Struct Reference

the func_info class collects all information needed for optimal evaluation

`#include <d1func.h>`

Collaboration diagram for coco::d1func::func_info:



Public Member Functions

- func_info ()
- func_info (d1func_monotonicity_t m, d1func_monotonicity_t p)
- func_info (d1func_monotonicity_t m, d1func_monotonicity_t p, const point_list &_v)
- func_info (const func_info &_f)
- ~func_info ()

Public Attributes

- interval value_at_left
- interval value_at_right
- d1func_monotonicity_t mon_towards_left
- d1func_monotonicity_t mon_towards_right
- point_list v

10.121.1 Detailed Description

The func_info class collects all information needed for optimal evaluation.

10.121.2 Constructor & Destructor Documentation

10.121.2.1 coco::d1func::func_info::func_info ( ) [inline]

Default Constructor

Definition at line 134 of file d1func.h.

10.121.2.2 coco::d1func::func_info::func_info (d1func_monotonicity_t *m*, d1func_monotonicity_t *p*) [inline]

Constructor setting the behaviour towards the borders

Definition at line 136 of file d1func.h.

10.121.2.3 coco::d1func::func_info::func_info (d1func_monotonicity_t *m*, d1func_monotonicity_t *p*, const point_list & *_v*) [inline]

Constructor setting all the behaviour

Definition at line 139 of file d1func.h.

10.121.2.4 coco::d1func::func_info::func_info (const func_info & *_f*) [inline]

Copy Constructor

Definition at line 144 of file d1func.h.

10.121.2.5 coco::d1func::func_info::~func_info ( ) [inline]

Standard Destructor

Definition at line 151 of file d1func.h.

10.121.3 Member Data Documentation

10.121.3.1 d1func_monotonicity_t coco::d1func::func_info::mon_towards_left

monotonicity towards the lower end of the DOD

Definition at line 127 of file d1func.h.

10.121.3.2 d1func_monotonicity_t coco::d1func::func_info::mon_towards_right

monotonicity towards the upper end of the DOD

Definition at line 129 of file d1func.h.

10.121.3.3 point_list coco::d1func::func_info::v

the list of all extrema of the corresponding derivative

Definition at line 131 of file d1func.h.

10.121.3.4 interval coco::d1func::func_info::value_at_left

asymptotic value at the lower end of the DOD

Definition at line 123 of file d1func.h.

10.121.3.5 interval coco::d1func::func_info::value_at_right

asymptotic value at the upper end of the DOD

Definition at line 125 of file d1func.h.

The documentation for this struct was generated from the following file:

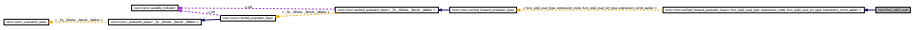
- d1func.h

10.122 coco::func_islp2_eval Class Reference

`#include <islp2_evaluator.h>`

Inheritance diagram for coco::func_islp2_eval:

Collaboration diagram for coco::func_islp2_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- func_islp2_eval (const std::vector< Islope > &__z, const std::vector< double > &__w, const std::vector< interval > &__rg, const variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > &__c, std::vector< std::vector< interval > > &__d, std::vector< std::vector< interval > > &__h, std::vector< Islope > *__zc, std::vector< interval > *__wc, bool inters=true)
- func_islp2_eval (const func_islp2_eval &__x)
- ~func_islp2_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void newPoint (const std::vector< Islope > &__z, const std::vector< double > &__w, const variable_indicator &__v)
- void newRange (const std::vector< interval > &__rg, const variable_indicator &__v)
- void changeIntersectBehaviour (bool intersect)
- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const func_islp2_eval_ret_type &__rval)
- int update (const expression_node &__data, const func_islp2_eval_ret_type &__rval)
- func_islp2_eval_ret_type calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.122.1 Member Typedef Documentation

10.122.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.122.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.122.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.122.2 Constructor & Destructor Documentation

10.122.2.1 coco::func_islp2_eval::func_islp2_eval (const std::vector< Islope > & __z, const std::vector< double > & __w, const std::vector< interval > & __rg, const variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > & __c, std::vector< std::vector< interval > > & __d, std::vector< std::vector< interval > > & __h, std::vector< Islope > * __zc, std::vector< interval > * __wc, bool inters = true) [inline]

Definition at line 237 of file islp2_evaluator.h.

10.122.2.2 coco::func_islp2_eval::func_islp2_eval (const func_islp2_eval & __x) [inline]

Definition at line 286 of file islp2_evaluator.h.

10.122.2.3 coco::func_islp2_eval::~func_islp2_eval ( ) [inline]

Definition at line 290 of file islp2_evaluator.h.

10.122.3 Member Function Documentation

10.122.3.1 void coco::func_islp2_eval::calculate (const expression_node & __data) [inline]

Definition at line 423 of file islp2_evaluator.h.

10.122.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.122.3.3 func_islp2_eval_ret_type coco::func_islp2_eval::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_islp2_eval_type, expression_node, func_islp2_eval_ret_type, expression_const_walker >.

Definition at line 1439 of file islp2_evaluator.h.

10.122.3.4 void coco::func_islp2_eval::changeIntersectBehaviour (bool *intersect*) [inline]

Definition at line 334 of file islp2_evaluator.h.

10.122.3.5 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The `___data` parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.122.3.6 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.122.3.7 void coco::func_islp2_eval::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_islp2_eval_type, expression_node, func_islp2_eval_ret_type, expression_const_walker >.

Definition at line 339 of file islp2_evaluator.h.

10.122.3.8 int coco::func_islp2_eval::initialize (const expression_node & *__data*) [inline]

Definition at line 349 of file islp2_evaluator.h.

10.122.3.9 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The `___data` parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.122.3.10 bool coco::func_islp2_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_islp2_eval_type, expression_node, func_islp2_eval_ret_type, expression_const_walker >.

Definition at line 194 of file islp2_evaluator.h.

10.122.3.11 void coco::func_islp2_eval::newPoint (const std::vector< Islope > & __z, const std::vector< double > & __w, const variable_indicator & __v) [inline]

Definition at line 295 of file islp2_evaluator.h.

10.122.3.12 void coco::func_islp2_eval::newRange (const std::vector< interval > & __rg, const variable_indicator & __v) [inline]

This method changes the node ranges to ___rg. The parameter $*$ ___v specifies which variables have changed value since the last $*$ evaluation.

Definition at line 328 of file islp2_evaluator.h.

10.122.3.13 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.122.3.14 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.122.3.15 void coco::func_islp2_eval::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 432 of file islp2_evaluator.h.

10.122.3.16 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.122.3.17 expression_const_walker coco::func_islp2_eval::short_cut_to (const expression_node & __data) [inline]

Definition at line 292 of file islp2_evaluator.h.

10.122.3.18 int coco::func_islp2_eval::update (const func_islp2_eval_ret_type & __rval) [inline]

Definition at line 439 of file islp2_evaluator.h.

10.122.3.19 int coco::func_islp2_eval::update (const expression_node & __data, const func_islp2_eval_ret_type & __rval) [inline]

Definition at line 449 of file islp2_evaluator.h.

10.122.3.20 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.122.3.21 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The ___data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.122.3.22 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter

`false`.

Definition at line 763 of file search_graph.cc.

10.122.3.23 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.122.3.24 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.122.3.25 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

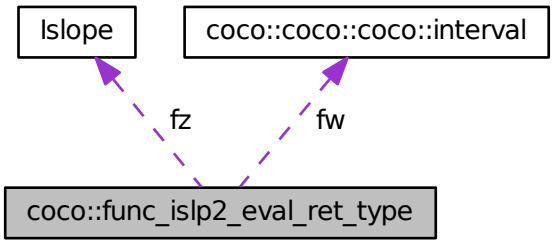
The documentation for this class was generated from the following file:

- islp2_evaluator.h

10.123 coco::func_islp2_eval_ret_type Struct Reference

`#include <islp2_evaluator.h>`

Collaboration diagram for coco::func_islp2_eval_ret_type:



Public Attributes

- Islope fz
- interval fw
- unsigned int nn

10.123.1 Member Data Documentation

10.123.1.1 interval coco::func_islp2_eval_ret_type::fw

Definition at line 153 of file islp2_evaluator.h.

10.123.1.2 Islope coco::func_islp2_eval_ret_type::fz

Definition at line 152 of file islp2_evaluator.h.

10.123.1.3 unsigned int coco::func_islp2_eval_ret_type::nn

Definition at line 155 of file islp2_evaluator.h.

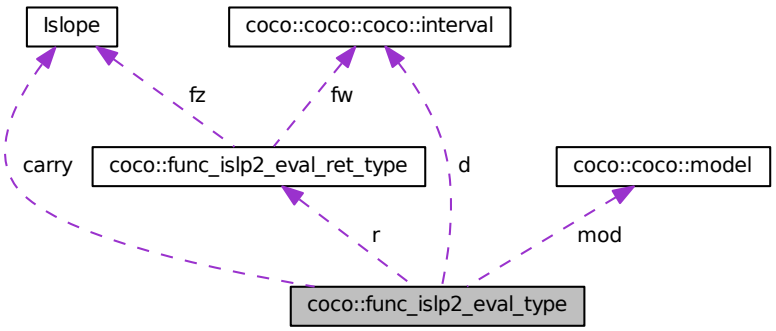
The documentation for this struct was generated from the following file:

- islp2_evaluator.h

10.124 coco::func_islp2_eval_type Struct Reference

`#include <islp2_evaluator.h>`

Collaboration diagram for coco::func_islp2_eval_type:



Public Attributes

- const std::vector< Islope > * z

- const std::vector< double > * w
- const std::vector< interval > * range
- std::vector< Islope > * z_cache
- std::vector< interval > * w_cache
- std::vector< std::vector < interval > > * h_data
- std::vector< std::vector < interval > > * d_data
- std::vector< std::vector < interval > > * c_data
- std::vector< unsigned int > * t
- std::vector< bool > * b
- bool do_intersect
- func_islp2_eval_ret_type r
- Islope carry
- const model * mod
- void * p
- interval d
- unsigned int n
- unsigned int info

10.124.1 Member Data Documentation

10.124.1.1 std::vector<bool>* coco::func_islp2_eval_type::b

Definition at line 175 of file islp2_evaluator.h.

10.124.1.2 std::vector<std::vector<interval> >* coco::func_islp2_eval_type::c_data

Definition at line 173 of file islp2_evaluator.h.

10.124.1.3 Islope coco::func_islp2_eval_type::carry

Definition at line 178 of file islp2_evaluator.h.

10.124.1.4 interval coco::func_islp2_eval_type::d

Definition at line 181 of file islp2_evaluator.h.

10.124.1.5 std::vector<std::vector<interval> >* coco::func_islp2_eval_type::d_data

Definition at line 172 of file islp2_evaluator.h.

10.124.1.6 bool coco::func_islp2_eval_type::do_intersect

Definition at line 176 of file islp2_evaluator.h.

10.124.1.7 std::vector<std::vector<interval> >* coco::func_islp2_eval_type::h_data

Definition at line 171 of file islp2_evaluator.h.

10.124.1.8 unsigned int coco::func_islp2_eval_type::info

Definition at line 182 of file islp2_evaluator.h.

10.124.1.9 const model* coco::func_islp2_eval_type::mod

Definition at line 179 of file islp2_evaluator.h.

10.124.1.10 unsigned int coco::func_islp2_eval_type::n

Definition at line 182 of file islp2_evaluator.h.

10.124.1.11 void* coco::func_islp2_eval_type::p

Definition at line 180 of file islp2_evaluator.h.

10.124.1.12 func_islp2_eval_ret_type coco::func_islp2_eval_type::r

Definition at line 177 of file islp2_evaluator.h.

10.124.1.13 const std::vector<interval>* coco::func_islp2_eval_type::range

the ranges of all nodes

Definition at line 168 of file islp2_evaluator.h.

10.124.1.14 std::vector<unsigned int>* coco::func_islp2_eval_type::t

Definition at line 174 of file islp2_evaluator.h.

10.124.1.15 const std::vector<double>* coco::func_islp2_eval_type::w

Definition at line 167 of file islp2_evaluator.h.

10.124.1.16 std::vector<interval>* coco::func_islp2_eval_type::w_cache

Definition at line 170 of file islp2_evaluator.h.

10.124.1.17 const std::vector<Islope>* coco::func_islp2_eval_type::z

Definition at line 166 of file islp2_evaluator.h.

10.124.1.18 std::vector<Islope>* coco::func_islp2_eval_type::z_cache

Definition at line 169 of file islp2_evaluator.h.

The documentation for this struct was generated from the following file:

- islp2_evaluator.h

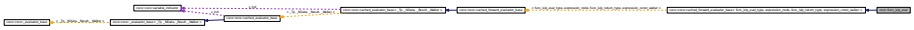
10.125 coco::func_islp_eval Class Reference

Forward function range evaluation with preparation of first order slope data.

`#include <islp_evaluator.h>`

Inheritance diagram for coco::func_islp_eval:

Collaboration diagram for coco::func_islp_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- func_islp_eval (const std::vector< double > &__z, const std::vector< interval > &__rg, const variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > &__d, std::vector< interval > &__f, bool intsect=true)
- func_islp_eval (const func_islp_eval &__x)
- ~func_islp_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_point (const std::vector< double > &__x, const variable_indicator &__v)
- void new_range (const std::vector< interval > &__rg, const variable_indicator &__v)
- void set_do_intersect (bool intsect)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const func_islp_return_type &__rval)
- int update (const expression_node &__data, const func_islp_return_type &__rval)
- func_islp_return_type calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const expression_node &__data)

10.125.1 Detailed Description

This class is a cached forward evaluator performing a function evaluation at the center, and it prepares partial first order slope information for backwards slope calculation.

10.125.2 Member Typedef Documentation

10.125.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.125.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.125.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.125.3 Constructor & Destructor Documentation

10.125.3.1 coco::func_islp_eval::func_islp_eval (const std::vector< double > & *__z*, const std::vector< interval > & *__rg*, const variable_indicator & *__v*, const model & *__m*, std::vector< std::vector< interval > > & *__d*, std::vector< interval > & *__f*, bool *intsect* = true) [inline]

Constructor: __z is the center of the slope, __rg contains the ranges of all the DAG nodes, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies

the DAG, __d the partial first order slope data, and __f the center values at each node.

Definition at line 201 of file islp_evaluator.h.

10.125.3.2 coco::func_islp_eval::func_islp_eval (const func_islp_eval & __x) [inline]

Standard Copy Constructor

Definition at line 220 of file islp_evaluator.h.

10.125.3.3 coco::func_islp_eval::~func_islp_eval ( ) [inline]

Standard Destructor

Definition at line 223 of file islp_evaluator.h.

10.125.4 Member Function Documentation

10.125.4.1 void coco::func_islp_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 325 of file islp_evaluator.h.

10.125.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.125.4.3 func_islp_return_type coco::func_islp_eval::calculate_value (bool eval_all) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 1529 of file islp_evaluator.h.

10.125.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.125.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.125.4.6 void coco::func_islp_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 256 of file islp_evaluator.h.

10.125.4.7 int coco::func_islp_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 258 of file islp_evaluator.h.

10.125.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.125.4.9 bool coco::func_islp_eval::is_cached (const expression_node & __data) [inline, protected]

This function determines, whether the function value and first order slope information for this node are already available.

Definition at line 179 of file islp_evaluator.h.

10.125.4.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & __data) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.125.4.11 void coco::func_islp_eval::new_point (const std::vector< double > & __x, const variable_indicator & __v) [inline]

This method changes the center to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 232 of file islp_evaluator.h.

10.125.4.12 void coco::func_islp_eval::new_range (const std::vector< interval > & __rg, const variable_indicator & __v) [inline]

This method changes the node ranges to ___rg. The parameter ___v specifies which variables have changed value since the last evaluation.

Definition at line 241 of file islp_evaluator.h.

10.125.4.13 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.125.4.14 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.125.4.15 void coco::func_islp_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 338 of file islp_evaluator.h.

10.125.4.16 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.125.4.17 void coco::func_islp_eval::set_do_intersect (bool intsect) [inline]

This method changes the intersect behaviour of the evaluator.

Definition at line 249 of file islp_evaluator.h.

10.125.4.18 expression_const_walker coco::func_islp_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 226 of file islp_evaluator.h.

10.125.4.19 int coco::func_islp_eval::update (const func_islp_return_type & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 361 of file islp_evaluator.h.

10.125.4.20 int coco::func_islp_eval::update (const expression_node & *__data,* const func_islp_return_type & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 367 of file islp_evaluator.h.

10.125.4.21 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data,* const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.125.4.22 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.125.4.23 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.125.4.24 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.125.4.25 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.125.4.26 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

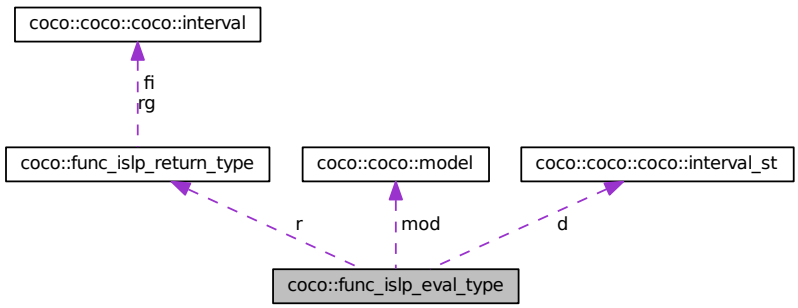
- islp_evaluator.h

10.126 coco::func_islp_eval_type Struct Reference

Visitor data for func_id_eval.

`#include <islp_evaluator.h>`

Collaboration diagram for coco::func_islp_eval_type:



Public Attributes

- const std::vector< double > * z
- const std::vector< interval > * range
- std::vector< interval > * f
- std::vector< std::vector < interval > > * islp_data
- const model * mod

- union {
  void * p
  interval_st d
  unsigned int info
  } u

- func_islp_return_type r
- unsigned int n
- bool do_intersect

10.126.1 Detailed Description

This class is the visitor data type for the func_id_eval evaluator.

10.126.2 Member Data Documentation

10.126.2.1 interval_st coco::func_islp_eval_type::d

Definition at line 153 of file islp_evaluator.h.

10.126.2.2 bool coco::func_islp_eval_type::do_intersect

intersect with known ranges?

Definition at line 157 of file islp_evaluator.h.

10.126.2.3 std::vector<interval>* coco::func_islp_eval_type::f

the central function value cache

Definition at line 150 of file islp_evaluator.h.

10.126.2.4 unsigned int coco::func_islp_eval_type::info

Definition at line 153 of file islp_evaluator.h.

10.126.2.5 std::vector<std::vector<interval> >* coco::func_islp_eval_type::islp_data

the first order slope data

Definition at line 151 of file islp_evaluator.h.

10.126.2.6 const model* coco::func_islp_eval_type::mod

the DAG

Definition at line 152 of file islp_evaluator.h.

10.126.2.7 unsigned int coco::func_islp_eval_type::n

children counter

Definition at line 156 of file islp_evaluator.h.

10.126.2.8 void* coco::func_islp_eval_type::p

Definition at line 153 of file islp_evaluator.h.

10.126.2.9 func_islp_return_type coco::func_islp_eval_type::r

return value

Definition at line 155 of file islp_evaluator.h.

10.126.2.10 const std::vector<interval>* coco::func_islp_eval_type::range

the ranges of all nodes

Definition at line 149 of file islp_evaluator.h.

10.126.2.11 union { ... } coco::func_islp_eval_type::u

additional data for complex nodes

10.126.2.12 const std::vector<double>* coco::func_islp_eval_type::z

the center

Definition at line 148 of file islp_evaluator.h.

The documentation for this struct was generated from the following file:

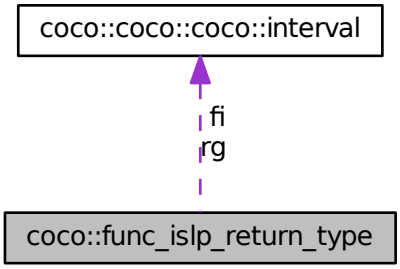
- islp_evaluator.h

10.127 coco::func_islp_return_type Struct Reference

The return type of the func_islp_eval evaluator.

`#include <islp_evaluator.h>`

Collaboration diagram for coco::func_islp_return_type:



Public Attributes

- interval rg
- interval fi

10.127.1 Detailed Description

This is the return type of the func_islp_eval evaluator.

10.127.2 Member Data Documentation

10.127.2.1 interval coco::func_islp_return_type::fi

Enclosure of the function value at the center

Definition at line 56 of file islp_evaluator.h.

10.127.2.2 interval coco::func_islp_return_type::rg

Function range

Definition at line 55 of file islp_evaluator.h.

The documentation for this struct was generated from the following file:

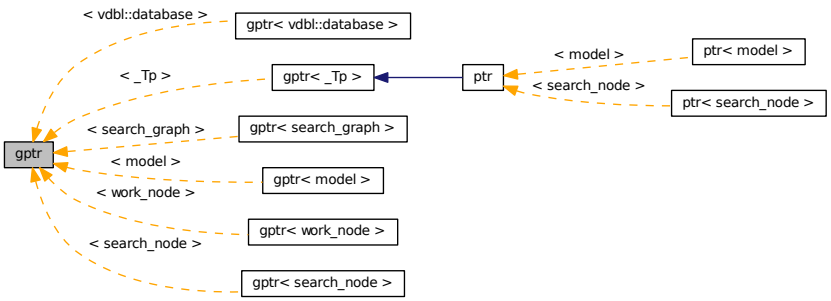
- islp_evaluator.h

10.128 gptr Class Reference

Global pointer class.

`#include <gptr.h>`

Inheritance diagram for gptr:



Public Member Functions

- virtual ~gptr ()

10.128.1 Detailed Description

This class is a generalization of a pointer which should be adaptable to various distributed and parallel computing paradigms.

10.128.2 Constructor & Destructor Documentation

10.128.2.1 virtual gptr::~gptr ( ) [inline, virtual]

Standard Destructor

Definition at line 53 of file gptr.h.

The documentation for this class was generated from the following file:

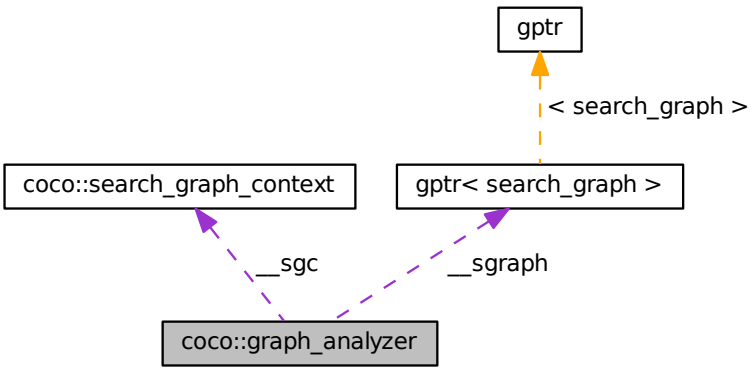
- gptr.h

10.129 coco::graph_analyzer Class Reference

Graph analyzer base class.

`#include <graph_analyzer.h>`

Collaboration diagram for coco::graph_analyzer:



Public Member Functions

- graph_analyzer (const gptr< search_graph > &sgraph, const search_focus &sfoc, const std::string &__n)
- graph_analyzer (const gptr< search_graph > &sgraph, const std::string &__n)
- virtual ~graph_analyzer ()
- virtual bool update_engine (const gptr< search_graph > &sgraph, const search_focus &sfoc)
- virtual bool update_engine (const gptr< search_graph > &sgraph)
- virtual ga_return_type analyze (const control_data &__c)
- const std::string & get_name () const

Protected Attributes

- std::string __name
- const gptr< search_graph > * __sgraph
- const search_focus * __sfoc
- const search_graph_context * __sgc
- const vdbl::viewdbase * __vdb

10.129.1 Detailed Description

This is the base class of all COCONUT graph analyzer modules. A graph analyzer is an inference module, which specializes in analyzing the search graph. A typical graph analyzer is, e.g., the box chooser, which selects the next search_node from the search_graph.

10.129.2 Constructor & Destructor Documentation

10.129.2.1 coco::graph_analyzer::graph_analyzer (const gptr< search_graph > & *sgraph*, const search_focus & *sfoc*, const std::string & *__n*) [inline]

This is the standard constructor for a graph_analyzer. It sets the indentifier string to `n`, the search graph to `sgraph`, and the search focus to `sfoc`.

Definition at line 102 of file graph_analyzer.h.

10.129.2.2 coco::graph_analyzer::graph_analyzer (const gptr< search_graph > & *sgraph*, const std::string & *__n*) [inline]

This is the standard constructor for a graph_analyzer if no search focus is available. It sets the indentifier string to `n` and the search graph to `sgraph`.

Definition at line 113 of file graph_analyzer.h.

10.129.2.3 virtual coco::graph_analyzer::~graph_analyzer ( ) [inline, virtual]

Standard Destructor

Definition at line 121 of file graph_analyzer.h.

10.129.3 Member Function Documentation

10.129.3.1 virtual ga_return_type coco::graph_analyzer::analyze (const control_data & *__c*) [inline, virtual]

This method is the main method of a graph analyzer. It is supposed to analyze the search graph and to return the information gained in the ga_return_type structure. This method is overloaded by the various subclasses. Service information and parameters are provided via the control_data structure __c.

Definition at line 151 of file graph_analyzer.h.

10.129.3.2 const std::string& coco::graph_analyzer::get_name ( ) const [inline]

The get_name method returns the identifier string of the graph analyzer.

Definition at line 155 of file graph_analyzer.h.

10.129.3.3 virtual bool coco::graph_analyzer::update_engine (const gptr< search_graph > & *sgraph*, const search_focus & *sfoc*) [inline, virtual]

This update_engine method changes the search graph to sgraph and the search focus to sfoc.

Definition at line 125 of file graph_analyzer.h.

10.129.3.4 virtual bool coco::graph_analyzer::update_engine (const gptr< search_graph > & *sgraph*) [inline, virtual]

This update_engine method changes the search graph to sgraph and clears the search focus.

Definition at line 137 of file graph_analyzer.h.

10.129.4 Member Data Documentation

10.129.4.1 std::string coco::graph_analyzer::__name [protected]

This is the identifier string for a graph analyzer.

Definition at line 87 of file graph_analyzer.h.

10.129.4.2 const search_focus* coco::graph_analyzer::__sfoc [protected]

This is the current search focus.

Definition at line 91 of file graph_analyzer.h.

10.129.4.3 const search_graph_context* coco::graph_analyzer::__sgc [protected]

This is the search_graph_context for extracting information from the search database.

Definition at line 94 of file graph_analyzer.h.

10.129.4.4 const gptr<search_graph>* coco::graph_analyzer::__sgraph [protected]

This variable contains a global pointer to the search graph.

Definition at line 89 of file graph_analyzer.h.

10.129.4.5 const vdbl::viewdbase* coco::graph_analyzer::__vdb [protected]

This is a view to the search database.

Definition at line 96 of file graph_analyzer.h.

The documentation for this class was generated from the following file:

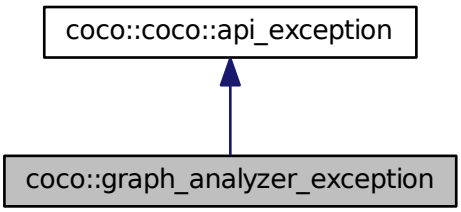
- graph_analyzer.h

10.130 coco::graph_analyzer_exception Class Reference

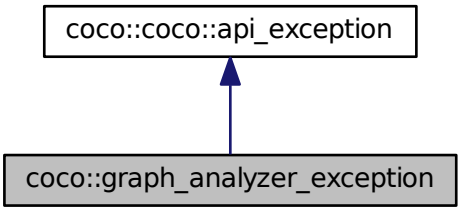
Graph analyzer exception class.

`#include <graph_analyzer.h>`

Inheritance diagram for coco::graph_analyzer_exception:



Collaboration diagram for coco::graph_analyzer_exception:



Public Member Functions

- graph_analyzer_exception (const std::string &msg)
- graph_analyzer_exception (const char ∗msg)
- virtual ∼graph_analyzer_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual char const ∗ what () const throw ()
- virtual char const ∗ what () const throw ()
- virtual char const ∗ what () const throw ()
- virtual char const ∗ what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()

10.130.1 Detailed Description

This is the exceptions class thrown by graph_analyzer subclasses. Every properly coded COCONUT graph analyzer module should only throw exceptions of this type.

10.130.2 Constructor & Destructor Documentation

10.130.2.1 coco::graph_analyzer_exception::graph_analyzer_exception (const std::string & *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 60 of file graph_analyzer.h.

10.130.2.2 coco::graph_analyzer_exception::graph_analyzer_exception (const char * *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 63 of file graph_analyzer.h.

10.130.2.3 virtual coco::graph_analyzer_exception::~graph_analyzer_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 67 of file graph_analyzer.h.

10.130.3 Member Function Documentation

10.130.3.1 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file expression.h.

10.130.3.2 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.130.3.3 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.130.3.4 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.130.3.5 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.130.3.6 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.130.3.7 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.130.3.8 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file expression.h.

10.130.3.9 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.130.3.10 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.130.3.11 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.130.3.12 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.130.3.13 const char * coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

Definition at line 57 of file api_exception.cc.

10.130.3.14 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.130.3.15 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.130.3.16 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.130.3.17 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.130.3.18 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file expression.h.

10.130.3.19 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.130.3.20 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following file:

- graph_analyzer.h

10.131 coco::graphorder_visitor Class Reference

This visitor class is used for computing a graph order.

Public Member Functions

- graphorder_visitor (std::vector< unsigned int > &__v, std::vector< bool > &__b, unsigned int __k=0)
- graphorder_visitor (const graphorder_visitor &__c)

- unsigned int vvalue ()
- unsigned int value ()
- bool postorder (const expression_node &r)
- void collect (const expression_node &r, unsigned int __v)
- void vcollect (unsigned int __v)

10.131.1 Detailed Description

This postorder visitor computes an order of the nodes compatible with the DAG structure.

10.131.2 Constructor & Destructor Documentation

10.131.2.1 coco::graphorder_visitor::graphorder_visitor (std::vector< unsigned int > & __v, std::vector< bool > & __b, unsigned int __k = 0) [inline]

Constructor setting phi to __v, phi_defined to __b, and k to __k.

Definition at line 56 of file graphorder.cc.

10.131.2.2 coco::graphorder_visitor::graphorder_visitor (const graphorder_visitor & __c) [inline]

Standard Copy Constructor

Definition at line 63 of file graphorder.cc.

10.131.3 Member Function Documentation

10.131.3.1 void coco::graphorder_visitor::collect (const expression_node & r, unsigned int __v) [inline]

This is a method required for a postorder visitor.

Definition at line 83 of file graphorder.cc.

10.131.3.2 bool coco::graphorder_visitor::postorder (const expression_node & r) [inline]

This is a method required for a postorder visitor.

Definition at line 72 of file graphorder.cc.

10.131.3.3 unsigned int coco::graphorder_visitor::value ( ) [inline]

This is a method required for a postorder visitor.

Definition at line 70 of file graphorder.cc.

10.131.3.4 void coco::graphorder_visitor::vcollect (unsigned int __v) [inline]

This is a method required for a postorder visitor.

Definition at line 88 of file graphorder.cc.

10.131.3.5 unsigned int coco::graphorder_visitor::vvalue ( ) [inline]

This is a method required for a postorder visitor.

Definition at line 69 of file graphorder.cc.

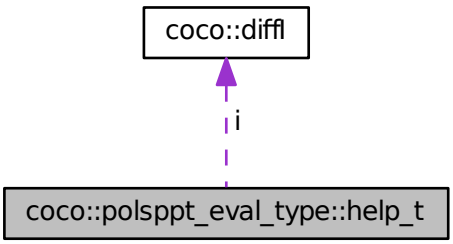
The documentation for this class was generated from the following file:

- graphorder.cc

10.132 coco::polsppt_eval_type::help_t Struct Reference

`#include <dagd1func_pv.hpp>`

Collaboration diagram for coco::polsppt_eval_type::help_t:



Public Attributes

- int info
- unsigned int n
- diffI i

10.132.1 Member Data Documentation

10.132.1.1 diffI coco::polsppt_eval_type::help_t::i

Definition at line 76 of file dagd1func_pv.hpp.

10.132.1.2 int coco::polsppt_eval_type::help_t::info

Definition at line 76 of file dagd1func_pv.hpp.

10.132.1.3 unsigned int coco::polsppt_eval_type::help_t::n

Definition at line 76 of file dagd1func_pv.hpp.

The documentation for this struct was generated from the following file:

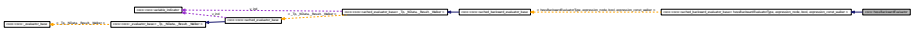
- dagd1func_pv.hpp

10.133 coco::hessBackwardEvaluator Class Reference

`#include <hess_evaluator.h>`

Inheritance diagram for coco::hessBackwardEvaluator:

Collaboration diagram for coco::hessBackwardEvaluator:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- hessBackwardEvaluator (std::vector< std::vector< double > > *__der_data, std::vector< std::vector< double > > *__hess_data, variable_indicator &__v, const model &__m, std::vector< std::vector< double > > *__d, std::vector< std::vector< double > > *__h, std::vector< double > *__grad, std::vector< double > *__hess)
- hessBackwardEvaluator (const hessBackwardEvaluator &__he)
- ~hessBackwardEvaluator ()

- void newPoint (std::vector< std::vector< double > > *__der_data, std::vector< std::vector< double > > *__hess_data, const variable_indicator &__v)
- void newResult (std::vector< double > *__grad, std::vector< double > *__hess)
- void set_mult (double scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.133.1 Member Typedef Documentation

10.133.1.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.133.1.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.133.1.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.133.2 Constructor & Destructor Documentation

10.133.2.1 coco::hessBackwardEvaluator::hessBackwardEvaluator (std::vector< std::vector< double > > * *__der_data*, std::vector< std::vector< double > > * *__hess_data*, variable_indicator & *__v*, const model & *__m*, std::vector< std::vector< double > > * *__d*, std::vector< std::vector< double > > * *__h*, std::vector< double > * *__grad*, std::vector< double > * *__hess*) [inline]

Definition at line 1258 of file hess_evaluator.h.

10.133.2.2 coco::hessBackwardEvaluator::hessBackwardEvaluator (const hessBackwardEvaluator & *__he*) [inline]

Definition at line 1288 of file hess_evaluator.h.

10.133.2.3 coco::hessBackwardEvaluator::~hessBackwardEvaluator ( ) [inline]

Definition at line 1290 of file hess_evaluator.h.

10.133.3 Member Function Documentation

10.133.3.1 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The `__data` parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.133.3.2 int coco::hessBackwardEvaluator::calculate (const expression_node & *__data*) [inline]

Definition at line 1330 of file hess_evaluator.h.

10.133.3.3 bool coco::hessBackwardEvaluator::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1569 of file hess_evaluator.h.

10.133.3.4 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.133.3.5 void coco::hessBackwardEvaluator::cleanup (const expression_node & __data) [inline]

Definition at line 1467 of file hess_evaluator.h.

10.133.3.6 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.133.3.7 void coco::hessBackwardEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1319 of file hess_evaluator.h.

10.133.3.8 bool coco::hessBackwardEvaluator::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1245 of file hess_evaluator.h.

10.133.3.9 void coco::hessBackwardEvaluator::newPoint (std::vector< std::vector< double > > * __der_data, std::vector< std::vector< double > > * __hess_data, const variable_indicator & __v) [inline]

Definition at line 1292 of file hess_evaluator.h.

10.133.3.10 void coco::hessBackwardEvaluator::newResult (std::vector< double > * __grad, std::vector< double > * __hess) [inline]

Definition at line 1302 of file hess_evaluator.h.

10.133.3.11 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.133.3.12 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.133.3.13 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.133.3.14 void coco::hessBackwardEvaluator::set_mult (double scal) [inline]

Definition at line 1308 of file hess_evaluator.h.

10.133.3.15 expression_const_walker coco::hessBackwardEvaluator::short_cut_to (const expression_node & __data) [inline]

Definition at line 1315 of file hess_evaluator.h.

10.133.3.16 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.133.3.17 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.133.3.18 int coco::hessBackwardEvaluator::update (const bool & *__rval*) [inline]

Definition at line 1505 of file hess_evaluator.h.

10.133.3.19 int coco::hessBackwardEvaluator::update (const expression_node & *__data*, const bool & *__rval*) [inline]

Definition at line 1515 of file hess_evaluator.h.

10.133.3.20 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 892 of file search_graph.cc.

10.133.3.21 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.133.3.22 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.133.3.23 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

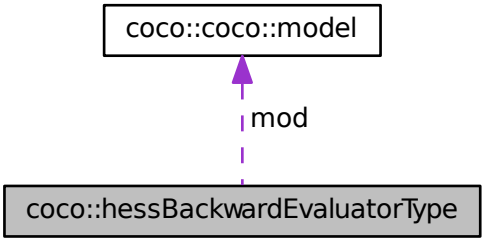
The documentation for this class was generated from the following file:

- hess_evaluator.h

10.134 coco::hessBackwardEvaluatorType Struct Reference

`#include <hess_evaluator.h>`

Collaboration diagram for coco::hessBackwardEvaluatorType:



Public Attributes

- std::vector< std::vector < double > > * d_data
- std::vector< std::vector < double > > * h_data
- std::vector< std::vector < double > > * d_cache
- std::vector< std::vector < double > > * h_cache
- std::vector< double > * grad_vec
- std::vector< double > * hess_vec
- bool calc_grad
- const model * mod
- double d_mult
- double hessian_of_current_node
- bool hess_zero
- double d_mult_trans
- double h_mult
- double h_mult_trans
- bool is_linear
- unsigned int child_n

10.134.1 Member Data Documentation

10.134.1.1 bool coco::hessBackwardEvaluatorType::calc_grad

Definition at line 1222 of file hess_evaluator.h.

10.134.1.2 unsigned int coco::hessBackwardEvaluatorType::child_n

Definition at line 1231 of file hess_evaluator.h.

10.134.1.3 std::vector<std::vector<double> >* coco::hessBackwardEvaluatorType::d_cache

Definition at line 1218 of file hess_evaluator.h.

10.134.1.4 std::vector<std::vector<double> >* coco::hessBackwardEvaluatorType::d_data

Definition at line 1216 of file hess_evaluator.h.

10.134.1.5 double coco::hessBackwardEvaluatorType::d_mult

Definition at line 1224 of file hess_evaluator.h.

10.134.1.6 double coco::hessBackwardEvaluatorType::d_mult_trans

Definition at line 1227 of file hess_evaluator.h.

10.134.1.7 std::vector<double>* coco::hessBackwardEvaluatorType::grad_vec

Definition at line 1220 of file hess_evaluator.h.

10.134.1.8 std::vector<std::vector<double> >* coco::hessBackwardEvaluatorType::h_cache

Definition at line 1219 of file hess_evaluator.h.

10.134.1.9 std::vector<std::vector<double> >* coco::hessBackwardEvaluatorType::h_data

Definition at line 1217 of file hess_evaluator.h.

10.134.1.10 double coco::hessBackwardEvaluatorType::h_mult

Definition at line 1228 of file hess_evaluator.h.

10.134.1.11 double coco::hessBackwardEvaluatorType::h_mult_trans

Definition at line 1229 of file hess_evaluator.h.

10.134.1.12 std::vector<double>* coco::hessBackwardEvaluatorType::hess_vec

Definition at line 1221 of file hess_evaluator.h.

10.134.1.13 bool coco::hessBackwardEvaluatorType::hess_zero

Definition at line 1226 of file hess_evaluator.h.

10.134.1.14 double coco::hessBackwardEvaluatorType::hessian_of_current_node

Definition at line 1225 of file hess_evaluator.h.

10.134.1.15 bool coco::hessBackwardEvaluatorType::is_linear

Definition at line 1230 of file hess_evaluator.h.

10.134.1.16 const model* coco::hessBackwardEvaluatorType::mod

Definition at line 1223 of file hess_evaluator.h.

The documentation for this struct was generated from the following file:

- hess_evaluator.h

10.135 coco::hessForwardEvaluator Class Reference

`#include <hess_evaluator.h>`

Inheritance diagram for coco::hessForwardEvaluator:

Collaboration diagram for coco::hessForwardEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- hessForwardEvaluator (const std::vector< diffNumber > &__x, const variable_indicator &__v, const model &__m, std::vector< std::vector< double > > &__d, std::vector< std::vector< double > > &__h, std::vector< diffNumber > *__c)
- hessForwardEvaluator (const hessForwardEvaluator &__x)
- ~hessForwardEvaluator ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void newPoint (const std::vector< diffNumber > &__x, const variable_indicator &__v)
- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const diffNumber &__rval)
- int update (const expression_node &__data, const hessForwardEvaluatorReturnValue &__rval)
- hessForwardEvaluatorReturnValue calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)

- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.135.1 Member Typedef Documentation

10.135.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.135.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.135.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.135.2 Constructor & Destructor Documentation

10.135.2.1 coco::hessForwardEvaluator::hessForwardEvaluator (const std::vector< diffNumber > & __x, const variable_indicator & __v, const model & __m, std::vector< std::vector< double > > & __d, std::vector< std::vector< double > > & __h, std::vector< diffNumber > * __c) [inline]

Definition at line 250 of file hess_evaluator.h.

10.135.2.2 coco::hessForwardEvaluator::hessForwardEvaluator (const hessForwardEvaluator & __x) [inline]

Definition at line 287 of file hess_evaluator.h.

10.135.2.3 coco::hessForwardEvaluator::~hessForwardEvaluator ( ) [inline]

Definition at line 291 of file hess_evaluator.h.

10.135.3 Member Function Documentation

10.135.3.1 void coco::hessForwardEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 409 of file hess_evaluator.h.

10.135.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.135.3.3 hessForwardEvaluatorReturnValue coco::hessForwardEvaluator::calculate_value (bool eval_all) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 1205 of file hess_evaluator.h.

10.135.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.135.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.135.3.6 void coco::hessForwardEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 321 of file hess_evaluator.h.

10.135.3.7 int coco::hessForwardEvaluator::initialize (const expression_node & __data) [inline]

Definition at line 330 of file hess_evaluator.h.

10.135.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.135.3.9 bool coco::hessForwardEvaluator::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 177 of file hess_evaluator.h.

10.135.3.10 void coco::hessForwardEvaluator::newPoint (const std::vector< diffNumber > & __x, const variable_indicator & __v) [inline]

Definition at line 296 of file hess_evaluator.h.

10.135.3.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.135.3.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.135.3.13 void coco::hessForwardEvaluator::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 420 of file hess_evaluator.h.

10.135.3.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.135.3.15 expression_const_walker coco::hessForwardEvaluator::short_cut_to (const expression_node & __data) [inline]

Definition at line 293 of file hess_evaluator.h.

10.135.3.16 int coco::hessForwardEvaluator::update (const diffNumber & __rval) [inline]

Definition at line 427 of file hess_evaluator.h.

10.135.3.17 int coco::hessForwardEvaluator::update (const expression_node & __data, const hessForwardEvaluatorReturnValue & __rval) [inline]

Definition at line 437 of file hess_evaluator.h.

10.135.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.135.3.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.135.3.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter

`false`.

Definition at line 763 of file search_graph.cc.

10.135.3.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.135.3.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.135.3.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

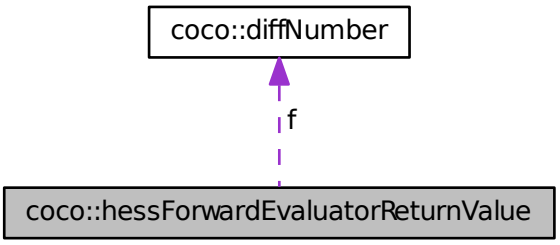
The documentation for this class was generated from the following file:

- hess_evaluator.h

10.136 coco::hessForwardEvaluatorReturnValue Struct Reference

`#include <hess_evaluator.h>`

Collaboration diagram for coco::hessForwardEvaluatorReturnValue:



Public Attributes

- diffNumber f
- unsigned int nn
- tristate in_chn

10.136.1 Member Data Documentation

10.136.1.1 diffNumber coco::hessForwardEvaluatorReturnValue::f

Definition at line 146 of file hess_evaluator.h.

10.136.1.2 tristate coco::hessForwardEvaluatorReturnValue::in_chn

Definition at line 149 of file hess_evaluator.h.

10.136.1.3 unsigned int coco::hessForwardEvaluatorReturnValue::nn

Definition at line 148 of file hess_evaluator.h.

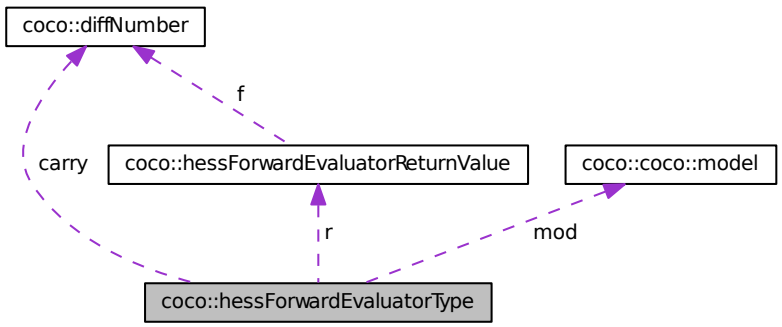
The documentation for this struct was generated from the following file:

- hess_evaluator.h

10.137 coco::hessForwardEvaluatorType Struct Reference

`#include <hess_evaluator.h>`

Collaboration diagram for coco::hessForwardEvaluatorType:



Public Attributes

- const std::vector< diffNumber > * x

- std::vector< diffNumber > * f_cache
- std::vector< std::vector < double > > * h_data
- std::vector< std::vector < double > > * d_data
- std::vector< unsigned int > * t
- std::vector< bool > * b
- hessForwardEvaluatorReturnValue r
- diffNumber carry
- const model * mod
- union {
  void * p
  double d
  } u

- unsigned int n
- unsigned int info

10.137.1 Member Data Documentation

10.137.1.1 std::vector<bool>* coco::hessForwardEvaluatorType::b

Definition at line 159 of file hess_evaluator.h.

10.137.1.2 diffNumber coco::hessForwardEvaluatorType::carry

Definition at line 161 of file hess_evaluator.h.

10.137.1.3 double coco::hessForwardEvaluatorType::d

Definition at line 163 of file hess_evaluator.h.

10.137.1.4 std::vector<std::vector<double> >* coco::hessForwardEvaluatorType::d_data

Definition at line 157 of file hess_evaluator.h.

10.137.1.5 std::vector<diffNumber>* coco::hessForwardEvaluatorType::f_cache

Definition at line 155 of file hess_evaluator.h.

10.137.1.6 std::vector<std::vector<double> >* coco::hessForwardEvaluatorType::h_data

Definition at line 156 of file hess_evaluator.h.

10.137.1.7 unsigned int coco::hessForwardEvaluatorType::info

Definition at line 164 of file hess_evaluator.h.

10.137.1.8 const model* coco::hessForwardEvaluatorType::mod

Definition at line 162 of file hess_evaluator.h.

10.137.1.9 unsigned int coco::hessForwardEvaluatorType::n

Definition at line 164 of file hess_evaluator.h.

10.137.1.10 void* coco::hessForwardEvaluatorType::p

Definition at line 163 of file hess_evaluator.h.

10.137.1.11 hessForwardEvaluatorReturnValue coco::hessForwardEvaluatorType::r

Definition at line 160 of file hess_evaluator.h.

10.137.1.12 std::vector<unsigned int>* coco::hessForwardEvaluatorType::t

Definition at line 158 of file hess_evaluator.h.

10.137.1.13 union { ... } coco::hessForwardEvaluatorType::u

10.137.1.14 const std::vector<diffNumber>* coco::hessForwardEvaluatorType::x

Definition at line 154 of file hess_evaluator.h.

The documentation for this struct was generated from the following file:

- hess_evaluator.h

10.138 coco::hessNumber Class Reference

`#include <hessNumber.h>`

Public Member Functions

- hessNumber (void)
- hessNumber (double a)
- hessNumber (double a, double u, double v)
- hessNumber (double a, double u, double v, double h)
- hessNumber (const hessNumber &a)
- virtual ~hessNumber ()
- hessNumber & operator= (const hessNumber &a)
- hessNumber & operator+= (const hessNumber &a)
- hessNumber & operator-= (const hessNumber &a)
- hessNumber & operator*= (const hessNumber &a)
- hessNumber & operator/= (const hessNumber &a)
- hessNumber & operator+= (double a)
- hessNumber & operator-= (double a)
- hessNumber & operator*= (double a)
- hessNumber & operator/= (double a)
- double & operator[ ] (unsigned int pos)
- const double & operator[ ] (unsigned int pos) const
- double operator() (unsigned int pos) const
- double setVal (unsigned int n, double v)

Static Public Member Functions

- static hessNumber getVar (const double &x)

10.138.1 Constructor & Destructor Documentation

10.138.1.1 coco::hessNumber::hessNumber (void) [inline]

Definition at line 91 of file hessNumber.h.

10.138.1.2 coco::hessNumber::hessNumber (double *a*) [inline]

Definition at line 98 of file hessNumber.h.

10.138.1.3 coco::hessNumber::hessNumber (double *a*, double *u*, double *v*) [inline]

Definition at line 105 of file hessNumber.h.

10.138.1.4 coco::hessNumber::hessNumber (double *a*, double *u*, double *v*, double *h*) [inline]

Definition at line 113 of file hessNumber.h.

10.138.1.5 coco::hessNumber::hessNumber (const hessNumber & *a*) [inline]

Definition at line 121 of file hessNumber.h.

10.138.1.6 coco::hessNumber::∼hessNumber ( ) [inline, virtual]

Definition at line 300 of file hessNumber.h.

10.138.2 Member Function Documentation

10.138.2.1 hessNumber coco::hessNumber::getVar (const double & *x*) [inline, static]

Definition at line 151 of file hessNumber.h.

10.138.2.2 double coco::hessNumber::operator() (unsigned int *pos*) const

10.138.2.3 hessNumber & coco::hessNumber::operator∗= (const hessNumber & *a*)

Definition at line 8 of file hessNumber.cc.

10.138.2.4 hessNumber & coco::hessNumber::operator∗= (double *a*) [inline]

Definition at line 188 of file hessNumber.h.

10.138.2.5 hessNumber & coco::hessNumber::operator+= (const hessNumber & *a*) [inline]

Definition at line 160 of file hessNumber.h.

10.138.2.6 hessNumber & coco::hessNumber::operator+= (double *a*) [inline]

Definition at line 168 of file hessNumber.h.

10.138.2.7 hessNumber & coco::hessNumber::operator-= (const hessNumber & *a*) [inline]

Definition at line 174 of file hessNumber.h.

10.138.2.8 hessNumber & coco::hessNumber::operator-= (double *a*) [inline]

Definition at line 182 of file hessNumber.h.

10.138.2.9 hessNumber & coco::hessNumber::operator/= (const hessNumber & *a*)

Definition at line 17 of file hessNumber.cc.

10.138.2.10 hessNumber & coco::hessNumber::operator/= (double *a*) [inline]

Definition at line 196 of file hessNumber.h.

10.138.2.11 hessNumber & coco::hessNumber::operator= (const hessNumber & *a*) [inline]

Definition at line 127 of file hessNumber.h.

10.138.2.12 double & coco::hessNumber::operator[ ] (unsigned int *pos*) [inline]

Definition at line 143 of file hessNumber.h.

10.138.2.13 const double & coco::hessNumber::operator[ ] (unsigned int *pos*) const [inline]

Definition at line 138 of file hessNumber.h.

10.138.2.14 double coco::hessNumber::setVal (unsigned int *n*, double *v*)

Definition at line 272 of file hessNumber.cc.

The documentation for this class was generated from the following files:

- hessNumber.h
- hessNumber.cc

10.139 coco::hessPreparationEvaluator Class Reference

`#include <hess_evaluator.h>`

Inheritance diagram for coco::hessPreparationEvaluator:

Collaboration diagram for coco::hessPreparationEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- hessPreparationEvaluator (std::vector< std::vector< double > > &__d, std::vector< std::vector< double > > &__h, unsigned int _num_of_nodes)
- hessPreparationEvaluator (const hessPreparationEvaluator &__x)
- ~hessPreparationEvaluator ()
- void initialize ()
- bool is_cached (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- int update (bool __rval)
- int update (const expression_node &__data, bool __rval)
- bool calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

10.139.1 Member Typedef Documentation

10.139.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.139.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.139.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.139.2 Constructor & Destructor Documentation

10.139.2.1 coco::hessPreparationEvaluator::hessPreparationEvaluator (std::vector< std::vector< double > > & *__d*, std::vector< std::vector< double > > & *__h*, unsigned int *_num_of_nodes*) [inline]

Definition at line 86 of file hess_evaluator.h.

10.139.2.2 coco::hessPreparationEvaluator::hessPreparationEvaluator (const hessPreparationEvaluator & *__x*) [inline]

Definition at line 102 of file hess_evaluator.h.

10.139.2.3 coco::hessPreparationEvaluator::~hessPreparationEvaluator ( ) [inline]

Definition at line 105 of file hess_evaluator.h.

10.139.3 Member Function Documentation

10.139.3.1 void coco::hessPreparationEvaluator::calculate (const expression_node & *__data*) [inline]

Definition at line 126 of file hess_evaluator.h.

10.139.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.139.3.3 bool coco::hessPreparationEvaluator::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 138 of file hess_evaluator.h.

10.139.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.139.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.139.3.6 void coco::hessPreparationEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 107 of file hess_evaluator.h.

10.139.3.7 int coco::hessPreparationEvaluator::initialize (const expression_node & __data) [inline]

Definition at line 116 of file hess_evaluator.h.

10.139.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The ___data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.139.3.9 bool coco::hessPreparationEvaluator::is_cached (const expression_node & __data) [inline]

Definition at line 109 of file hess_evaluator.h.

10.139.3.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & *__data*) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.139.3.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.139.3.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.139.3.13 void coco::hessPreparationEvaluator::retrieve_from_cache (const expression_node & *__data*) [inline]

Definition at line 114 of file hess_evaluator.h.

10.139.3.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.139.3.15 int coco::hessPreparationEvaluator::update (bool *__rval*) [inline]

Definition at line 128 of file hess_evaluator.h.

10.139.3.16 int coco::hessPreparationEvaluator::update (const expression_node & *__data*, bool *__rval*) [inline]

Definition at line 130 of file hess_evaluator.h.

10.139.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.139.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.139.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.139.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.139.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.139.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- hess_evaluator.h

10.140 coco::hessPreparationEvaluatorType Struct Reference

`#include <hess_evaluator.h>`

Public Attributes

- std::vector< std::vector < double > > * d
- std::vector< std::vector < double > > * h

10.140.1 Member Data Documentation

10.140.1.1 std::vector<std::vector<double> >* coco::hessPreparationEvaluatorType::d

Definition at line 74 of file hess_evaluator.h.

10.140.1.2 std::vector<std::vector<double> >* coco::hessPreparationEvaluatorType::h

Definition at line 75 of file hess_evaluator.h.

The documentation for this struct was generated from the following file:

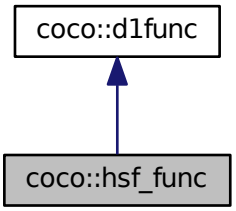
- hess_evaluator.h

10.141 coco::hsf_func Class Reference

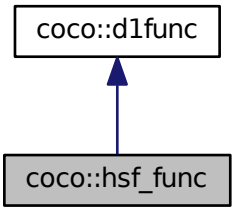
The hsf_func class (one dimensional function)

`#include <hsf.h>`

Inheritance diagram for coco::hsf_func:



Collaboration diagram for coco::hsf_func:



Public Types

- enum d1func_point_t { d1fp_unknown = -2, d1fp_regular = -1, d1fp_extremum = 0, d1fp_evenpoleup = 1, d1fp_evenpoledn = 2, d1fp_oddpoleup = 3, d1fp_oddpoledn = 4, d1fp_jumpldef = 5, d1fp_jumpludef = 6, d1fp_point = 7, d1fp_undefined = 8, d1fp_wild = 9, d1fp_jumprdef = 10, d1fp_jumprudef = 11 }
- enum d1solve_t { d1solve_0, d1solve_1, d1solve_2 }
- typedef std::triple< interval, interval, d1func_point_t > point_info
- typedef std::vector< point_info > point_list

Public Member Functions

- virtual uint64_t thash ()
- virtual uint64_t chash ()
- virtual const char * func_name () const
- virtual std::string dag_out () const
- virtual std::string print_params () const
- virtual double eval (double x, unsigned int k) const
- virtual interval imperfect_eval (const interval &x, unsigned int k) const
- virtual bool operator== (const d1func &b) const
- virtual void raise_order (unsigned int neworder)
- hsf_func (double _c, unsigned int MAXORDER=5)
- virtual int differentiability (const interval &i, unsigned int k) const
- virtual diffNumber eval (const diffNumber &x, unsigned int k) const
- virtual hessNumber eval (const hessNumber &x, unsigned int k) const
- virtual interval eval (const interval &x, unsigned int k) const
- virtual diffI eval (const diffI &x, unsigned int k) const
- virtual ihessNumber eval (const ihessNumber &x, unsigned int k) const
- virtual Islope eval (const Islope &x, unsigned int k) const
- virtual interval_set eval (const interval_set &x, unsigned int k) const
- virtual std::ostream & print_add (std::ostream &o) const
- virtual interval eval_slp (const interval &z, const interval &x, unsigned int k) const

- virtual interval eval_slp2 (const interval &z, const interval &y, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &x, unsigned int k) const
- virtual void internal_eval_d1fp (const diffI &x, diffI &res, const std::vector< d1func_point_t > &ptin, std::vector< d1func_point_t > &ptres, unsigned int order) const
- virtual interval_set intersect_inv (const interval &x, const interval &r, unsigned int k) const
- virtual interval_set intersect_inv (const interval_set &x, const interval_set &r, unsigned int k) const
- virtual std::pair< double, double > evald (double x, unsigned int k) const
- virtual std::triple< double, double, double > evalt (double x, unsigned int k) const
- virtual void evalf (double x, double *r, unsigned int k) const
- virtual convex_info convexity (const interval &i, unsigned int k) const
- virtual d1func_monotonicity_t monotonicity (const interval &i, unsigned int k) const
- virtual int degree_update (int in_degree) const
- virtual bool order_is_supported (unsigned int k) const
- virtual unsigned int supported_eval_order () const
- virtual const func_info & ff (unsigned int k) const
- virtual bool operator!= (const d1func &b) const
- const char * toString (const d1func::d1func_point_t &t) const

Public Attributes

- volatile int MAXLEN

Protected Attributes

- double dod_left
- double dod_right
- int basic_diff
- std::vector< func_info > _f

10.141.1 Detailed Description

hsf_func is a one dimensional function defined by (exp(-c*sqrt(x))-exp(c*sqrt(x)))/sqrt(x) = -2*sinh(c*sqrt(x))/sqrt(x), where c is a real constant. Its domain is]0,inf[.

10.141.2 Member Typedef Documentation

10.141.2.1 typedef std::triple<interval, interval, d1func_point_t> coco::d1func::point_info [inherited]

The information on one point .first is the point, .second the value

Definition at line 110 of file d1func.h.

10.141.2.2 typedef std::vector<point_info> coco::d1func::point_list [inherited]

a list of points for function value, first and second derivatives

Definition at line 113 of file d1func.h.

10.141.3 Member Enumeration Documentation

10.141.3.1 enum coco::d1func::d1func_point_t [inherited]

The possible classes for special points in the lists: d1fp_regular: At this point the function is regular (for internal use only!) d1fp_extremum: The point is a local extremum d1fp_evenpoleup: At this point the function has a positive even pole d1fp_evenpoledn: At this point the function has a negative even pole d1fp_oddpoleup: At this point the function has an odd pole like 1/x at 0 d1fp_oddpoledn: At this point the function has an odd pole like -1/x at 0 d1fp_jumpldef: At this point the function has a jump (left side defined) d1fp_jumpludef: At this point the function has a jump (left side not defined) d1fp_point-: At this point the function has the specified value d1fp_undefined: In the given interval the function is undefined d1fp_wild: In the given interval the function has "wild" behaviour d1fp_jumprdef: At this point the function has a jump (right side defined) d1fp_jumprudef: At this point the function has a jump (right side not defined) d1fp_unknown: At this point the function has a unknown behaviour (for internal use only!)

Enumerator:

d1fp_unknown

d1fp_regular

d1fp_extremum

d1fp_evenpoleup

d1fp_evenpoledn

d1fp_oddpoleup

d1fp_oddpoledn

d1fp_jumpldef

d1fp_jumpludef

d1fp_point

d1fp_undefined

d1fp_wild

d1fp_jumprdef

d1fp_jumprudef

Definition at line 95 of file d1func.h.

10.141.3.2 enum coco::d1func::d1solve_t [inherited]

The possible functions for 1D Newton solving: d1solve_0: f(x) = c d1solve_1: f(x) - f(z) - f'(x)(x-z) = 0 d1solve_2: 1/(x-z)(f(x)-f(z))(1+(x-w)(x-z)) - (f(w)-f(z))/(w-z) - (x-w)/(x-z) f'(x) = 0

Enumerator:

d1solve_0

d1solve_1

d1solve_2

Definition at line 106 of file d1func.h.

10.141.4 Constructor & Destructor Documentation

10.141.4.1 coco::hsf_func::hsf_func (double _c, unsigned int MAXORDER = 5) [inline]

Definition at line 377 of file hsf.h.

10.141.5 Member Function Documentation

10.141.5.1 virtual uint64_t coco::hsf_func::chash ( ) [inline, virtual]

the const hash value for hash calculation in the DAG

Implements coco::d1func.

Definition at line 54 of file hsf.h.

10.141.5.2 convex_info coco::d1func::convexity (const interval & i, unsigned int k) const [virtual, inherited]

return convexity information on `i`.

Definition at line 2308 of file d1func.cc.

10.141.5.3 virtual std::string coco::hsf_func::dag_out ( ) const [inline, virtual]

the parameters written in DAG format (it must contain necessary ':'s)

Reimplemented from coco::d1func.

Definition at line 60 of file hsf.h.

10.141.5.4 virtual int coco::d1func::degree_update (int in_degree) const [inline, virtual, inherited]

return degree information for in-argument degree `in_degree`.

Definition at line 457 of file d1func.h.

10.141.5.5 virtual int coco::hsf_func::differentiability (const interval & i, unsigned int k) const [inline, virtual]

return differentiability information on `i`.

Reimplemented from coco::d1func.

Definition at line 383 of file hsf.h.

10.141.5.6 virtual double coco::hsf_func::eval (double x, unsigned int k) const [inline, virtual]

the imperfect function evaluations

Implements coco::d1func.

Definition at line 66 of file hsf.h.

10.141.5.7 diffNumber coco::d1func::eval (const diffNumber & *x*, unsigned int *k*) const [virtual, inherited]

the diffNumber evaluation

Reimplemented in coco::dag_d1func.

Definition at line 349 of file d1func.cc.

10.141.5.8 hessNumber coco::d1func::eval (const hessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the hessNumber evaluation

Definition at line 334 of file d1func.cc.

10.141.5.9 interval coco::d1func::eval (const interval & *x_in*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 390 of file d1func.cc.

10.141.5.10 diffI coco::d1func::eval (const diffI & *x*, unsigned int *k*) const [virtual, inherited]

the interval diffNumber evaluation

Definition at line 715 of file d1func.cc.

10.141.5.11 ihessNumber coco::d1func::eval (const ihessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the interval hessNumber evaluation

the hessNumber evaluation

Definition at line 411 of file d1func.cc.

10.141.5.12 Islope coco::d1func::eval (const Islope & *x*, unsigned int *k*) const [virtual, inherited]

the interval slope evaluation

the interval diffNumber evaluation

Definition at line 429 of file d1func.cc.

10.141.5.13 interval_set coco::d1func::eval (const interval_set & *x*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 794 of file d1func.cc.

10.141.5.14 interval coco::d1func::eval_slp (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval first order slope evaluation at center z

Definition at line 1403 of file d1func.cc.

10.141.5.15 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *y*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at centers z and y

Definition at line 1984 of file d1func.cc.

10.141.5.16 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at double center z

Definition at line 1991 of file d1func.cc.

10.141.5.17 virtual std::pair<double,double> coco::d1func::evald (double *x*, unsigned int *k*) const [inline, virtual, inherited]

the point and derivative evaluation

Reimplemented in coco::xexp_func, and coco::dag_d1func.

Definition at line 437 of file d1func.h.

10.141.5.18 virtual void coco::d1func::evalf (double *x*, double * *r*, unsigned int *k*) const [inline, virtual, inherited]

the point, first to third derivative evaluation

Reimplemented in coco::xexp_func.

Definition at line 443 of file d1func.h.

10.141.5.19 virtual std::triple<double,double,double> coco::d1func::evalt (double *x*, unsigned int *k*) const [inline, virtual, inherited]

the point and first and second derivative evaluation

Reimplemented in coco::xexp_func, and coco::dag_d1func.

Definition at line 440 of file d1func.h.

10.141.5.20 virtual const func_info& coco::d1func::ff (unsigned int *k*) const [inline, virtual, inherited]

retrieve the k th special points list

Definition at line 469 of file d1func.h.

10.141.5.21 virtual const char* coco::hsf_func::func_name ( ) const [inline, virtual]

the name of the d1func.

Implements coco::d1func.

Definition at line 57 of file hsf.h.

10.141.5.22 virtual interval coco::hsf_func::imperfect_eval (const interval & *x*, unsigned int *k*) const [inline, virtual]

the imperfect function evaluations

Implements coco::d1func.

Definition at line 176 of file hsf.h.

10.141.5.23 void coco::d1func::internal_eval_d1fp (const diffI & *x*, diffI & *res*, const std::vector< d1func_point_t > & *ptin*, std::vector< d1func_point_t > & *ptres*, unsigned int *k*) const [virtual, inherited]

perform internal evaluations to update eventual point lists of "higher" functions.

Definition at line 168 of file d1func.cc.

10.141.5.24 interval_set coco::d1func::intersect_inv (const interval & *x*, const interval & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1708 of file d1func.cc.

10.141.5.25 interval_set coco::d1func::intersect_inv (const interval_set & *x*, const interval_set & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1778 of file d1func.cc.

10.141.5.26 d1func_monotonicity_t coco::d1func::monotonicity (const interval & *x*, unsigned int *k*) const [virtual, inherited]

return monotonicity information on `i`.

Definition at line 2185 of file d1func.cc.

10.141.5.27 virtual bool coco::d1func::operator!= (const d1func & *b*) const [inline, virtual, inherited]

the other comparison operator

Definition at line 476 of file d1func.h.

10.141.5.28 virtual bool coco::hsf_func::operator== (const d1func & *b*) const [inline, virtual]

the comparison operator

Reimplemented from coco::d1func.

Definition at line 319 of file hsf.h.

10.141.5.29 virtual bool coco::d1func::order_is_supported (unsigned int *k*) const [inline, virtual, inherited]

check whether evaluation of this order is supported.

Definition at line 460 of file d1func.h.

10.141.5.30 virtual std::ostream& coco::d1func::print_add (std::ostream & *o*) const [inline, virtual, inherited]

additional info written in DAG format (it must contain dag lines)

Reimplemented in coco::dag_d1func.

Definition at line 208 of file d1func.h.

10.141.5.31 virtual std::string coco::hsf_func::print_params ( ) const [inline, virtual]

the parameters written in printable form

Reimplemented from coco::d1func.

Definition at line 63 of file hsf.h.

10.141.5.32 virtual void coco::hsf_func::raise_order (unsigned int *neworder*) [inline, virtual]

raise the order of the supported function evaluation to `neworder`

Reimplemented from coco::d1func.

Definition at line 326 of file hsf.h.

10.141.5.33 virtual unsigned int coco::d1func::supported_eval_order ( ) const [inline, virtual, inherited]

return the maximal evaluation order supported.

Definition at line 466 of file d1func.h.

10.141.5.34 virtual uint64_t coco::hsf_func::thash ( ) [inline, virtual]

the pure hash value for hash calculation in the DAG

Implements coco::d1func.

Definition at line 51 of file hsf.h.

10.141.5.35 const char * coco::d1func::toString (const d1func::d1func_point_t & *t*) const [inherited]

Definition at line 2415 of file d1func.cc.

10.141.6 Member Data Documentation

10.141.6.1 std::vector<func_info> coco::d1func::_f [protected, inherited]

the function info tables for function values and derivatives.

Definition at line 163 of file d1func.h.

10.141.6.2 int coco::d1func::basic_diff [protected, inherited]

basic differentiability

Definition at line 160 of file d1func.h.

10.141.6.3 double coco::d1func::dod_left [protected, inherited]

the left limit of the DOD

Definition at line 156 of file d1func.h.

10.141.6.4 double coco::d1func::dod_right [protected, inherited]

the right limit of the DOD

Definition at line 158 of file d1func.h.

10.141.6.5 volatile int coco::d1func::MAXLEN [inherited]

maximal lenght of the interval set returned

Definition at line 167 of file d1func.h.

The documentation for this class was generated from the following file:

- hsf.h

10.142 coco::ider_eval Class Reference

Backward interval gradient evaluation with prepared interval derivative data.

`#include <ider_evaluator.h>`

Inheritance diagram for coco::ider_eval:

Collaboration diagram for coco::ider_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- ider_eval (const std::vector< interval > &__x, std::vector< std::vector< interval > > &__ider_-data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__d, std::vector< interval > &__grad)
- ider_eval (const ider_eval &__d)
- ~ider_eval ()
- void new_box (std::vector< interval > &__x, std::vector< std::vector< interval > > &__ider_data, const variable_indicator &__v)
- void new_result (std::vector< interval > &__grad)
- void set_mult (const interval &scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.142.1 Detailed Description

This class is a cached backward evaluator performing an interval gradient evaluation with available partial interval derivative information.

10.142.2 Member Typedef Documentation

10.142.2.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.142.2.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.142.2.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.142.3 Constructor & Destructor Documentation

10.142.3.1 coco::ider_eval::ider_eval (const std::vector< interval > & __x, std::vector< std::vector< interval > > & __ider_data, variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > * __d, std::vector< interval > & __grad) [inline]

Constructor: __x is the evaluation box, __ider_data contains the partial interval derivative information, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies the DAG, and __d the cache (may be NULL). Eventually, __grad is a reference to the result vector. This vector must have the correct length and has to be initialized with zero, since all components are actually computed by adding to the components of __grad.

Definition at line 1609 of file ider_evaluator.h.

10.142.3.2 coco::ider_eval::ider_eval (const ider_eval & __d) [inline]

Standard Copy Constructor

Definition at line 1625 of file ider_evaluator.h.

10.142.3.3 coco::ider_eval::~ider_eval ( ) [inline]

Standard Destructor

Definition at line 1628 of file ider_evaluator.h.

10.142.4 Member Function Documentation

10.142.4.1 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.142.4.2 int coco::ider_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1668 of file ider_evaluator.h.

10.142.4.3 bool coco::ider_eval::calculate_value (bool eval_all) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1763 of file ider_evaluator.h.

10.142.4.4 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.142.4.5 void coco::ider_eval::cleanup (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1719 of file ider_evaluator.h.

10.142.4.6 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.142.4.7 void coco::ider_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1662 of file ider_evaluator.h.

10.142.4.8 bool coco::ider_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the interval gradient for this node is already available.

Reimplemented from coco::coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1587 of file ider_evaluator.h.

10.142.4.9 void coco::ider_eval::new_box (std::vector< interval > & __x, std::vector< std::vector< interval > > & __ider_data, const variable_indicator & __v) [inline]

This method changes the evaluation box to ___x and the new interval derivative data ___ider_data. The parameter ___v specifies which variables have changed value since the last evaluation.

Definition at line 1633 of file ider_evaluator.h.

10.142.4.10 void coco::ider_eval::new_result (std::vector< interval > & __grad) [inline]

This method changes the result vector to ___grad.

Definition at line 1643 of file ider_evaluator.h.

10.142.4.11 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.142.4.12 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.142.4.13 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.142.4.14 void coco::ider_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1730 of file ider_evaluator.h.

10.142.4.15 void coco::ider_eval::set_mult (const interval & *scal*) [inline]

This method causes the gradient to be multiplied by `scal`.

Definition at line 1649 of file ider_evaluator.h.

10.142.4.16 expression_const_walker coco::ider_eval::short_cut_to (const expression_node & *__data*) [inline]

NOP version, not needed

Definition at line 1656 of file ider_evaluator.h.

10.142.4.17 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.142.4.18 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.142.4.19 int coco::ider_eval::update (const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1737 of file ider_evaluator.h.

10.142.4.20 int coco::ider_eval::update (const expression_node & *__data*, const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1744 of file ider_evaluator.h.

10.142.4.21 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 892 of file search_graph.cc.

10.142.4.22 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.142.4.23 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.142.4.24 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

The documentation for this class was generated from the following file:

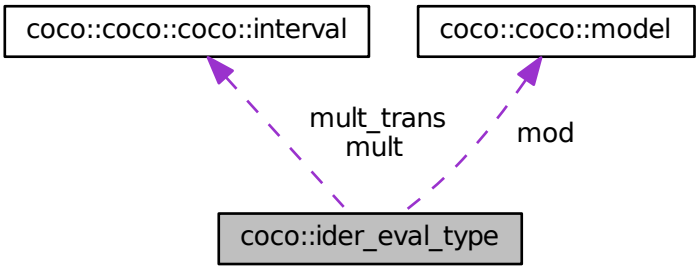
- ider_evaluator.h

10.143 coco::ider_eval_type Struct Reference

Visitor data for ider_eval.

`#include <ider_evaluator.h>`

Collaboration diagram for coco::ider_eval_type:



Public Attributes

- std::vector< std::vector < interval > > * id_data
- std::vector< std::vector < interval > > * d_cache
- std::vector< interval > * grad_vec
- const std::vector< interval > * x
- const model * mod
- interval mult
- interval mult_trans
- unsigned int child_n

10.143.1 Detailed Description

This class is the visitor data type for the ider_eval evaluator.

10.143.2 Member Data Documentation

10.143.2.1 unsigned int coco::ider_eval_type::child_n

children counter

Definition at line 1566 of file ider_evaluator.h.

10.143.2.2 std::vector<std::vector<interval> >* coco::ider_eval_type::d_cache

interval derivative cache

Definition at line 1560 of file ider_evaluator.h.

10.143.2.3 std::vector<interval>* coco::ider_eval_type::grad_vec

the interval gradient vector

Definition at line 1561 of file ider_evaluator.h.

10.143.2.4 std::vector<std::vector<interval> >* coco::ider_eval_type::id_data

partial interval derivative data

Definition at line 1559 of file ider_evaluator.h.

10.143.2.5 const model* coco::ider_eval_type::mod

the DAG

Definition at line 1563 of file ider_evaluator.h.

10.143.2.6 interval coco::ider_eval_type::mult

the current value on the path

Definition at line 1564 of file ider_evaluator.h.

10.143.2.7 interval coco::ider_eval_type::mult_trans

transfer variable for mult

Definition at line 1565 of file ider_evaluator.h.

10.143.2.8 const std::vector<interval>* coco::ider_eval_type::x

the evaluation box

Definition at line 1562 of file ider_evaluator.h.

The documentation for this struct was generated from the following file:

- ider_evaluator.h

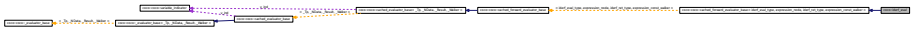
10.144 coco::iderf_eval Class Reference

Forward function and derivative range evaluation.

`#include <iderf_evaluator.h>`

Inheritance diagram for coco::iderf_eval:

Collaboration diagram for coco::iderf_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- iderf_eval (const std::vector< interval > &__x, const std::vector< interval > &__rg, const variable_indicator &__v, const model &__m, std::vector< iderf_ret_type > *__c, bool do_i=true)
- iderf_eval (const iderf_eval &__v)
- ~iderf_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_box (const std::vector< interval > &__x, const variable_indicator &__v)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const iderf_ret_type &__rval)
- int update (const expression_node &__data, const iderf_ret_type &__rval)
- iderf_ret_type calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.144.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural interval extension together with a derivative range evaluation.

10.144.2 Member Typedef Documentation

10.144.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.144.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.144.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.144.3 Constructor & Destructor Documentation

10.144.3.1 coco::iderf_eval::iderf_eval (const std::vector< interval > & __x, const std::vector< interval > & __rg, const variable_indicator & __v, const model & __m, std::vector< iderf_ret_type > * __c, bool do_i = true) [inline]

Constructor: ___x is the box over which the function range shall be evaluated, ___v is a variable indicator specifying the variables that have changed value since the last evaluation, ___m specifies the DAG, and ___c the cache (may be NULL). The boolean do_i determines whether the natural interval extension will use known node ranges for reducing the ranges.

Definition at line 156 of file iderf_evaluator.h.

10.144.3.2 coco::iderf_eval::iderf_eval (const iderf_eval & __v) [inline]

Standard Copy Constructor

Definition at line 172 of file iderf_evaluator.h.

10.144.3.3 coco::iderf_eval::~iderf_eval ( ) [inline]

Standard Destructor

Definition at line 175 of file iderf_evaluator.h.

10.144.4 Member Function Documentation

10.144.4.1 void coco::iderf_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 256 of file iderf_evaluator.h.

10.144.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.144.4.3 iderf_ret_type coco::iderf_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 901 of file iderf_evaluator.h.

10.144.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.144.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.144.4.6 void coco::iderf_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 192 of file iderf_evaluator.h.

10.144.4.7 int coco::iderf_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 194 of file iderf_evaluator.h.

10.144.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.144.4.9 bool coco::iderf_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the range for this node is already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 101 of file iderf_evaluator.h.

10.144.4.10 void coco::iderf_eval::new_box (const std::vector< interval > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation box to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 184 of file iderf_evaluator.h.

10.144.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.144.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.144.4.13 void coco::iderf_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 271 of file iderf_evaluator.h.

10.144.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.144.4.15 expression_const_walker coco::iderf_eval::short_cut_to (const expression_node & *__data*) [inline]

NOP version, not needed

Definition at line 178 of file iderf_evaluator.h.

10.144.4.16 int coco::iderf_eval::update (const iderf_ret_type & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 293 of file iderf_evaluator.h.

10.144.4.17 int coco::iderf_eval::update (const expression_node & *__data*, const iderf_ret_type & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 299 of file iderf_evaluator.h.

10.144.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.144.4.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.144.4.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.144.4.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.144.4.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.144.4.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

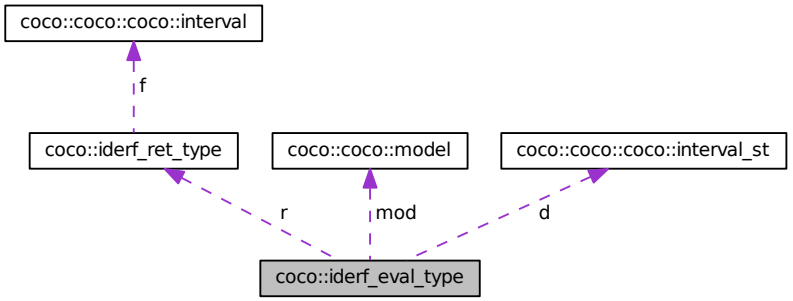
- iderf_evaluator.h

10.145 coco::iderf_eval_type Struct Reference

Visitor data for iderf_eval.

```
#include <iderf_evaluator.h>
```

Collaboration diagram for coco::iderf_eval_type:



Public Attributes

- const std::vector< interval > * x
- const std::vector< interval > * range
- std::vector< iderf_ret_type > * cache
- const model * mod
- union {
  void * p
  interval_st d
  int info
  } u
- iderf_ret_type r
- unsigned int n
- bool do_intersect

10.145.1 Detailed Description

This class is the visitor data type for the iderf_eval evaluator.

10.145.2 Member Data Documentation

10.145.2.1 std::vector<iderf_ret_type>* coco::iderf_eval_type::cache

the function range cache

Definition at line 68 of file iderf_evaluator.h.

10.145.2.2 interval_st coco::iderf_eval_type::d

Definition at line 72 of file iderf_evaluator.h.

10.145.2.3 bool coco::iderf_eval_type::do_intersect

compute an intersection with the node range?

Definition at line 76 of file iderf_evaluator.h.

10.145.2.4 int coco::iderf_eval_type::info

Definition at line 72 of file iderf_evaluator.h.

10.145.2.5 const model* coco::iderf_eval_type::mod

the DAG

Definition at line 71 of file iderf_evaluator.h.

10.145.2.6 unsigned int coco::iderf_eval_type::n

children counter

Definition at line 75 of file iderf_evaluator.h.

10.145.2.7 void* coco::iderf_eval_type::p

Definition at line 72 of file iderf_evaluator.h.

10.145.2.8 iderf_ret_type coco::iderf_eval_type::r

return value

Definition at line 74 of file iderf_evaluator.h.

10.145.2.9 const std::vector<interval>* coco::iderf_eval_type::range

the ranges of all nodes

Definition at line 67 of file iderf_evaluator.h.

10.145.2.10 union { ... } coco::iderf_eval_type::u

additional data for complex nodes

10.145.2.11 const std::vector<interval>* coco::iderf_eval_type::x

the box

Definition at line 66 of file iderf_evaluator.h.

The documentation for this struct was generated from the following file:

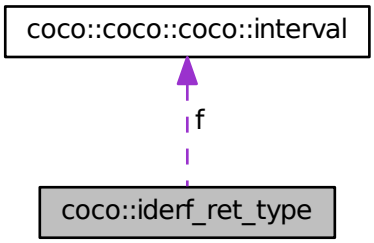
- iderf_evaluator.h

10.146 coco::iderf_ret_type Struct Reference

Visitor data for iderf_eval return value.

`#include <iderf_evaluator.h>`

Collaboration diagram for coco::iderf_ret_type:



Public Attributes

- interval f
- vmtl::sparse_vector< interval > g

10.146.1 Detailed Description

This class is the visitor data type for the return type of the iderf_eval evaluator.

10.146.2 Member Data Documentation

10.146.2.1 interval coco::iderf_ret_type::f

Definition at line 51 of file iderf_evaluator.h.

10.146.2.2 vmtl::sparse_vector<interval> coco::iderf_ret_type::g

Definition at line 52 of file iderf_evaluator.h.

The documentation for this struct was generated from the following file:

- iderf_evaluator.h

10.147 coco::ie_return_type Class Reference

The return class of all inference engines.

`#include <ie_rettype.h>`

Public Member Functions

- ie_return_type ()
- ie_return_type (const termination_reason &__t)
- ie_return_type (const delta_base *__d, double __w)
- ie_return_type (const delta_base *__d, double __w, const certificate_base *__c)
- ie_return_type (const std::pair< delta_base *, double > &__d)
- ie_return_type (const std::triple< delta_base *, double, certificate_base * > &__d)
- ie_return_type (delta __d, double __w)
- ie_return_type (delta __d, double __w, certificate __c)
- ie_return_type (const std::pair< delta, double > &__d)
- ie_return_type (const std::triple< delta, double, certificate > &__d)
- ie_return_type (const std::list< delta > &__d, const std::list< double > &__w)
- ie_return_type (const std::list< delta > &__d, const std::list< double > &__w, const std::list< certificate > &__c)
- ie_return_type (const delta_base *__d, double __w, const termination_reason &__t)
- ie_return_type (const delta_base *__d, double __w, const certificate_base *__c, const termination_reason &__t)
- ie_return_type (const std::pair< delta_base *, double > &__d, const termination_reason &__t)
- ie_return_type (const std::triple< delta_base *, double, certificate_base * > &__d, const termination_reason &__t)
- ie_return_type (delta __d, double __w, const termination_reason &__t)
- ie_return_type (delta __d, double __w, certificate __c, const termination_reason &__t)
- ie_return_type (const std::pair< delta, double > &__d, const termination_reason &__t)
- ie_return_type (const std::triple< delta, double, certificate > &__d, const termination_reason &__t)
- ie_return_type (const std::list< delta > &__d, const std::list< double > &__w, const termination_reason &__t)
- ie_return_type (const std::list< delta > &__d, const std::list< double > &__w, const std::list< certificate > &__c, const termination_reason &__t)
- ie_return_type (const ie_return_type &__r)
- virtual ~ie_return_type ()
- void set_termination_reason (const termination_reason &__t)
- const termination_reason & term_reason () const
- ie_return_type operator+ (const std::pair< delta_base *, double > &__d)
- ie_return_type & operator+= (const std::pair< delta_base *, double > &__d)
- ie_return_type operator+ (const std::triple< delta_base *, double, certificate_base * > &__d)
- ie_return_type & operator+= (const std::triple< delta_base *, double, certificate_base * > &__d)
- ie_return_type & operator= (const ie_return_type &__d)
- unsigned int n_deltas () const
- const std::list< delta_id > & get (work_node &wn, double thresh=-COCO_INF)
- void getd (std::list< delta > &dels, std::list< certificate > &certl, double thresh)

- const info_contents & get_info () const

- bool set_information (const std::string &iname, const basic_alltype &a, bool force=false)
- bool set_information (const char *iname, const basic_alltype &a, bool force=false)

- bool set_information_i (const std::string &iname, int i, const basic_alltype &a, bool force=false)
- bool set_information_i (const char *iname, int i, const basic_alltype &a, bool force=false)

- const basic_alltype & information (const std::string &iname) const
- const basic_alltype & information (const char *iname) const

- const basic_alltype & information (const std::string &iname, int i) const
- const basic_alltype & information (const char *iname, int i) const

- bool has_information (const std::string &__s) const
- bool has_information (const char *__s) const

- bool has_information (const std::string &__s, int i) const
- bool has_information (const char *__s, int i) const

- bool information_indices_set (const std::string &__s, std::vector< int > &__idx) const
- bool information_indices_set (const char *__s, std::vector< int > &__idx) const

- void unset_information (const std::string &__s)
- void unset_information (const char *__s)

- void unset_information (const std::string &__s, int i)
- void unset_information (const char *__s, int i)

- template<class _TS >
  bool retrieve_from_info (const std::string &__s, const std::vector< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info (const std::string &__s, const vmtl::sparse_vector< _TS > *&__b) const

- template<class _TS >
  bool retrieve_from_info (const std::string &__s, const vmtl::dense_matrix< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info (const std::string &__s, const vmtl::sparse_matrix< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info (const std::string &__s, _TS &__b) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, const std::vector< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, const vmtl::sparse_vector< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, const vmtl::dense_matrix< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, const vmtl::sparse_matrix< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, _TS &__b) const

- template<class _TS >
  bool retrieve_from_info (const std::string &__s, const std::vector< _TS > *&__b, const std::vector< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info (const std::string &__s, const vmtl::sparse_vector< _TS > *&__b, const vmtl::sparse_vector< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info (const std::string &__s, const vmtl::dense_matrix< _TS > *&__b, const vmtl::dense_matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info (const std::string &__s, const vmtl::sparse_matrix< _TS > *&__b, const vmtl::sparse_matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info (const std::string &__s, _TS &__b, const _TS &__def) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, const std::vector< _TS > *&__b, const std::vector< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, const vmtl::sparse_vector< _TS > *&__b, const vmtl::sparse_vector< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, const vmtl::dense_matrix< _TS > *&__b, const vmtl::dense_matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, const vmtl::sparse_matrix< _TS > *&__b, const vmtl::sparse_matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info (const char *__s, _TS &__b, const _TS &__def) const

- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, const std::vector< _TS > *&__b) const

- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, const vmtl::sparse_vector< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, const vmtl::dense_matrix< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, const vmtl::sparse_matrix< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, _TS &__b) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, const std::vector< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, const vmtl::sparse_vector< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, const vmtl::dense_matrix< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, const vmtl::sparse_matrix< _TS > *&__b) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, _TS &__b) const

- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, const std::vector< _TS > *&__b, const std::vector< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, const vmtl::sparse_vector< _TS > *&__b, const vmtl::sparse_vector< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, const vmtl::dense_matrix< _TS > *&__b, const vmtl::dense_matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, const vmtl::sparse_matrix< _TS > *&__b, const vmtl::sparse_matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const std::string &__s, int i, _TS &__b, const _TS &__def) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, const std::vector< _TS > *&__b, const std::vector< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, const vmtl::sparse_vector< _TS > *&__b, const vmtl::sparse_vector< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, const vmtl::dense_matrix< _TS > *&__b, const vmtl::dense_matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, const vmtl::sparse_matrix< _TS > *&__b, const vmtl::sparse_matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, _TS &__b, const _TS &__def) const

10.147.1 Detailed Description

This class is returned by the `infer` method of all COCONUT inference engines. It holds the reason of termination (the return code) of the engine, the deltas with weights and certificates calculated by the engine, and additional information the engine provides.

10.147.2 Constructor & Destructor Documentation

10.147.2.1 coco::ie_return_type::ie_return_type ( ) [inline]

Standard Constructor

Definition at line 71 of file ie_rettype.h.

10.147.2.2 coco::ie_return_type::ie_return_type (const termination_reason & __t) [inline]

Constructor, which sets the termination_reason to ___t

Definition at line 73 of file ie_rettype.h.

10.147.2.3 coco::ie_return_type::ie_return_type (const delta_base * __d, double __w) [inline]

Constructor, which initializes the ie_return_type with one delta ___d and the corresponding weight ___w.

Definition at line 77 of file ie_rettype.h.

10.147.2.4 coco::ie_return_type::ie_return_type (const delta_base * __d, double __w, const certificate_base * __c) [inline]

Constructor, which initializes the ie_return_type with one delta ___d, the corresponding weight ___w and certificate ___c.

Definition at line 83 of file ie_rettype.h.

10.147.2.5 coco::ie_return_type::ie_return_type (const std::pair< delta_base *, double > & __d) [inline]

Constructor, which initializes the ie_return_type with one delta and the corresponding weight collected in a pair ___d.

Definition at line 90 of file ie_rettype.h.

10.147.2.6 coco::ie_return_type::ie_return_type (const std::triple< delta_base *, double, certificate_base * > & __d) [inline]

Constructor, which initializes the ie_return_type with one delta, the corresponding weight and certificate collected in a triple ___d.

Definition at line 96 of file ie_rettype.h.

10.147.2.7 coco::ie_return_type::ie_return_type (delta __d, double __w) [inline]

Constructor, which initializes the ie_return_type with one delta ___d and the corresponding weight ___w.

Definition at line 103 of file ie_rettype.h.

10.147.2.8 coco::ie_return_type::ie_return_type (delta __d, double __w, certificate __c) [inline]

Constructor, which initializes the ie_return_type with one delta ___d, the corresponding weight ___w and certificate ___c.

Definition at line 109 of file ie_rettype.h.

10.147.2.9 coco::ie_return_type::ie_return_type (const std::pair< delta, double > & __d) [inline]

Constructor, which initializes the ie_return_type with one delta and the corresponding weight collected in a pair ___d.

Definition at line 114 of file ie_rettype.h.

10.147.2.10 coco::ie_return_type::ie_return_type (const std::triple< delta, double, certificate > & __d) [inline]

Constructor, which initializes the ie_return_type with one delta, the corresponding weight and certificate collected in a triple ___d.

Definition at line 119 of file ie_rettype.h.

10.147.2.11 coco::ie_return_type::ie_return_type (const std::list< delta > & __d, const std::list< double > & __w) [inline]

Constructor, which initializes the ie_return_type with a list of deltas and the corresponding list of weights.

Definition at line 124 of file ie_rettype.h.

10.147.2.12 coco::ie_return_type::ie_return_type (const std::list< delta > & __d, const std::list< double > & __w, const std::list< certificate > & __c) [inline]

Constructor, which initializes the ie_return_type with a list of deltas and the corresponding lists of weights and certificates.

Definition at line 136 of file ie_rettype.h.

10.147.2.13 coco::ie_return_type::ie_return_type (const delta_base * __d, double __w, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with one delta ___d, the corresponding weight ___w, and the reason of termination ___t.

Definition at line 147 of file ie_rettype.h.

10.147.2.14 coco::ie_return_type::ie_return_type (const delta_base * __d, double __w, const certificate_base * __c, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with one delta ___d, the corresponding weight ___w and certificate ___c, and the reason of termination ___t.

Definition at line 155 of file ie_rettype.h.

10.147.2.15 coco::ie_return_type::ie_return_type (const std::pair< delta_base *, double > & __d, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with one delta and the corresponding weight collected in a pair ___d, and the reason of termination ___t.

Definition at line 163 of file ie_rettype.h.

10.147.2.16 coco::ie_return_type::ie_return_type (const std::triple< delta_base *, double, certificate_base * > & __d, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with one delta, the corresponding weight and certificate collected in a triple ___d, and the reason of termination ___t.

Definition at line 171 of file ie_rettype.h.

10.147.2.17 coco::ie_return_type::ie_return_type (delta __d, double __w, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with one delta ___d, the corresponding weight ___w, and the reason of termination ___t.

Definition at line 179 of file ie_rettype.h.

10.147.2.18 coco::ie_return_type::ie_return_type (delta __d, double __w, certificate __c, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with one delta ___d, the corresponding weight ___w and certificate ___c, and the reason of termination ___t.

Definition at line 186 of file ie_rettype.h.

10.147.2.19 coco::ie_return_type::ie_return_type (const std::pair< delta, double > & __d, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with one delta and the corresponding weight collected in a pair ___d, and the reason of termination ___t.

Definition at line 194 of file ie_rettype.h.

10.147.2.20 coco::ie_return_type::ie_return_type (const std::triple< delta, double, certificate > & __d, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with one delta, the corresponding weight and certificate collected in a triple ___d, and the reason of termination ___t.

Definition at line 201 of file ie_rettype.h.

10.147.2.21 coco::ie_return_type::ie_return_type (const std::list< delta > & __d, const std::list< double > & __w, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with a list of deltas and the corresponding list of weights, and the reason of termination ___t.

Definition at line 208 of file ie_rettype.h.

10.147.2.22 coco::ie_return_type::ie_return_type (const std::list< delta > & __d, const std::list< double > & __w, const std::list< certificate > & __c, const termination_reason & __t) [inline]

Constructor, which initializes the ie_return_type with a list of deltas and the corresponding lists of weights and certificates, and the reason of termination ___t.

Definition at line 222 of file ie_rettype.h.

10.147.2.23 coco::ie_return_type::ie_return_type (const ie_return_type & __r) [inline]

Standard Copy Constructor

Definition at line 233 of file ie_rettype.h.

10.147.2.24 virtual coco::ie_return_type::~ie_return_type ( ) [inline, virtual]

Standard Destructor

Definition at line 238 of file ie_rettype.h.

10.147.3 Member Function Documentation

10.147.3.1 const std::list< delta_id > & coco::ie_return_type::get (work_node & wn, double thresh = -COCO_INF)

This method gets all deltas with weight greater or equal thresh. A delta can be got at most once from any of and . The method converts and stores the delta in the deltas table of the search database. A second call to get or getd returns only deltas which have not previously been got. The work_node wn becomes the **keeper** work node of the retrieved deltas.

Definition at line 34 of file ie_rettype.cc.

10.147.3.2 const info_contents & coco::ie_return_type::get_info ( ) const [inline]

This method accesses the whole info content.

Definition at line 35 of file ie_rettype.hpp.

10.147.3.3 void coco::ie_return_type::getd (std::list< delta > & dels, std::list< certificate > & certl, double thresh)

This method gets all deltas with weight greater or equal thresh. A delta can be got at most once from any of and . A second call to either of get and returns only deltas which have not previously been got. The deltas and corresponding certificates are appended to the lists dels and certs, respectively.

Definition at line 74 of file ie_rettype.cc.

10.147.3.4 bool coco::ie_return_type::has_information (const std::string & __s) const [inline]

This method checks whether the variable iname is defined in the information structure.

Definition at line 168 of file ie_rettype.hpp.

10.147.3.5 bool coco::ie_return_type::has_information (const char * __s) const [inline]

This method checks whether the variable iname is defined in the information structure.

Definition at line 173 of file ie_rettype.hpp.

10.147.3.6 bool coco::ie_return_type::has_information (const std::string & __s, int i) const [inline]

This method checks whether the variable iname with index i is defined in the information structure.

Definition at line 178 of file ie_rettype.hpp.

10.147.3.7 bool coco::ie_return_type::has_information (const char * __s, int i) const [inline]

This method checks whether the variable iname with index i is defined in the information structure.

Definition at line 183 of file ie_rettype.hpp.

10.147.3.8 const basic_alltype & coco::ie_return_type::information (const std::string & iname) const [inline]

This method retrieves the value of the variable iname from the information structure.

Definition at line 144 of file ie_rettype.hpp.

10.147.3.9 const basic_alltype & coco::ie_return_type::information (const char * iname) const [inline]

This method retrieves the value of the variable iname from the information structure.

Definition at line 150 of file ie_rettype.hpp.

10.147.3.10 const basic_alltype & coco::ie_return_type::information (const std::string & iname, int i) const [inline]

This method retrieves the value of the variable iname with index i from the information structure.

Definition at line 156 of file ie_rettype.hpp.

10.147.3.11 const basic_alltype & coco::ie_return_type::information (const char * iname, int i) const [inline]

This method retrieves the value of the variable iname with index i from the information structure.

Definition at line 162 of file ie_rettype.hpp.

10.147.3.12 bool coco::ie_return_type::information_indices_set (const std::string & __s, std::vector< int > & __idx) const [inline]

The information_indices_set methods set __idx to the list of indices defined for the variable with name __s. The methods return whether the variable __s is defined.

Definition at line 188 of file ie_rettype.hpp.

10.147.3.13 bool coco::ie_return_type::information_indices_set (const char * __s, std::vector< int > & __idx) const [inline]

The information_indices_set methods set ___idx to the list of indices defined for the variable with name ___s. The methods return whether the variable ___s is defined.

Definition at line 194 of file ie_rettype.hpp.

10.147.3.14 unsigned int coco::ie_return_type::n_deltas ( ) const [inline]

This method returns the number of deltas defined.

Definition at line 544 of file ie_rettype.h.

10.147.3.15 ie_return_type coco::ie_return_type::operator+ (const std::pair< delta_base *, double > & __d) [inline]

This operator adds another delta with weight to the ie_return_type.

Definition at line 50 of file ie_rettype.hpp.

10.147.3.16 ie_return_type coco::ie_return_type::operator+ (const std::triple< delta_base *, double, certificate_base * > & __d) [inline]

This operator adds another delta with weight and certificate to the ie_return_type.

Definition at line 71 of file ie_rettype.hpp.

10.147.3.17 ie_return_type & coco::ie_return_type::operator+= (const std::pair< delta_base *, double > & __d) [inline]

This operator increments the ie_return_type with another delta with weight.

Definition at line 60 of file ie_rettype.hpp.

10.147.3.18 ie_return_type & coco::ie_return_type::operator+= (const std::triple< delta_base *, double, certificate_base * > & __d) [inline]

This operator increments the ie_return_type with another delta with weight and certificate.

Definition at line 81 of file ie_rettype.hpp.

10.147.3.19 ie_return_type & coco::ie_return_type::operator= (const ie_return_type & __d) [inline]

Standard Assignment Operator

Definition at line 107 of file ie_rettype.hpp.

10.147.3.20 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const std::vector< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable ___s of the information structure in ___b. If the variable is undefined the method returns false. Note that the simple types are assigned but for

the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 221 of file ie_rettype.hpp.

10.147.3.21 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const vmtl::sparse_vector< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 228 of file ie_rettype.hpp.

10.147.3.22 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const vmtl::dense_matrix< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 235 of file ie_rettype.hpp.

10.147.3.23 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const vmtl::sparse_matrix< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 242 of file ie_rettype.hpp.

10.147.3.24 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, _TS & __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 249 of file ie_rettype.hpp.

10.147.3.25 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const std::vector< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 294 of file ie_rettype.hpp.

10.147.3.26 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const vmtl::sparse_vector< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 301 of file ie_rettype.hpp.

10.147.3.27 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const vmtl::dense_matrix< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 308 of file ie_rettype.hpp.

10.147.3.28 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const vmtl::sparse_matrix< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 315 of file ie_rettype.hpp.

10.147.3.29 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, _TS & __b) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 322 of file ie_rettype.hpp.

10.147.3.30 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const std::vector< _TS > *& __b, const std::vector< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 256 of file ie_rettype.hpp.

10.147.3.31 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const vmtl::sparse_vector< _TS > *& __b, const vmtl::sparse_vector< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 263 of file ie_rettype.hpp.

10.147.3.32 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const vmtl::dense_matrix< _TS > *& __b, const vmtl::dense_matrix< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 271 of file ie_rettype.hpp.

10.147.3.33 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const vmtl::sparse_matrix< _TS > *& __b, const vmtl::sparse_matrix< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 279 of file ie_rettype.hpp.

10.147.3.34 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, _TS & __b, const _TS & __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 287 of file ie_rettype.hpp.

10.147.3.35 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const std::vector< _TS > *& __b, const std::vector< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 328 of file ie_rettype.hpp.

10.147.3.36 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const vmtl::sparse_vector< _TS > *& __b, const vmtl::sparse_vector< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 335 of file ie_rettype.hpp.

10.147.3.37 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const vmtl::dense_matrix< _TS > *& __b, const vmtl::dense_matrix< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 343 of file ie_rettype.hpp.

10.147.3.38 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const vmtl::sparse_matrix< _TS > *& __b, const vmtl::sparse_matrix< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 351 of file ie_rettype.hpp.

10.147.3.39 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, _TS & __b, const _TS & __def) const [inline]

The retrieve_from_info method stores the contents of variable __s of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 359 of file ie_rettype.hpp.

10.147.3.40 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const std::vector< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned

but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 366 of file ie_rettype.hpp.

10.147.3.41 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const vmtl::sparse_vector< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable ___s with index i of the information structure in ___b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 373 of file ie_rettype.hpp.

10.147.3.42 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const vmtl::dense_matrix< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable ___s with index i of the information structure in ___b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 380 of file ie_rettype.hpp.

10.147.3.43 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const vmtl::sparse_matrix< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable ___s with index i of the information structure in ___b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 387 of file ie_rettype.hpp.

10.147.3.44 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, _TS & __b) const [inline]

The retrieve_from_info method stores the contents of variable ___s with index i of the information structure in ___b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 394 of file ie_rettype.hpp.

10.147.3.45 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const std::vector< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable ___s with index i of the information structure in ___b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 439 of file ie_rettype.hpp.

10.147.3.46 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const vmtl::sparse_vector< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 446 of file ie_rettype.hpp.

10.147.3.47 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const vmtl::dense_matrix< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 453 of file ie_rettype.hpp.

10.147.3.48 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const vmtl::sparse_matrix< _TS > *& __b) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 460 of file ie_rettype.hpp.

10.147.3.49 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, _TS & __b) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method returns false. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations.

Definition at line 467 of file ie_rettype.hpp.

10.147.3.50 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const std::vector< _TS > *& __b, const std::vector< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 401 of file ie_rettype.hpp.

10.147.3.51 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const vmtl::sparse_vector< _TS > *& __b, const vmtl::sparse_vector< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 408 of file ie_rettype.hpp.

10.147.3.52 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const vmtl::dense_matrix< _TS > *& __b, const vmtl::dense_matrix< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 416 of file ie_rettype.hpp.

10.147.3.53 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const vmtl::sparse_matrix< _TS > *& __b, const vmtl::sparse_matrix< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 424 of file ie_rettype.hpp.

10.147.3.54 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, _TS & __b, const _TS & __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 432 of file ie_rettype.hpp.

10.147.3.55 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const std::vector< _TS > *& __b, const std::vector< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 474 of file ie_rettype.hpp.

10.147.3.56 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const vmtl::sparse_vector< _TS > *& __b, const vmtl::sparse_vector< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 481 of file ie_rettype.hpp.

10.147.3.57 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const vmtl::dense_matrix< _TS > *& __b, const vmtl::dense_matrix< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 489 of file ie_rettype.hpp.

10.147.3.58 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const vmtl::sparse_matrix< _TS > *& __b, const vmtl::sparse_matrix< _TS > * __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 497 of file ie_rettype.hpp.

10.147.3.59 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, _TS & __b, const _TS & __def) const [inline]

The retrieve_from_info method stores the contents of variable __s with index i of the information structure in __b. If the variable is undefined the method assigns the default value __def to __b. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the information structure, in order to reduce unnecessary big copy operations. The method returns true if any assignment to __b was successful.

Definition at line 505 of file ie_rettype.hpp.

10.147.3.60 bool coco::ie_return_type::set_information (const std::string & iname, const basic_alltype & a, bool force = false) [inline]

This method sets the variable iname in the information structure to a. If force is true the variable will be overwritten if it is already set. The method returns whether the set operation was successful.

Definition at line 120 of file ie_rettype.hpp.

10.147.3.61 bool coco::ie_return_type::set_information (const char * *iname*, const basic_alltype & *a*, bool *force* = false) [inline]

This method sets the variable iname in the information structure to a. If force is true the variable will be overwritten if it is already set. The method returns whether the set operation was successful.

Definition at line 126 of file ie_rettype.hpp.

10.147.3.62 bool coco::ie_return_type::set_information_i (const std::string & *iname*, int *i*, const basic_alltype & *a*, bool *force* = false) [inline]

This method sets the variable iname with index i in the information structure to a. If force is true the variable will be overwritten if it is already set. The method returns whether the set operation was successful.

Definition at line 132 of file ie_rettype.hpp.

10.147.3.63 bool coco::ie_return_type::set_information_i (const char * *iname*, int *i*, const basic_alltype & *a*, bool *force* = false) [inline]

This method sets the variable iname with index i in the information structure to a. If force is true the variable will be overwritten if it is already set. The method returns whether the set operation was successful.

Definition at line 138 of file ie_rettype.hpp.

10.147.3.64 void coco::ie_return_type::set_termination_reason (const termination_reason & *__t*) [inline]

This method sets the termination_reason to __t.

Definition at line 40 of file ie_rettype.hpp.

10.147.3.65 const termination_reason & coco::ie_return_type::term_reason ( ) const [inline]

This method returns the reason of termination.

Definition at line 45 of file ie_rettype.hpp.

10.147.3.66 void coco::ie_return_type::unset_information (const std::string & *__s*) [inline]

This method removes the variable __s from the information structure.

Definition at line 200 of file ie_rettype.hpp.

10.147.3.67 void coco::ie_return_type::unset_information (const char * *__s*) [inline]

This method removes the variable __s from the information structure.

Definition at line 205 of file ie_rettype.hpp.

10.147.3.68 void coco::ie_return_type::unset_information (const std::string & *__s*, int *i*) [inline]

This method removes the variable __s with index i from the information structure.

Definition at line 210 of file ie_rettype.hpp.

10.147.3.69 void coco::ie_return_type::unset_information (const char * __s, int i) [inline]

This method removes the variable __s with index i from the information structure.

Definition at line 215 of file ie_rettype.hpp.

The documentation for this class was generated from the following files:

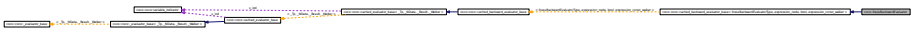
- ie_rettype.h
- ie_rettype.cc
- ie_rettype.hpp

10.148 coco::ihessBackwardEvaluator Class Reference

`#include <ihess_evaluator.h>`

Inheritance diagram for coco::ihessBackwardEvaluator:

Collaboration diagram for coco::ihessBackwardEvaluator:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- ihessBackwardEvaluator (std::vector< std::vector< interval > > *__der_data, std::vector< std::vector< interval > > *__hess_data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__d, std::vector< std::vector< interval > > *__h, std::vector< interval > *__grad, std::vector< interval > *__hess)
- ihessBackwardEvaluator (const ihessBackwardEvaluator &__he)
- ~ihessBackwardEvaluator ()
- void newPoint (std::vector< std::vector< interval > > *__der_data, std::vector< std::vector< interval > > *__hess_data, const variable_indicator &__v)
- void newResult (std::vector< interval > *__grad, std::vector< interval > *__hess)
- void set_mult (interval scal)
- expression_const_walker short_cut_to (const expression_node &__data)

- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.148.1 Member Typedef Documentation

10.148.1.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.148.1.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.148.1.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.148.2 Constructor & Destructor Documentation

10.148.2.1 coco::ihessBackwardEvaluator::ihessBackwardEvaluator (std::vector< std::vector< interval > > * __der_data, std::vector< std::vector< interval > > * __hess_data, variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > * __d, std::vector< std::vector< interval > > * __h, std::vector< interval > * __grad, std::vector< interval > * __hess) [inline]

Definition at line 1444 of file ihess_evaluator.h.

10.148.2.2 coco::ihessBackwardEvaluator::ihessBackwardEvaluator (const ihessBackwardEvaluator & __he) [inline]

Definition at line 1474 of file ihess_evaluator.h.

10.148.2.3 coco::ihessBackwardEvaluator::~ihessBackwardEvaluator ( ) [inline]

Definition at line 1476 of file ihess_evaluator.h.

10.148.3 Member Function Documentation

10.148.3.1 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.148.3.2 int coco::ihessBackwardEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 1516 of file ihess_evaluator.h.

10.148.3.3 bool coco::ihessBackwardEvaluator::calculate_value (bool eval_all) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1755 of file ihess_evaluator.h.

10.148.3.4 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.148.3.5 void coco::ihessBackwardEvaluator::cleanup (const expression_node & __data) [inline]

Definition at line 1653 of file ihess_evaluator.h.

10.148.3.6 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.148.3.7 void coco::ihessBackwardEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1505 of file ihess_evaluator.h.

10.148.3.8 bool coco::ihessBackwardEvaluator::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1431 of file ihess_evaluator.h.

10.148.3.9 void coco::ihessBackwardEvaluator::newPoint (std::vector< std::vector< interval > > * __der_data, std::vector< std::vector< interval > > * __hess_data, const variable_indicator & __v) [inline]

Definition at line 1478 of file ihess_evaluator.h.

10.148.3.10 void coco::ihessBackwardEvaluator::newResult (std::vector< interval > * __grad, std::vector< interval > * __hess) [inline]

Definition at line 1488 of file ihess_evaluator.h.

10.148.3.11 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.148.3.12 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.148.3.13 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.148.3.14 void coco::ihessBackwardEvaluator::set_mult (interval scal) [inline]

Definition at line 1494 of file ihess_evaluator.h.

10.148.3.15 expression_const_walker coco::ihessBackwardEvaluator::short_cut_to (const expression_node & __data) [inline]

Definition at line 1501 of file ihess_evaluator.h.

10.148.3.16 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.148.3.17 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.148.3.18 int coco::ihessBackwardEvaluator::update (const bool & *__rval*) [inline]

Definition at line 1691 of file ihess_evaluator.h.

10.148.3.19 int coco::ihessBackwardEvaluator::update (const expression_node & *__data*, const bool & *__rval*) [inline]

Definition at line 1701 of file ihess_evaluator.h.

10.148.3.20 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 892 of file search_graph.cc.

10.148.3.21 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.148.3.22 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.148.3.23 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

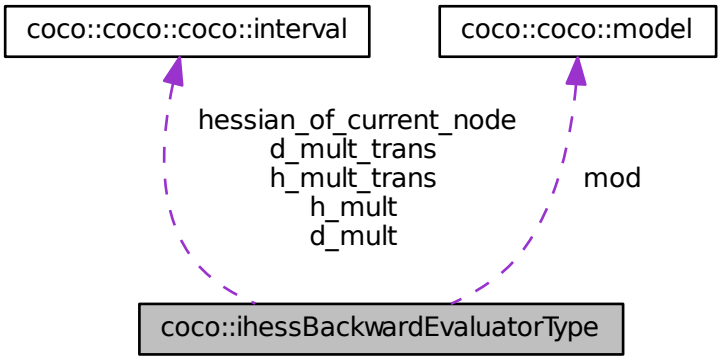
The documentation for this class was generated from the following file:

- ihess_evaluator.h

10.149 coco::ihessBackwardEvaluatorType Struct Reference

`#include <ihess_evaluator.h>`

Collaboration diagram for coco::ihessBackwardEvaluatorType:



Public Attributes

- std::vector< std::vector < interval > > * d_data
- std::vector< std::vector < interval > > * h_data
- std::vector< std::vector < interval > > * d_cache
- std::vector< std::vector < interval > > * h_cache
- std::vector< interval > * grad_vec
- std::vector< interval > * hess_vec
- bool calc_grad
- const model * mod
- interval d_mult
- interval hessian_of_current_node
- bool hess_zero
- interval d_mult_trans
- interval h_mult
- interval h_mult_trans
- bool is_linear
- unsigned int child_n

10.149.1 Member Data Documentation

10.149.1.1 bool coco::ihessBackwardEvaluatorType::calc_grad

Definition at line 1408 of file ihess_evaluator.h.

10.149.1.2 unsigned int coco::ihessBackwardEvaluatorType::child_n

Definition at line 1417 of file ihess_evaluator.h.

10.149.1.3 std::vector<std::vector<interval> >* coco::ihessBackwardEvaluatorType::d_cache

Definition at line 1404 of file ihess_evaluator.h.

10.149.1.4 std::vector<std::vector<interval> >* coco::ihessBackwardEvaluatorType::d_data

Definition at line 1402 of file ihess_evaluator.h.

10.149.1.5 interval coco::ihessBackwardEvaluatorType::d_mult

Definition at line 1410 of file ihess_evaluator.h.

10.149.1.6 interval coco::ihessBackwardEvaluatorType::d_mult_trans

Definition at line 1413 of file ihess_evaluator.h.

10.149.1.7 std::vector<interval>* coco::ihessBackwardEvaluatorType::grad_vec

Definition at line 1406 of file ihess_evaluator.h.

10.149.1.8 std::vector<std::vector<interval> >* coco::ihessBackwardEvaluatorType::h_cache

Definition at line 1405 of file ihess_evaluator.h.

10.149.1.9 std::vector<std::vector<interval> >* coco::ihessBackwardEvaluatorType::h_data

Definition at line 1403 of file ihess_evaluator.h.

10.149.1.10 interval coco::ihessBackwardEvaluatorType::h_mult

Definition at line 1414 of file ihess_evaluator.h.

10.149.1.11 interval coco::ihessBackwardEvaluatorType::h_mult_trans

Definition at line 1415 of file ihess_evaluator.h.

10.149.1.12 std::vector<interval>* coco::ihessBackwardEvaluatorType::hess_vec

Definition at line 1407 of file ihess_evaluator.h.

10.149.1.13 bool coco::ihessBackwardEvaluatorType::hess_zero

Definition at line 1412 of file ihess_evaluator.h.

10.149.1.14 interval coco::ihessBackwardEvaluatorType::hessian_of_current_node

Definition at line 1411 of file ihess_evaluator.h.

10.149.1.15 bool coco::ihessBackwardEvaluatorType::is_linear

Definition at line 1416 of file ihess_evaluator.h.

10.149.1.16 const model* coco::ihessBackwardEvaluatorType::mod

Definition at line 1409 of file ihess_evaluator.h.

The documentation for this struct was generated from the following file:

- ihess_evaluator.h

10.150 coco::ihessForwardEvaluator Class Reference

`#include <ihess_evaluator.h>`

Inheritance diagram for coco::ihessForwardEvaluator:

Collaboration diagram for coco::ihessForwardEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- ihessForwardEvaluator (const std::vector< diffI > &__x, const std::vector< interval > &__rg, const variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > &__d, std::vector< std::vector< interval > > &__h, std::vector< diffI > *__c, bool inters=true, const std::vector< interval > *_center=NULL)
- ihessForwardEvaluator (const ihessForwardEvaluator &__x)
- ~ihessForwardEvaluator ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void newPoint (const std::vector< diffI > &__x, const variable_indicator &__v)
- void newRange (const std::vector< interval > &__rg, const variable_indicator &__v)
- void newCenter (const std::vector< interval > *_center)
- void changeIntersectBehaviour (bool intersect)
- void initialize ()

- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const diffI &__rval)
- int update (const expression_node &__data, const ihessForwardEvaluatorReturnValue &__rval)
- ihessForwardEvaluatorReturnValue calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.150.1 Member Typedef Documentation

10.150.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.150.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.150.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.150.2 Constructor & Destructor Documentation

10.150.2.1 coco::ihessForwardEvaluator::ihessForwardEvaluator (const std::vector< diffI > & __x, const std::vector< interval > & __rg, const variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > & __d, std::vector< std::vector< interval > > & __h, std::vector< diffI > * __c, bool *inters* = true, const std::vector< interval > * *_center* = NULL) [inline]

Definition at line 254 of file ihess_evaluator.h.

10.150.2.2 coco::ihessForwardEvaluator::ihessForwardEvaluator (const ihessForwardEvaluator & __x) [inline]

Definition at line 298 of file ihess_evaluator.h.

10.150.2.3 coco::ihessForwardEvaluator::~ihessForwardEvaluator ( ) [inline]

Definition at line 302 of file ihess_evaluator.h.

10.150.3 Member Function Documentation

10.150.3.1 void coco::ihessForwardEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 437 of file ihess_evaluator.h.

10.150.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.150.3.3 ihessForwardEvaluatorReturnValue coco::ihessForwardEvaluator::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 1391 of file ihess_evaluator.h.

10.150.3.4 void coco::ihessForwardEvaluator::changeIntersectBehaviour (bool *intersect*) [inline]

Definition at line 346 of file ihess_evaluator.h.

10.150.3.5 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.150.3.6 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.150.3.7 void coco::ihessForwardEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 351 of file ihess_evaluator.h.

10.150.3.8 int coco::ihessForwardEvaluator::initialize (const expression_node & __data) [inline]

Definition at line 360 of file ihess_evaluator.h.

10.150.3.9 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.150.3.10 bool coco::ihessForwardEvaluator::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 181 of file ihess_evaluator.h.

10.150.3.11 void coco::ihessForwardEvaluator::newCenter (const std::vector< interval > * _center) [inline]

This method changes the center for centered forms to _center.

Definition at line 341 of file ihess_evaluator.h.

10.150.3.12 void coco::ihessForwardEvaluator::newPoint (const std::vector< diffI > & __x, const variable_indicator & __v) [inline]

Definition at line 307 of file ihess_evaluator.h.

10.150.3.13 void coco::ihessForwardEvaluator::newRange (const std::vector< interval > & __rg, const variable_indicator & __v) [inline]

This method changes the node ranges to ___rg. The parameter * ___v specifies which variables have changed value since the last * evaluation.

Definition at line 334 of file ihess_evaluator.h.

10.150.3.14 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.150.3.15 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.150.3.16 void coco::ihessForwardEvaluator::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 448 of file ihess_evaluator.h.

10.150.3.17 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.150.3.18 expression_const_walker coco::ihessForwardEvaluator::short_cut_to (const expression_node & __data) [inline]

Definition at line 304 of file ihess_evaluator.h.

10.150.3.19 int coco::ihessForwardEvaluator::update (const diffI & __rval) [inline]

Definition at line 455 of file ihess_evaluator.h.

10.150.3.20 int coco::ihessForwardEvaluator::update (const expression_node & *__data*, const ihessForwardEvaluatorReturnValue & *__rval*) [inline]

Definition at line 465 of file ihess_evaluator.h.

10.150.3.21 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.150.3.22 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.150.3.23 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.150.3.24 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.150.3.25 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.150.3.26 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

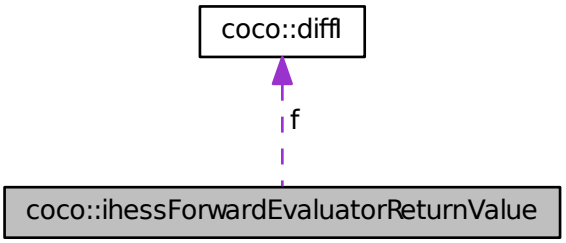
The documentation for this class was generated from the following file:

- ihess_evaluator.h

10.151 coco::ihessForwardEvaluatorReturnValue Struct Reference

`#include <ihess_evaluator.h>`

Collaboration diagram for coco::ihessForwardEvaluatorReturnValue:



Public Attributes

- diffI f
- unsigned int nn
- tristate in_chn

10.151.1 Member Data Documentation

10.151.1.1 diffI coco::ihessForwardEvaluatorReturnValue::f

Definition at line 147 of file ihess_evaluator.h.

10.151.1.2 tristate coco::ihessForwardEvaluatorReturnValue::in_chn

Definition at line 150 of file ihess_evaluator.h.

10.151.1.3 unsigned int coco::ihessForwardEvaluatorReturnValue::nn

Definition at line 149 of file ihess_evaluator.h.

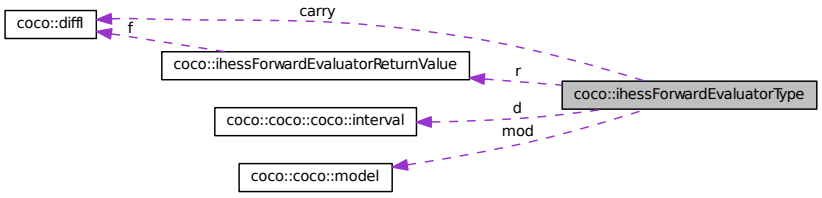
The documentation for this struct was generated from the following file:

- ihess_evaluator.h

10.152 coco::ihessForwardEvaluatorType Struct Reference

`#include <ihess_evaluator.h>`

Collaboration diagram for coco::ihessForwardEvaluatorType:



Public Attributes

- const std::vector< diffI > * x
- const std::vector< interval > * range
- std::vector< diffI > * f_cache
- std::vector< std::vector < interval > > * h_data
- std::vector< std::vector < interval > > * d_data
- std::vector< unsigned int > * t
- std::vector< bool > * b
- bool do_intersect
- ihessForwardEvaluatorReturnValue r
- diffI carry
- const model * mod
- void * p
- interval d
- unsigned int n
- unsigned int info
- const std::vector< interval > * c

10.152.1 Member Data Documentation

10.152.1.1 std::vector<bool>* coco::ihessForwardEvaluatorType::b

Definition at line 161 of file ihess_evaluator.h.

10.152.1.2 const std::vector<interval>* coco::ihessForwardEvaluatorType::c

Definition at line 169 of file ihess_evaluator.h.

10.152.1.3 diffI coco::ihessForwardEvaluatorType::carry

Definition at line 164 of file ihess_evaluator.h.

10.152.1.4 interval coco::ihessForwardEvaluatorType::d

Definition at line 167 of file ihess_evaluator.h.

10.152.1.5 std::vector<std::vector<interval> >* coco::ihessForwardEvaluatorType::d_data

Definition at line 159 of file ihess_evaluator.h.

10.152.1.6 bool coco::ihessForwardEvaluatorType::do_intersect

Definition at line 162 of file ihess_evaluator.h.

10.152.1.7 std::vector<diffI>* coco::ihessForwardEvaluatorType::f_cache

Definition at line 157 of file ihess_evaluator.h.

10.152.1.8 std::vector<std::vector<interval> >* coco::ihessForwardEvaluatorType::h_data

Definition at line 158 of file ihess_evaluator.h.

10.152.1.9 unsigned int coco::ihessForwardEvaluatorType::info

Definition at line 168 of file ihess_evaluator.h.

10.152.1.10 const model* coco::ihessForwardEvaluatorType::mod

Definition at line 165 of file ihess_evaluator.h.

10.152.1.11 unsigned int coco::ihessForwardEvaluatorType::n

Definition at line 168 of file ihess_evaluator.h.

10.152.1.12 void* coco::ihessForwardEvaluatorType::p

Definition at line 166 of file ihess_evaluator.h.

10.152.1.13 ihessForwardEvaluatorReturnValue coco::ihessForwardEvaluatorType::r

Definition at line 163 of file ihess_evaluator.h.

10.152.1.14 const std::vector<interval>* coco::ihessForwardEvaluatorType::range

the ranges of all nodes

Definition at line 156 of file ihess_evaluator.h.

10.152.1.15 std::vector<unsigned int>* coco::ihessForwardEvaluatorType::t

Definition at line 160 of file ihess_evaluator.h.

10.152.1.16 const std::vector<diffI>* coco::ihessForwardEvaluatorType::x

Definition at line 155 of file ihess_evaluator.h.

The documentation for this struct was generated from the following file:

- ihess_evaluator.h

10.153 coco::ihessNumber Class Reference

`#include <ihessNumber.h>`

Public Member Functions

- ihessNumber (void)
- ihessNumber (const interval &a)
- ihessNumber (const interval &a, const interval &u, const interval &v)
- ihessNumber (const interval &a, const interval &u, const interval &v, const interval &h)
- ihessNumber (const ihessNumber &a)
- virtual ~ihessNumber ()
- ihessNumber & operator= (const ihessNumber &a)
- ihessNumber & operator+= (const ihessNumber &a)
- ihessNumber & operator-= (const ihessNumber &a)
- ihessNumber & operator*= (const ihessNumber &a)
- ihessNumber & operator/= (const ihessNumber &a)
- ihessNumber & operator+= (const interval &a)
- ihessNumber & operator-= (const interval &a)
- ihessNumber & operator*= (const interval &a)
- ihessNumber & operator/= (const interval &a)
- interval & operator[ ] (unsigned int pos)
- const interval & operator[ ] (unsigned int pos) const
- interval operator() (unsigned int pos) const
- const interval & setVal (unsigned int n, const interval &v)

Static Public Member Functions

- static ihessNumber getVar (const interval &x)

10.153.1 Constructor & Destructor Documentation

10.153.1.1 coco::ihessNumber::ihessNumber (void) [inline]

Definition at line 101 of file ihessNumber.h.

10.153.1.2 coco::ihessNumber::ihessNumber (const interval & *a*) [inline]

Definition at line 108 of file ihessNumber.h.

10.153.1.3 coco::ihessNumber::ihessNumber (const interval & *a,* const interval & *u,* const interval & *v*) [inline]

Definition at line 115 of file ihessNumber.h.

10.153.1.4 coco::ihessNumber::ihessNumber (const interval & *a,* const interval & *u,* const interval & *v,* const interval & *h*) [inline]

Definition at line 123 of file ihessNumber.h.

10.153.1.5 coco::ihessNumber::ihessNumber (const ihessNumber & *a*) [inline]

Definition at line 132 of file ihessNumber.h.

10.153.1.6 coco::ihessNumber::∼ihessNumber ( ) [inline, virtual]

Definition at line 376 of file ihessNumber.h.

10.153.2 Member Function Documentation

10.153.2.1 ihessNumber coco::ihessNumber::getVar (const interval & *x*) [inline, static]

Definition at line 204 of file ihessNumber.h.

10.153.2.2 interval coco::ihessNumber::operator() (unsigned int *pos*) const

10.153.2.3 ihessNumber & coco::ihessNumber::operator∗= (const ihessNumber & *a*)

Definition at line 8 of file ihessNumber.cc.

10.153.2.4 ihessNumber & coco::ihessNumber::operator∗= (const interval & *a*) [inline]

Definition at line 186 of file ihessNumber.h.

10.153.2.5 ihessNumber & coco::ihessNumber::operator+= (const ihessNumber & *a*) [inline]

Definition at line 158 of file ihessNumber.h.

10.153.2.6 ihessNumber & coco::ihessNumber::operator+= (const interval & *a*) [inline]

Definition at line 166 of file ihessNumber.h.

10.153.2.7 ihessNumber & coco::ihessNumber::operator-= (const ihessNumber & *a*) [inline]

Definition at line 172 of file ihessNumber.h.

10.153.2.8 ihessNumber & coco::ihessNumber::operator-= (const interval & *a*) [inline]

Definition at line 180 of file ihessNumber.h.

10.153.2.9 ihessNumber & coco::ihessNumber::operator/= (const ihessNumber & *a*)

Definition at line 17 of file ihessNumber.cc.

10.153.2.10 ihessNumber & coco::ihessNumber::operator/= (const interval & *a*) [inline]

Definition at line 194 of file ihessNumber.h.

10.153.2.11 ihessNumber & coco::ihessNumber::operator= (const ihessNumber & *a*) [inline]

Definition at line 138 of file ihessNumber.h.

10.153.2.12 interval & coco::ihessNumber::operator[ ] (unsigned int *pos*) [inline]

Definition at line 152 of file ihessNumber.h.

10.153.2.13 const interval & coco::ihessNumber::operator[ ] (unsigned int *pos*) const [inline]

Definition at line 147 of file ihessNumber.h.

10.153.2.14 const interval & coco::ihessNumber::setVal (unsigned int *n*, const interval & *v*)

Definition at line 304 of file ihessNumber.cc.

The documentation for this class was generated from the following files:

- ihessNumber.h
- ihessNumber.cc

10.154 coco::ihessPreparationEvaluator Class Reference

`#include <ihess_evaluator.h>`

Inheritance diagram for coco::ihessPreparationEvaluator:

Collaboration diagram for coco::ihessPreparationEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- ihessPreparationEvaluator (std::vector< std::vector< interval > > &__d, std::vector< std::vector< interval > > &__h, unsigned int _num_of_nodes)
- ihessPreparationEvaluator (const ihessPreparationEvaluator &__x)
- ~ihessPreparationEvaluator ()
- void initialize ()
- bool is_cached (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- int update (bool __rval)
- int update (const expression_node &__data, bool __rval)
- bool calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

10.154.1 Member Typedef Documentation

10.154.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.154.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.154.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.154.2 Constructor & Destructor Documentation

10.154.2.1 coco::ihessPreparationEvaluator::ihessPreparationEvaluator (std::vector< std::vector< interval > > & __d, std::vector< std::vector< interval > > & __h, unsigned int _num_of_nodes) [inline]

Definition at line 83 of file ihess_evaluator.h.

10.154.2.2 coco::ihessPreparationEvaluator::ihessPreparationEvaluator (const ihessPreparationEvaluator & __x) [inline]

Definition at line 102 of file ihess_evaluator.h.

10.154.2.3 coco::ihessPreparationEvaluator::~ihessPreparationEvaluator ( ) [inline]

Definition at line 106 of file ihess_evaluator.h.

10.154.3 Member Function Documentation

10.154.3.1 void coco::ihessPreparationEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 127 of file ihess_evaluator.h.

10.154.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.154.3.3 bool coco::ihessPreparationEvaluator::calculate_value (bool eval_all) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 139 of file ihess_evaluator.h.

10.154.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.154.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.154.3.6 void coco::ihessPreparationEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 108 of file ihess_evaluator.h.

10.154.3.7 int coco::ihessPreparationEvaluator::initialize (const expression_node & __data) [inline]

Definition at line 117 of file ihess_evaluator.h.

10.154.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.154.3.9 bool coco::ihessPreparationEvaluator::is_cached (const expression_node & __data) [inline]

Definition at line 110 of file ihess_evaluator.h.

10.154.3.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & __data) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.154.3.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.154.3.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.154.3.13 void coco::ihessPreparationEvaluator::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 115 of file ihess_evaluator.h.

10.154.3.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.154.3.15 int coco::ihessPreparationEvaluator::update (bool __rval) [inline]

Definition at line 129 of file ihess_evaluator.h.

10.154.3.16 int coco::ihessPreparationEvaluator::update (const expression_node & __data, bool __rval) [inline]

Definition at line 131 of file ihess_evaluator.h.

10.154.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.154.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.154.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.154.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.154.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.154.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- ihess_evaluator.h

10.155 coco::ihessPreparationEvaluatorType Struct Reference

`#include <ihess_evaluator.h>`

Public Attributes

- std::vector< std::vector < interval > > * d
- std::vector< std::vector < interval > > * h

10.155.1 Member Data Documentation

10.155.1.1 std::vector<std::vector<interval> >* coco::ihessPreparationEvaluatorType::d

Definition at line 70 of file ihess_evaluator.h.

10.155.1.2 std::vector<std::vector<interval> >* coco::ihessPreparationEvaluatorType::h

Definition at line 71 of file ihess_evaluator.h.

The documentation for this struct was generated from the following file:

- ihess_evaluator.h

10.156 coco::infbound_eval Class Reference

```
#include <infb_evaluator.h>
```

Inheritance diagram for coco::infbound_eval:

Collaboration diagram for coco::infbound_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- infbound_eval (const std::vector< infbound > &__x, const variable_indicator &__v, const model &__m, std::vector< infbound > *__c)
- infbound_eval (const infbound_eval &__v)
- ~infbound_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const infbound &__rval)
- int update (const expression_node &__data, const infbound &__rval)
- infbound calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.156.1 Member Typedef Documentation

10.156.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.156.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.156.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.156.2 Constructor & Destructor Documentation

10.156.2.1 coco::infbound_eval::infbound_eval (const std::vector< infbound > & __x, const variable_indicator & __v, const model & __m, std::vector< infbound > * __c) [inline]

Definition at line 116 of file infb_evaluator.h.

10.156.2.2 coco::infbound_eval::infbound_eval (const infbound_eval & __v) [inline]

Definition at line 128 of file infb_evaluator.h.

10.156.2.3 coco::infbound_eval::~infbound_eval ( ) [inline]

Definition at line 130 of file infb_evaluator.h.

10.156.3 Member Function Documentation

10.156.3.1 void coco::infbound_eval::calculate (const expression_node & __data) [inline]

Definition at line 193 of file infb_evaluator.h.

10.156.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.156.3.3 infbound coco::infbound_eval::calculate_value (bool eval_all) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 709 of file infb_evaluator.h.

10.156.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.156.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.156.3.6 void coco::infbound_eval::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 135 of file infb_evaluator.h.

10.156.3.7 int coco::infbound_eval::initialize (const expression_node & *__data*) [inline]

Definition at line 137 of file infb_evaluator.h.

10.156.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.156.3.9 bool coco::infbound_eval::is_cached (const node_data_type & *__data*) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 70 of file infb_evaluator.h.

10.156.3.10 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.156.3.11 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.156.3.12 void coco::infbound_eval::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 205 of file infb_evaluator.h.

10.156.3.13 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.156.3.14 expression_const_walker coco::infbound_eval::short_cut_to (const expression_node & __data) [inline]

Definition at line 132 of file infb_evaluator.h.

10.156.3.15 int coco::infbound_eval::update (const infbound & __rval) [inline]

Definition at line 221 of file infb_evaluator.h.

10.156.3.16 int coco::infbound_eval::update (const expression_node & __data, const infbound & __rval) [inline]

Definition at line 227 of file infb_evaluator.h.

10.156.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.156.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.156.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.156.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.156.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.156.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

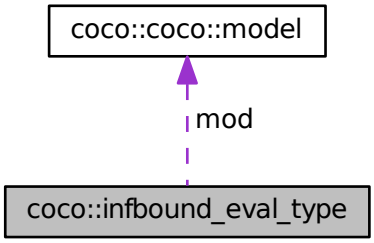
The documentation for this class was generated from the following file:

- infb_evaluator.h

10.157 coco::infbound_eval_type Struct Reference

`#include <infb_evaluator.h>`

Collaboration diagram for coco::infbound_eval_type:



Public Attributes

- const std::vector< infbound > * x
- std::vector< infbound > * cache
- const model * mod
- void * p
- infbound d
- int info
- infbound r
- unsigned int n

10.157.1 Member Data Documentation

10.157.1.1 std::vector<infbound>* coco::infbound_eval_type::cache

Definition at line 52 of file infb_evaluator.h.

10.157.1.2 infbound coco::infbound_eval_type::d

Definition at line 55 of file infb_evaluator.h.

10.157.1.3 int coco::infbound_eval_type::info

Definition at line 56 of file infb_evaluator.h.

10.157.1.4 const model* coco::infbound_eval_type::mod

Definition at line 53 of file infb_evaluator.h.

10.157.1.5 unsigned int coco::infbound_eval_type::n

Definition at line 58 of file infb_evaluator.h.

10.157.1.6 void* coco::infbound_eval_type::p

Definition at line 54 of file infb_evaluator.h.

10.157.1.7 infbound coco::infbound_eval_type::r

Definition at line 57 of file infb_evaluator.h.

10.157.1.8 const std::vector<infbound>* coco::infbound_eval_type::x

Definition at line 51 of file infb_evaluator.h.

The documentation for this struct was generated from the following file:

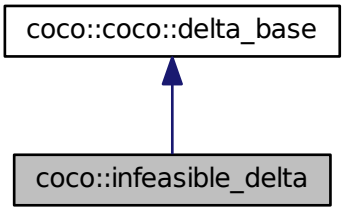
- infb_evaluator.h

10.158 coco::infeasible_delta Class Reference

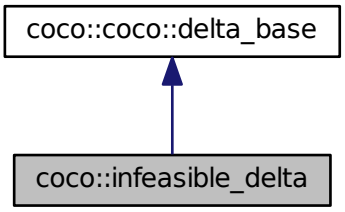
The infeasible delta class for marking a model as infeasible.

`#include <infeasible_delta.h>`

Inheritance diagram for coco::infeasible_delta:



Collaboration diagram for coco::infeasible_delta:



Public Member Functions

- infeasible_delta ()
- infeasible_delta (const infeasible_delta &__d)
- ~infeasible_delta ()
- infeasible_delta * new_copy () const
- void destroy_copy (delta_base *__d) const
- bool apply (work_node &_x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const infeasible_delta &_c) const
- bool operator!= (const infeasible_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- const std::string & get_action () const
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual void unkeep ()
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Protected Attributes

- std::string _action

10.158.1 Detailed Description

This class is used to mark a model as infeasible.

10.158.2 Constructor & Destructor Documentation

10.158.2.1 coco::infeasible_delta::infeasible_delta ( ) [inline]

Standard Constructor

Definition at line 85 of file infeasible_delta.h.

10.158.2.2 coco::infeasible_delta::infeasible_delta (const infeasible_delta & __d) [inline]

Standard Copy Constructor

Definition at line 87 of file infeasible_delta.h.

10.158.2.3 coco::infeasible_delta::~infeasible_delta ( ) [inline]

Standard Destructor

Definition at line 95 of file infeasible_delta.h.

10.158.3 Member Function Documentation

10.158.3.1 bool coco::infeasible_delta::apply (work_node & _x, undelta_base *& _u, const delta_id & _did, size_t & delta_size) const [inline]

Apply the delta with delta_id _d to work node _x, hereby marking the model as infeasible.

Definition at line 104 of file infeasible_delta.h.

10.158.3.2 virtual bool coco::coco::delta_base::apply (work_node & _x, undelta_base *& _u, const delta_id & _di, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.158.3.3 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.158.3.4 bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & *delta_size*) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.158.3.5 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & *delta_size*) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.158.3.6 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.158.3.7 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.158.3.8 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.158.3.9 void coco::infeasible_delta::destroy_copy (delta_base * __d) const [inline]

Clone Destructor

Definition at line 100 of file infeasible_delta.h.

10.158.3.10 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.158.3.11 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.158.3.12 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.158.3.13 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.158.3.14 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.158.3.15 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.158.3.16 infeasible_delta* coco::infeasible_delta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::delta_base.

Definition at line 98 of file infeasible_delta.h.

10.158.3.17 bool coco::infeasible_delta::operator!= (const delta_base & *_c*) const [inline]

Definition at line 116 of file infeasible_delta.h.

10.158.3.18 bool coco::infeasible_delta::operator!= (const infeasible_delta & *_c*) const [inline]

Definition at line 122 of file infeasible_delta.h.

10.158.3.19 bool coco::infeasible_delta::operator== (const delta_base & *_c*) const [inline]

Comparison operators

Definition at line 113 of file infeasible_delta.h.

10.158.3.20 bool coco::infeasible_delta::operator== (const infeasible_delta & _c) const [inline]

Comparison operators

Definition at line 121 of file infeasible_delta.h.

10.158.3.21 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.158.3.22 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.158.3.23 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.158.4 Member Data Documentation

10.158.4.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

The documentation for this class was generated from the following file:

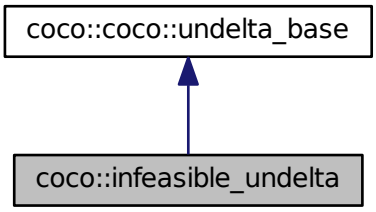
- infeasible_delta.h

10.159 coco::infeasible_undelta Class Reference

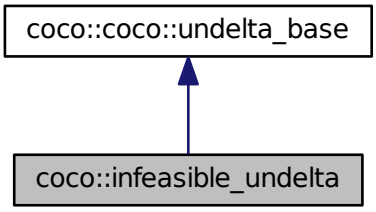
The infeasible undelta class for undoing changes to the feasibility of a model.

`#include <infeasible_delta.h>`

Inheritance diagram for coco::infeasible_undelta:



Collaboration diagram for coco::infeasible_undelta:



Public Member Functions

- infeasible_undelta (bool _oi=false)
- infeasible_undelta (const infeasible_undelta &__d)
- ~infeasible_undelta ()
- infeasible_undelta * new_copy () const
- void destroy_copy (undelta_base *__d) const
- bool unapply (work_node &_x, const delta_id &_did) const
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

10.159.1 Detailed Description

This class is used to specify undo information for changes to the feasibility status of a model. This class undoes the apply of a infeasible_delta.

10.159.2 Constructor & Destructor Documentation

10.159.2.1 coco::infeasible_undelta::infeasible_undelta (bool *_oi* = `false`) [inline]

Constructor initializing old_infeasible with `_oi`

Definition at line 50 of file infeasible_delta.h.

10.159.2.2 coco::infeasible_undelta::infeasible_undelta (const infeasible_undelta & *__d*) [inline]

Standard Copy Constructor

Definition at line 52 of file infeasible_delta.h.

10.159.2.3 coco::infeasible_undelta::~infeasible_undelta ( ) [inline]

Standard Destructor

Definition at line 61 of file infeasible_delta.h.

10.159.3 Member Function Documentation

10.159.3.1 void coco::infeasible_undelta::destroy_copy (undelta_base * *__d*) const [inline]

Clone Destructor

Definition at line 66 of file infeasible_delta.h.

10.159.3.2 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.159.3.3 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.159.3.4 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.159.3.5 infeasible_undelta* coco::infeasible_undelta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::undelta_base.

Definition at line 64 of file infeasible_delta.h.

10.159.3.6 bool coco::infeasible_undelta::unapply (work_node & _x, const delta_id & _did) const [inline, virtual]

Undo the infeasible_delta with delta_id _i in work node _x

Reimplemented from coco::coco::undelta_base.

Definition at line 69 of file infeasible_delta.h.

10.159.3.7 virtual bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.159.3.8 virtual bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.159.3.9 bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [inline, virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

Definition at line 94 of file api_delta.h.

The documentation for this class was generated from the following file:

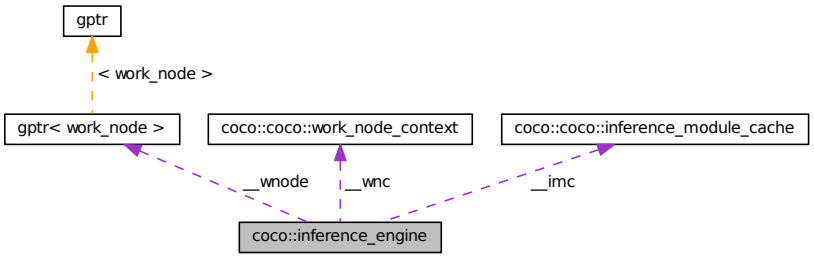
- infeasible_delta.h

10.160 coco::inference_engine Class Reference

Inference engine base class.

`#include <inference_engine.h>`

Collaboration diagram for coco::inference_engine:



Public Member Functions

- inference_engine (const gptr< work_node > &wnode, const std::string &__n, inference_module_cache &cache)
- inference_engine (const gptr< work_node > &wnode, const std::string &__n)
- virtual ~inference_engine ()
- virtual bool update_engine (const gptr< work_node > &wnode)
- std::pair< std::list< delta_id > , std::list< delta_id > > new_deltas ()
- const delta * get_delta (const delta_id &__d) const
- const model * get_model () const
- virtual ie_return_type infer (const control_data &__c)
- const std::string & get_name () const
- virtual statistic_info last_call_stat ()
- virtual statistic_info cumulative_stat ()

Protected Attributes

- std::string __name
- const gptr< work_node > * __wnode
- const work_node_context * __wnc
- const vdbl::viewdbase & __vdb
- inference_module_cache * __imc
- std::vector< delta_id > _old_deltas
- std::vector< delta_id > _new_deltas

Friends

- class inference_module_cache

10.160.1 Detailed Description

This is the base class of all COCONUT inference engine modules. An inference engine is an inference module, which specializes in analyzing a work node. Typical inference engines are, e.g., convexity analyzers, local solvers, incomplete global solvers,...

10.160.2 Constructor & Destructor Documentation

10.160.2.1 coco::inference_engine::inference_engine (const gptr< work_node > & *wnode,* const std::string & *__n,* inference_module_cache & *cache*) [inline]

This is the standard constructor for an inference_engine using the inference module cache. It sets the indentifier string to n, the work node to wnode and the cache to cache.

Definition at line 109 of file inference_engine.h.

10.160.2.2 coco::inference_engine::inference_engine (const gptr< work_node > & *wnode,* const std::string & *__n*) [inline]

This is the standard constructor for an inference_engine. It sets the indentifier string to n and the work node to wnode.

Definition at line 120 of file inference_engine.h.

10.160.2.3 virtual coco::inference_engine::~inference_engine ( ) [inline, virtual]

Standard Destructor

Definition at line 129 of file inference_engine.h.

10.160.3 Member Function Documentation

10.160.3.1 virtual statistic_info coco::inference_engine::cumulative_stat ( ) [inline, virtual]

This method returns the cumulative statistics gathered through all calls of the infer method.

Definition at line 168 of file inference_engine.h.

10.160.3.2 const delta* coco::inference_engine::get_delta (const delta_id & *__d*) const [inline]

The get_delta retrieves the delta with delta_id ___d.

Definition at line 145 of file inference_engine.h.

10.160.3.3 const model* coco::inference_engine::get_model ( ) const [inline]

A call to this method returns a pointer to the model of the work node.

Definition at line 149 of file inference_engine.h.

10.160.3.4 const std::string& coco::inference_engine::get_name ( ) const [inline]

The get_name method returns the identifier string of the inference engine.

Definition at line 161 of file inference_engine.h.

10.160.3.5 virtual ie_return_type coco::inference_engine::infer (const control_data & *__c*) [inline, virtual]

This method is the main method of an inference engine. It is supposed to analyze the work node and to return the information gained in the ie_return_type structure. This method is overloaded by the various

subclasses. Service information and parameters are provided via the control_data structure __c.

Definition at line 157 of file inference_engine.h.

10.160.3.6 virtual statistic_info coco::inference_engine::last_call_stat ( ) [inline, virtual]

This method returns the statistics gathered in the last call of the infer method.

Definition at line 165 of file inference_engine.h.

10.160.3.7 std::pair< std::list< delta_id >, std::list< delta_id > > coco::inference_engine::new_deltas ( )

This method returns the deltas to be applied and unapplied for properly updating the inference engine.

Definition at line 51 of file inference_engine.cc.

10.160.3.8 virtual bool coco::inference_engine::update_engine (const gptr< work_node > & *wnode*) [inline, virtual]

This update_engine method changes the work node to wnode.

Definition at line 132 of file inference_engine.h.

10.160.4 Friends And Related Function Documentation

10.160.4.1 friend class inference_module_cache [friend]

Definition at line 170 of file inference_engine.h.

10.160.5 Member Data Documentation

10.160.5.1 inference_module_cache* coco::inference_engine::__imc [protected]

This is the inference module cache used to store and retrieve data frequently used by various inference modules.

Definition at line 93 of file inference_engine.h.

10.160.5.2 std::string coco::inference_engine::__name [protected]

This is the identifier string for an inference engine.

Definition at line 83 of file inference_engine.h.

10.160.5.3 const vdbl::viewdbase& coco::inference_engine::__vdb [protected]

This is a view to the search database.

Definition at line 90 of file inference_engine.h.

10.160.5.4 const work_node_context* coco::inference_engine::__wnc [protected]

This is the work_node_context for extracting information from the search database.

Definition at line 88 of file inference_engine.h.

10.160.5.5 const gptr<work_node>* coco::inference_engine::__wnode [protected]

This variable contains a global pointer to the work node.

Definition at line 85 of file inference_engine.h.

10.160.5.6 std::vector<delta_id> coco::inference_engine::_new_deltas [protected]

This vector keeps the new deltas for updating an engine with a call to the update_engine method (these deltas need to be applied).

Definition at line 99 of file inference_engine.h.

10.160.5.7 std::vector<delta_id> coco::inference_engine::_old_deltas [protected]

This vector keeps the old deltas for updating an engine with a call to the update_engine method (these deltas need to be unapplied).

Definition at line 96 of file inference_engine.h.

The documentation for this class was generated from the following files:

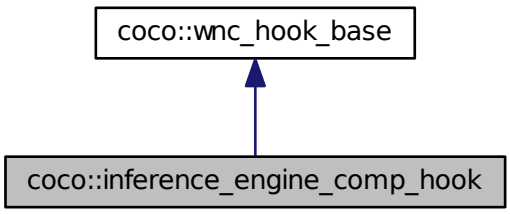
- inference_engine.h
- inference_engine.cc

10.161 coco::inference_engine_comp_hook Class Reference

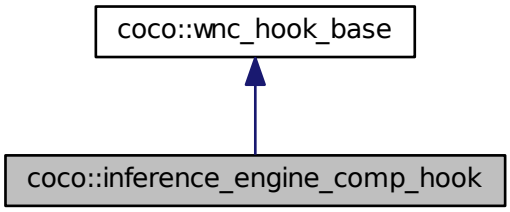
The inference engine meta computation hook (work node computation hook)

`#include <infeng_hook.h>`

Inheritance diagram for coco::inference_engine_comp_hook:



Collaboration diagram for coco::inference_engine_comp_hook:



Public Member Functions

- inference_engine_comp_hook (inference_engine *_ie, const control_data &_cd, double _thr=-COCO_INF)
- inference_engine_comp_hook (inference_engine *_ie, const control_data &_cd, const std::map< std::triple< std::string, bool, int >, std::pair< std::string, basic_alltype > > &_itc, double _thr=-COCO_INF)
- inference_engine_comp_hook (const inference_engine_comp_hook &iec)
- virtual ~inference_engine_comp_hook ()
- inference_engine_comp_hook * new_copy () const
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *add_ds=NULL, std::list< certificate > *add_cs=NULL) const
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)
- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.161.1 Detailed Description

This class is a work node computation hook (

See also

work_node_comp_hook), which calls an inference engine to compute additional information like deltas, etc. for the work node. The information is stored in configurable columns.

10.161.2 Constructor & Destructor Documentation

10.161.2.1 coco::inference_engine_comp_hook::inference_engine_comp_hook (inference_engine * *_ie*, const control_data & *_cd*, double *_thr* = -COCO_INF) [inline]

Standard Constructor

Definition at line 66 of file infeng_hook.h.

10.161.2.2 coco::inference_engine_comp_hook::inference_engine_comp_hook (inference_engine * *_ie*, const control_data & *_cd*, const std::map< std::triple< std::string, bool, int >, std::pair< std::string, basic_alltype > > & *_itc*, double *_thr* = -COCO_INF) [inline]

Enhanced Standard Constructor

Definition at line 73 of file infeng_hook.h.

10.161.2.3 coco::inference_engine_comp_hook::inference_engine_comp_hook (const inference_engine_comp_hook & *iec*) [inline]

Copy Constructor

Definition at line 82 of file infeng_hook.h.

10.161.2.4 virtual coco::inference_engine_comp_hook::~inference_engine_comp_hook ( ) [inline, virtual]

Standard Destructor

Definition at line 88 of file infeng_hook.h.

10.161.3 Member Function Documentation

10.161.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected, inherited]

Remove all columns, whose identifiers are stored in __managed_cols from table stable.

Definition at line 34 of file comp_hook.cc.

10.161.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in __managed_cols.

Definition at line 238 of file comp_hook.h.

10.161.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in __managed_cols.

Definition at line 135 of file comp_hook.h.

10.161.3.4 bool coco::inference_engine_comp_hook::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon unregistering this hook, destroy the registered columns

Reimplemented from coco::wnc_hook_base.

Definition at line 415 of file infeng_hook.h.

10.161.3.5 bool coco::inference_engine_comp_hook::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the columns defined.

Reimplemented from coco::wnc_hook_base.

Definition at line 245 of file infeng_hook.h.

10.161.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline, inherited]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.161.3.7 inference_engine_comp_hook* coco::inference_engine_comp_hook::new_copy ( ) const [inline, virtual]

Clone Operation

Implements coco::wnc_hook_base.

Definition at line 91 of file infeng_hook.h.

10.161.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_id of work_node `wn`.

Definition at line 43 of file comp_hook.cc.

10.161.3.9 void coco::inference_engine_comp_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *add_ds* = NULL, std::list< certificate > * *add_cs* = NULL) const [inline, virtual]

The evaluation operator of this computation hook. It stores the lower and upper bounds on the objective function on the work_node `wn` in dbt_row `dbr`.

Implements coco::wnc_hook_base.

Definition at line 97 of file infeng_hook.h.

10.161.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_relation of work_node `wn`.

Definition at line 46 of file comp_hook.cc.

The documentation for this class was generated from the following file:

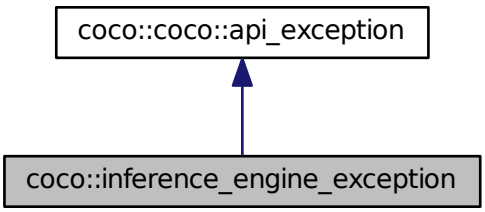
- infeng_hook.h

10.162 coco::inference_engine_exception Class Reference

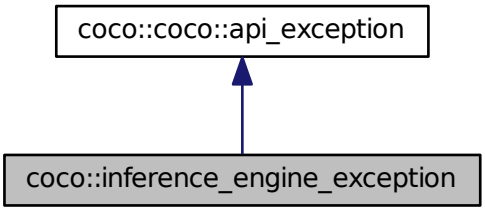
Inference engine exception class.

`#include <inference_engine.h>`

Inheritance diagram for coco::inference_engine_exception:



Collaboration diagram for coco::inference_engine_exception:



Public Member Functions

- inference_engine_exception (const std::string &msg)

- inference_engine_exception (const char *msg)
- virtual ~inference_engine_exception () throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()

10.162.1 Detailed Description

This is the exceptions class thrown by inference_engine subclasses. Every properly coded COCONUT inference engine module should only throw exceptions of this type.

10.162.2 Constructor & Destructor Documentation

10.162.2.1 coco::inference_engine_exception::inference_engine_exception (const std::string & *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 60 of file inference_engine.h.

10.162.2.2 coco::inference_engine_exception::inference_engine_exception (const char * *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 63 of file inference_engine.h.

10.162.2.3 virtual coco::inference_engine_exception::~inference_engine_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 67 of file inference_engine.h.

10.162.3 Member Function Documentation

10.162.3.1 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file expression.h.

10.162.3.2 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.162.3.3 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.162.3.4 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.162.3.5 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.162.3.6 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.162.3.7 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.162.3.8 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file expression.h.

10.162.3.9 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.162.3.10 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.162.3.11 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.162.3.12 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.162.3.13 const char * coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

Definition at line 57 of file api_exception.cc.

10.162.3.14 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.162.3.15 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.162.3.16 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.162.3.17 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.162.3.18 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file expression.h.

10.162.3.19 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.162.3.20 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following file:

- inference_engine.h

10.163 coco::coco::inference_module_cache Class Reference

Inference module cache class.

Public Member Functions

- inference_module_cache ()
- inference_module_cache (const inference_module_cache &ic, bool all=true)
- ~inference_module_cache ()
- void set_from (const inference_module_cache &im, tristate inval)
- void invalidate (bool for_subbox=false)
- void invalidate_for_subbox ()
- template<class _TS >
  tristate get (unsigned int i, unsigned int cl, std::vector< _TS > *&ret, const gptr< work_node > &wn, bool optimal=false)
- template<class _TS >
  tristate get (unsigned int i, unsigned int cl, vmtl::sparse_vector< _TS > *&ret, const gptr< work_node > &wn, bool optimal=false)
- template<class _TS >
  tristate get (unsigned int i, unsigned int cl, vmtl::dense_matrix< _TS > *&ret, const gptr< work_node > &wn, bool optimal=false)
- template<class _TS >
  tristate get (unsigned int i, unsigned int cl, vmtl::sparse_matrix< _TS > *&ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, bool &ret, const gptr< work_node > &wn, bool optimal=false)

- tristate get (unsigned int i, unsigned int cl, int &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, unsigned int &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, double &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, interval &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, void *&ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, std::string &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, basic_alltype *&ret, const gptr< work_node > &wn, bool optimal=false)
- void set (unsigned int i, unsigned int cl, const basic_alltype &ret, bool thisbox)
- bool register_autogen_method (unsigned int i, unsigned int cl, inference_module_cache_autogen *mth)
- bool deregister_autogen_method (unsigned int i, unsigned int cl)
- search_node_id get_top_box_id () const

10.163.1 Detailed Description

This is the class keeping the cache for all COCONUT inference modules. This cache can be directly written to and read from in inference modules. Typical data stored in the cache are 1d function info, point evaluations at the center of a box, range enclosures over a box,...

10.163.2 Constructor & Destructor Documentation

10.163.2.1 coco::coco::inference_module_cache::inference_module_cache ( ) [inline]

This is the standard constructor for an inference_module_cache.

Definition at line 123 of file search_graph.cc.

10.163.2.2 coco::coco::inference_module_cache::inference_module_cache (const inference_module_cache & *ic,* bool *all* = true) [inline]

This is the standard constructor for an inference_module_cache.

Definition at line 133 of file search_graph.cc.

10.163.2.3 coco::coco::inference_module_cache::~inference_module_cache ( ) [inline]

Standard Destructor

Definition at line 141 of file search_graph.cc.

10.163.3 Member Function Documentation

10.163.3.1 bool coco::inference_module_cache::deregister_autogen_method (unsigned int *i,* unsigned int *cl*)

This method is used to deregister a new auto-generate method for class and entry `of` the inference module cache.

Definition at line 104 of file inf_module_cache.cc.

10.163.3.2 template<class _TS > tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* std::vector< _TS > *& *ret,* const gptr< work_node > & *wn,* bool *optimal* = `false`) [inline]

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

Definition at line 118 of file inf_module_cache.hpp.

10.163.3.3 template<class _TS > tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* vmtl::sparse_vector< _TS > *& *ret,* const gptr< work_node > & *wn,* bool *optimal* = `false`) [inline]

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

Definition at line 131 of file inf_module_cache.hpp.

10.163.3.4 template<class _TS > tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* vmtl::dense_matrix< _TS > *& *ret,* const gptr< work_node > & *wn,* bool *optimal* = `false`) [inline]

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

Definition at line 144 of file inf_module_cache.hpp.

10.163.3.5 template<class _TS > tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* vmtl::sparse_matrix< _TS > *& *ret,* const gptr< work_node > & *wn,* bool *optimal* = `false`) [inline]

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

Definition at line 157 of file inf_module_cache.hpp.

10.163.3.6 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* bool & *ret,* const gptr< work_node > & *wn,* bool *optimal* = `false`) [inline]

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

Definition at line 169 of file inf_module_cache.hpp.

10.163.3.7 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* int & *ret,* const gptr< work_node > & *wn,* bool *optimal* = false) [inline]

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

Definition at line 179 of file inf_module_cache.hpp.

10.163.3.8 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* unsigned int & *ret,* const gptr< work_node > & *wn,* bool *optimal* = false) [inline]

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

Definition at line 189 of file inf_module_cache.hpp.

10.163.3.9 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* double & *ret,* const gptr< work_node > & *wn,* bool *optimal* = false) [inline]

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

Definition at line 199 of file inf_module_cache.hpp.

10.163.3.10 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* interval & *ret,* const gptr< work_node > & *wn,* bool *optimal* = false) [inline]

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

Definition at line 209 of file inf_module_cache.hpp.

10.163.3.11 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* void *& *ret,* const gptr< work_node > & *wn,* bool *optimal* = false) [inline]

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

Definition at line 219 of file inf_module_cache.hpp.

10.163.3.12 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* std::string & *ret,* const gptr< work_node > & *wn,* bool *optimal* = false) [inline]

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

Definition at line 229 of file inf_module_cache.hpp.

10.163.3.13 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* basic_alltype *& *ret,* const gptr< work_node > & *wn,* bool *optimal* = false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

Definition at line 36 of file inf_module_cache.cc.

10.163.3.14 search_node_id coco::inference_module_cache::get_top_box_id ( ) const [inline]

This is the getter method for `top_box_id`.

Definition at line 239 of file inf_module_cache.hpp.

10.163.3.15 void coco::coco::inference_module_cache::invalidate (bool *for_subbox* = `false`) [inline]

This method invalidates the non-persistent part of the cache if `for_subbox` is `false`. Otherwise it only invalidates the thisbox part of the cache.

This method invalidates the non-persistent part of the cache.

Definition at line 53 of file inf_module_cache.hpp.

10.163.3.16 void coco::coco::inference_module_cache::invalidate_for_subbox ( ) [inline]

This method invalidates all parts of the cache which are neither persistent nor subbox.

Definition at line 71 of file inf_module_cache.hpp.

10.163.3.17 bool coco::inference_module_cache::register_autogen_method (unsigned int *i,* unsigned int *cl,* inference_module_cache_autogen * *mth*)

This method is used to register a new auto-generate method for class and entry `of` the inference module cache. Note that the methods MUST be generated by a call to new!

Definition at line 89 of file inf_module_cache.cc.

10.163.3.18 void coco::coco::inference_module_cache::set (unsigned int *i,* unsigned int *cl,* const basic_alltype & *val,* bool *thisbox*)

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

Definition at line 75 of file inf_module_cache.cc.

10.163.3.19 void coco::coco::inference_module_cache::set_from (const inference_module_cache & *im,* tristate *inval*) [inline]

This method sets the cache from `im`. If `inval` is `t_true`, the copied cache is invalidated, if it is `t_maybe` only the thisbox part is invalidated.

This method sets the persistent part of the cache and copies the registered methods.

Definition at line 35 of file inf_module_cache.hpp.

The documentation for this class was generated from the following files:

- inf_module_cache.h
- inf_module_cache.cc
- inf_module_cache.hpp

10.164 coco::inference_module_cache Class Reference

Inference module cache class.

`#include <inf_module_cache.h>`

Public Member Functions

- inference_module_cache ()
- inference_module_cache (const inference_module_cache &ic, bool all=true)
- ~inference_module_cache ()
- void set_from (const inference_module_cache &im, tristate inval)
- void invalidate (bool for_subbox=false)
- void invalidate_for_subbox ()
- template<class _TS >
  tristate get (unsigned int i, unsigned int cl, std::vector< _TS > *&ret, const gptr< work_node > &wn, bool optimal=false)
- template<class _TS >
  tristate get (unsigned int i, unsigned int cl, vmtl::sparse_vector< _TS > *&ret, const gptr< work_node > &wn, bool optimal=false)
- template<class _TS >
  tristate get (unsigned int i, unsigned int cl, vmtl::dense_matrix< _TS > *&ret, const gptr< work_node > &wn, bool optimal=false)
- template<class _TS >
  tristate get (unsigned int i, unsigned int cl, vmtl::sparse_matrix< _TS > *&ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, bool &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, int &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, unsigned int &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, double &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, interval &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, void *&ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, std::string &ret, const gptr< work_node > &wn, bool optimal=false)
- tristate get (unsigned int i, unsigned int cl, basic_alltype *&ret, const gptr< work_node > &wn, bool optimal=false)
- void set (unsigned int i, unsigned int cl, const basic_alltype &ret, bool thisbox)
- bool register_autogen_method (unsigned int i, unsigned int cl, inference_module_cache_autogen *mth)
- bool deregister_autogen_method (unsigned int i, unsigned int cl)
- search_node_id get_top_box_id () const

10.164.1 Detailed Description

This is the class keeping the cache for all COCONUT inference modules. This cache can be directly written to and read from in inference modules. Typical data stored in the cache are 1d function info, point evaluations at the center of a box, range enclosures over a box,...

10.164.2 Constructor & Destructor Documentation

10.164.2.1 coco::inference_module_cache::inference_module_cache ( ) [inline]

This is the standard constructor for an inference_module_cache.

Definition at line 123 of file inf_module_cache.h.

10.164.2.2 coco::inference_module_cache::inference_module_cache (const inference_module_cache & *ic*, bool *all* = true) [inline]

This is the standard constructor for an inference_module_cache.

Definition at line 133 of file inf_module_cache.h.

10.164.2.3 coco::inference_module_cache::~inference_module_cache ( ) [inline]

Standard Destructor

Definition at line 141 of file inf_module_cache.h.

10.164.3 Member Function Documentation

10.164.3.1 bool coco::inference_module_cache::deregister_autogen_method (unsigned int *i*, unsigned int *cl*)

This method is used to deregister a new auto-generate method for class and entry of the inference module cache.

10.164.3.2 template<class _TS > tristate coco::inference_module_cache::get (unsigned int *i*, unsigned int *cl*, std::vector< _TS > *& *ret*, const gptr< work_node > & *wn*, bool *optimal* = false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.3 template<class _TS > tristate coco::inference_module_cache::get (unsigned int *i*, unsigned int *cl*, vmtl::sparse_vector< _TS > *& *ret*, const gptr< work_node > & *wn*, bool *optimal* = false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.4 template<class _TS > tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* vmtl::dense_matrix< _TS > *&* *ret,* const gptr< work_node > & *wn,* bool *optimal =* false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.5 template<class _TS > tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* vmtl::sparse_matrix< _TS > *&* *ret,* const gptr< work_node > & *wn,* bool *optimal =* false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.6 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* bool & *ret,* const gptr< work_node > & *wn,* bool *optimal =* false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.7 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* int & *ret,* const gptr< work_node > & *wn,* bool *optimal =* false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.8 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* unsigned int & *ret,* const gptr< work_node > & *wn,* bool *optimal =* false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.9 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* double & *ret,* const gptr< work_node > & *wn,* bool *optimal =* false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.10 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* interval & *ret,* const gptr< work_node > & *wn,* bool *optimal =* false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.11 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* void *&* *ret,* const gptr< work_node > & *wn,* bool *optimal =* false)

This method retrieves the information in entry i of the persistent cache in ret. It returns whether it was successful.

10.164.3.12 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* std::string & *ret,* const gptr< work_node > & *wn,* bool *optimal* = `false`)

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

10.164.3.13 tristate coco::inference_module_cache::get (unsigned int *i,* unsigned int *cl,* basic_alltype *& *ret,* const gptr< work_node > & *wn,* bool *optimal* = `false`)

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

10.164.3.14 search_node_id coco::inference_module_cache::get_top_box_id ( ) const

This is the getter method for `top_box_id`.

10.164.3.15 void coco::inference_module_cache::invalidate (bool *for_subbox* = `false`)

This method invalidates the non-persistent part of the cache if `for_subbox` is `false`. Otherwise it only invalidates the thisbox part of the cache.

10.164.3.16 void coco::inference_module_cache::invalidate_for_subbox ( )

This method invalidates all parts of the cache which are neither persistent nor subbox.

10.164.3.17 bool coco::inference_module_cache::register_autogen_method (unsigned int *i,* unsigned int *cl,* inference_module_cache_autogen * *mth*)

This method is used to register a new auto-generate method for class and entry `of` the inference module cache. Note that the methods MUST be generated by a call to new!

10.164.3.18 void coco::inference_module_cache::set (unsigned int *i,* unsigned int *cl,* const basic_alltype & *ret,* bool *thisbox*)

This method retrieves the information in entry `i` of the persistent cache in ret. It returns whether it was successful.

10.164.3.19 void coco::inference_module_cache::set_from (const inference_module_cache & *im,* tristate *inval*)

This method sets the cache from `im`. If `inval` is `t_true`, the copied cache is invalidated, if it is `t_maybe` only the thisbox part is invalidated.

The documentation for this class was generated from the following file:

- inf_module_cache.h

10.165 coco::inference_module_cache_autogen Class Reference

Inference module cache auto-generate method base class.

`#include <inf_module_cache.h>`

Public Member Functions

- virtual bool operator() (inference_module_cache &imc, unsigned int i, unsigned int cl, const gptr< work_node > &wn)=0
- virtual ~inference_module_cache_autogen ()

10.165.1 Detailed Description

This is the base class for auto-generate methods of the inference module cache. All methods must be derived from this base class. They can then be registered in the cache.

10.165.2 Constructor & Destructor Documentation

10.165.2.1 virtual coco::inference_module_cache_autogen::~inference_module_cache_autogen ( ) [inline, virtual]

Definition at line 62 of file inf_module_cache.h.

10.165.3 Member Function Documentation

10.165.3.1 virtual bool coco::inference_module_cache_autogen::operator() (inference_module_cache & *imc,* unsigned int *i,* unsigned int *cl,* const gptr< work_node > & *wn*) [pure virtual]

This module will be called to automatically to regenerate information for the inference module cache.

The documentation for this class was generated from the following file:

- inf_module_cache.h

10.166 coco::coco::inference_module_cache_autogen Class Reference

Inference module cache auto-generate method base class.

Public Member Functions

- virtual bool operator() (inference_module_cache &imc, unsigned int i, unsigned int cl, const gptr< work_node > &wn)=0
- virtual ~inference_module_cache_autogen ()

10.166.1 Detailed Description

This is the base class for auto-generate methods of the inference module cache. All methods must be derived from this base class. They can then be registered in the cache.

10.166.2 Constructor & Destructor Documentation

10.166.2.1 virtual coco::coco::inference_module_cache_autogen::~inference_module_cache_autogen ( ) [inline, virtual]

Definition at line 62 of file search_graph.cc.

10.166.3 Member Function Documentation

10.166.3.1 virtual bool coco::coco::inference_module_cache_autogen::operator() (inference_module_cache & *imc,* unsigned int *i,* unsigned int *cl,* const gptr< work_node > & *wn*) [pure virtual]

This module will be called to automatically to regenerate information for the inference module cache.

The documentation for this class was generated from the following file:

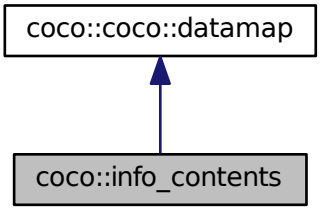
- inf_module_cache.h

10.167 coco::info_contents Class Reference

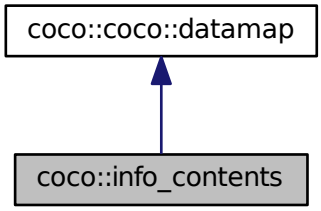
The class for returning additional information from inference modules.

`#include <info_contents.h>`

Inheritance diagram for coco::info_contents:



Collaboration diagram for coco::info_contents:



Public Member Functions

- info_contents ()
- info_contents (const std::string &__n, const basic_alltype &__v)
- info_contents (const char *__n, const basic_alltype &__v)
- info_contents (const info_contents &__c)
- virtual ~info_contents ()
- info_contents & operator= (const info_contents &__c)
- void list (std::vector< std::string > &__v) const

- bool sinsert (const std::string &__s, const basic_alltype &__h, bool replace)
- bool sinsert (const char *__s, const basic_alltype &__h, bool replace)

- bool sinsert (const std::string &__s, int i, const basic_alltype &__h, bool replace)
- bool sinsert (const char *__s, int i, const basic_alltype &__h, bool replace)

- const basic_alltype & sfind (const std::string &__s) const
- const basic_alltype & sfind (const char *__s) const

- const basic_alltype & sfind (const std::string &__s, int i) const
- const basic_alltype & sfind (const char *__s, int i) const

- void remove (const std::string &__s)
- void remove (const char *__s)

- void remove (const std::string &__s, int i)
- void remove (const char *__s, int i)

- bool defd (const std::string &__s) const
- bool defd (const char *__s) const

- bool defd (const std::string &__s, int i) const
- bool defd (const char *__s, int i) const

- bool which (const std::string &__s, std::vector< int > &__idx) const
- bool which (const char *__s, std::vector< int > &__idx) const

- bool retrieve (const std::string &__s, bool &__v) const
- bool retrieve (const std::string &__s, int &__v) const
- bool retrieve (const std::string &__s, unsigned int &__v) const
- bool retrieve (const std::string &__s, double &__v) const
- bool retrieve (const std::string &__s, interval &__v) const
- bool retrieve (const std::string &__s, std::string &__is) const
- bool retrieve (const std::string &__s, num::Number &__v) const
- bool retrieve (const std::string &__s, const std::vector< bool > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< int > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< unsigned int > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< double > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< interval > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< std::string > *&__v) const
- bool retrieve (const std::string &__s, const std::vector< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< interval > *&__v) const

- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< num::Number > *&__v) const
- bool retrieve (const char *__s, bool &__v) const
- bool retrieve (const char *__s, int &__v) const
- bool retrieve (const char *__s, unsigned int &__v) const
- bool retrieve (const char *__s, double &__v) const
- bool retrieve (const char *__s, interval &__v) const
- bool retrieve (const char *__s, std::string &__is) const
- bool retrieve (const char *__s, num::Number &__is) const
- bool retrieve (const char *__s, const std::vector< bool > *&__v) const
- bool retrieve (const char *__s, const std::vector< unsigned int > *&__v) const
- bool retrieve (const char *__s, const std::vector< int > *&__v) const
- bool retrieve (const char *__s, const std::vector< double > *&__v) const
- bool retrieve (const char *__s, const std::vector< interval > *&__v) const
- bool retrieve (const char *__s, const std::vector< std::string > *&__v) const
- bool retrieve (const char *__s, const std::vector< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< num::Number > *&__v) const

- bool retrieve (const std::string &__s, bool &__v, bool __def) const
- bool retrieve (const std::string &__s, int &__v, int __def) const
- bool retrieve (const std::string &__s, unsigned int &__v, unsigned int __def) const
- bool retrieve (const std::string &__s, double &__v, double __def) const
- bool retrieve (const std::string &__s, interval &__v, const interval &__def) const
- bool retrieve (const std::string &__s, std::string &__is, const std::string &__def) const
- bool retrieve (const std::string &__s, num::Number &__is, const num::Number &__def) const
- bool retrieve (const std::string &__s, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve (const std::string &__s, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve (const std::string &__s, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve (const std::string &__s, const std::vector< double > *&__v, const std::vector< double > *__def) const

- bool retrieve (const std::string &__s, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve (const std::string &__s, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve (const std::string &__s, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve (const std::string &__s, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const
- bool retrieve (const char *__s, bool &__v, bool __def) const
- bool retrieve (const char *__s, int &__v, int __def) const
- bool retrieve (const char *__s, unsigned int &__v, unsigned int __def) const
- bool retrieve (const char *__s, double &__v, double __def) const
- bool retrieve (const char *__s, interval &__v, const interval &__def) const
- bool retrieve (const char *__s, std::string &__v, const std::string &__def) const
- bool retrieve (const char *__s, num::Number &__v, const num::Number &__def) const
- bool retrieve (const char *__s, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve (const char *__s, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const

- bool retrieve (const char *__s, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve (const char *__s, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve (const char *__s, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve (const char *__s, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve (const char *__s, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve (const char *__s, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const

- bool retrieve_i (const std::string &__s, int i, bool &__v) const
- bool retrieve_i (const std::string &__s, int i, int &__v) const
- bool retrieve_i (const std::string &__s, int i, unsigned int &__v) const
- bool retrieve_i (const std::string &__s, int i, double &__v) const
- bool retrieve_i (const std::string &__s, int i, interval &__v) const

- bool retrieve_i (const std::string &__s, int i, std::string &__v) const
- bool retrieve_i (const std::string &__s, int i, num::Number &__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< bool > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< unsigned int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, bool &__v) const
- bool retrieve_i (const char *__s, int i, int &__v) const
- bool retrieve_i (const char *__s, int i, unsigned int &__v) const
- bool retrieve_i (const char *__s, int i, double &__v) const
- bool retrieve_i (const char *__s, int i, interval &__v) const
- bool retrieve_i (const char *__s, int i, std::string &__is) const
- bool retrieve_i (const char *__s, int i, num::Number &__is) const
- bool retrieve_i (const char *__s, int i, const std::vector< bool > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< unsigned int > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const std::vector< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< bool > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< unsigned int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< std::string > *&__v) const

- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< num::Number > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< std::string > *&__v) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< num::Number > *&__v) const

- bool retrieve_i (const std::string &__s, int i, bool &__v, bool __def) const
- bool retrieve_i (const std::string &__s, int i, int &__v, int __def) const
- bool retrieve_i (const std::string &__s, int i, unsigned int &__v, unsigned int __def) const
- bool retrieve_i (const std::string &__s, int i, double &__v, double __def) const
- bool retrieve_i (const std::string &__s, int i, interval &__v, const interval &__def) const
- bool retrieve_i (const std::string &__s, int i, std::string &__is, const std::string &__def) const
- bool retrieve_i (const std::string &__s, int i, num::Number &__is, const num::Number &__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const

- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve_i (const std::string &__s, int i, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, bool &__v, bool __def) const
- bool retrieve_i (const char *__s, int i, int &__v, int __def) const
- bool retrieve_i (const char *__s, int i, unsigned int &__v, unsigned int __def) const
- bool retrieve_i (const char *__s, int i, double &__v, double __def) const
- bool retrieve_i (const char *__s, int i, interval &__v, const interval &__def) const
- bool retrieve_i (const char *__s, int i, std::string &__v, const std::string &__def) const
- bool retrieve_i (const char *__s, int i, num::Number &__v, const num::Number &__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< bool > *&__v, const std::vector< bool > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< int > *&__v, const std::vector< int > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< unsigned int > *&__v, const std::vector< unsigned int > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< double > *&__v, const std::vector< double > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< interval > *&__v, const std::vector< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< std::string > *&__v, const std::vector< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const std::vector< num::Number > *&__v, const std::vector< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< bool > *&__v, const vmtl::sparse_vector< bool > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< int > *&__v, const vmtl::sparse_vector< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< unsigned int > *&__v, const vmtl::sparse_vector< unsigned int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< double > *&__v, const vmtl::sparse_vector< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< interval > *&__v, const vmtl::sparse_vector< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< std::string > *&__v, const vmtl::sparse_vector< std::string > *__def) const

- bool retrieve_i (const char *__s, int i, const vmtl::sparse_vector< num::Number > *&__v, const vmtl::sparse_vector< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< double > *&__v, const vmtl::dense_matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< int > *&__v, const vmtl::dense_matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< interval > *&__v, const vmtl::dense_matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< std::string > *&__v, const vmtl::dense_matrix< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::dense_matrix< num::Number > *&__v, const vmtl::dense_matrix< num::Number > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< double > *&__v, const vmtl::sparse_matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< int > *&__v, const vmtl::sparse_matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< interval > *&__v, const vmtl::sparse_matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< std::string > *&__v, const vmtl::sparse_matrix< std::string > *__def) const
- bool retrieve_i (const char *__s, int i, const vmtl::sparse_matrix< num::Number > *&__v, const vmtl::sparse_matrix< num::Number > *__def) const

10.167.1 Detailed Description

This class is used for returning additional information from inference modules. It is basically a datamap.

10.167.2 Constructor & Destructor Documentation

10.167.2.1 coco::info_contents::info_contents ( ) [inline]

Standard Constructor

Definition at line 50 of file info_contents.h.

10.167.2.2 coco::info_contents::info_contents (const std::string & *__n*, const basic_alltype & *__v*) [inline]

Constructor setting the variable __n to value __v.

Definition at line 52 of file info_contents.h.

10.167.2.3 coco::info_contents::info_contents (const char * *__n*, const basic_alltype & *__v*) [inline]

Constructor setting the variable __n to value __v.

Definition at line 55 of file info_contents.h.

10.167.2.4 coco::info_contents::info_contents (const info_contents & __c) [inline]

Standard Copy Constructor

Definition at line 58 of file info_contents.h.

10.167.2.5 virtual coco::info_contents::~info_contents ( ) [inline, virtual]

Standard Destructor

Definition at line 60 of file info_contents.h.

10.167.3 Member Function Documentation

10.167.3.1 bool coco::datamap::defd (const std::string & __s) const [inline, inherited]

This method returns whether the variable with name __s is defined.

Definition at line 109 of file datamap.hpp.

10.167.3.2 bool coco::datamap::defd (const char * __s) const [inline, inherited]

This method returns whether the variable with name __s is defined.

Definition at line 114 of file datamap.hpp.

10.167.3.3 bool coco::datamap::defd (const std::string & __s, int i) const [inline, inherited]

This method returns whether the variable with name __s and index is defined.

Definition at line 119 of file datamap.hpp.

10.167.3.4 bool coco::datamap::defd (const char * __s, int i) const [inline, inherited]

This method returns whether the variable with name __s and index is defined.

Definition at line 124 of file datamap.hpp.

10.167.3.5 void coco::datamap::list (std::vector< std::string > & __v) const [inherited]

The list method sets __v to the list of entry keys stored in this datamap.

Reimplemented in coco::control_data.

Definition at line 101 of file datamap.cc.

10.167.3.6 info_contents& coco::info_contents::operator= (const info_contents & __c) [inline]

Standard Assignment Operator

Definition at line 63 of file info_contents.h.

10.167.3.7 void coco::datamap::remove (const std::string & __s) [inline, inherited]

The remove methods remove the variable with name ___s from the datamap.

Definition at line 57 of file datamap.hpp.

10.167.3.8 void coco::datamap::remove (const char * __s) [inline, inherited]

The remove methods remove the variable with name ___s from the datamap.

Definition at line 65 of file datamap.hpp.

10.167.3.9 void coco::datamap::remove (const std::string & __s, int i) [inline, inherited]

The remove methods remove the variable with name ___s and index i from the datamap.

Definition at line 70 of file datamap.hpp.

10.167.3.10 void coco::datamap::remove (const char * __s, int i) [inline, inherited]

The remove methods remove the variable with name ___s and index i from the datamap.

Definition at line 76 of file datamap.hpp.

10.167.3.11 bool coco::datamap::retrieve (const std::string & __s, bool & __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 149 of file datamap.hpp.

10.167.3.12 bool coco::datamap::retrieve (const std::string & __s, int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 169 of file datamap.hpp.

10.167.3.13 bool coco::datamap::retrieve (const std::string & __s, unsigned int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 191 of file datamap.hpp.

10.167.3.14 bool coco::datamap::retrieve (const std::string & __s, double & __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to

the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 213 of file datamap.hpp.

10.167.3.15 bool coco::datamap::retrieve (const std::string & __s, interval & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 235 of file datamap.hpp.

10.167.3.16 bool coco::datamap::retrieve (const std::string & __s, std::string & __is) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 257 of file datamap.hpp.

10.167.3.17 bool coco::datamap::retrieve (const std::string & __s, num::Number & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 279 of file datamap.hpp.

10.167.3.18 bool coco::datamap::retrieve (const std::string & __s, const std::vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 301 of file datamap.hpp.

10.167.3.19 bool coco::datamap::retrieve (const std::string & __s, const std::vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 323 of file datamap.hpp.

10.167.3.20 bool coco::datamap::retrieve (const std::string & __s, const std::vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 345 of file datamap.hpp.

10.167.3.21 bool coco::datamap::retrieve (const std::string & __s, const std::vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 367 of file datamap.hpp.

10.167.3.22 bool coco::datamap::retrieve (const std::string & __s, const std::vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 389 of file datamap.hpp.

10.167.3.23 bool coco::datamap::retrieve (const std::string & __s, const std::vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 411 of file datamap.hpp.

10.167.3.24 bool coco::datamap::retrieve (const std::string & __s, const std::vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 433 of file datamap.hpp.

10.167.3.25 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 455 of file datamap.hpp.

10.167.3.26 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 477 of file datamap.hpp.

10.167.3.27 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 499 of file datamap.hpp.

10.167.3.28 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 521 of file datamap.hpp.

10.167.3.29 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 543 of file datamap.hpp.

10.167.3.30 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 565 of file datamap.hpp.

10.167.3.31 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 587 of file datamap.hpp.

10.167.3.32 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 609 of file datamap.hpp.

10.167.3.33 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 631 of file datamap.hpp.

10.167.3.34 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 653 of file datamap.hpp.

10.167.3.35 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 675 of file datamap.hpp.

10.167.3.36 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 697 of file datamap.hpp.

10.167.3.37 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 719 of file datamap.hpp.

10.167.3.38 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 741 of file datamap.hpp.

10.167.3.39 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 763 of file datamap.hpp.

10.167.3.40 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 785 of file datamap.hpp.

10.167.3.41 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 807 of file datamap.hpp.

10.167.3.42 bool coco::datamap::retrieve (const char * __s, bool & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 829 of file datamap.hpp.

10.167.3.43 bool coco::datamap::retrieve (const char * __s, int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 839 of file datamap.hpp.

10.167.3.44 bool coco::datamap::retrieve (const char * __s, unsigned int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 849 of file datamap.hpp.

10.167.3.45 bool coco::datamap::retrieve (const char * __s, double & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 859 of file datamap.hpp.

10.167.3.46 bool coco::datamap::retrieve (const char * __s, interval & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 869 of file datamap.hpp.

10.167.3.47 bool coco::datamap::retrieve (const char * __s, std::string & __is) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 879 of file datamap.hpp.

10.167.3.48 bool coco::datamap::retrieve (const char * __s, num::Number & __is) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 889 of file datamap.hpp.

10.167.3.49 bool coco::datamap::retrieve (const char * __s, const std::vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 899 of file datamap.hpp.

10.167.3.50 bool coco::datamap::retrieve (const char * __s, const std::vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 919 of file datamap.hpp.

10.167.3.51 bool coco::datamap::retrieve (const char * __s, const std::vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 909 of file datamap.hpp.

10.167.3.52 bool coco::datamap::retrieve (const char * __s, const std::vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 929 of file datamap.hpp.

10.167.3.53 bool coco::datamap::retrieve (const char * __s, const std::vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 939 of file datamap.hpp.

10.167.3.54 bool coco::datamap::retrieve (const char * __s, const std::vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 949 of file datamap.hpp.

10.167.3.55 bool coco::datamap::retrieve (const char * __s, const std::vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 959 of file datamap.hpp.

10.167.3.56 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 969 of file datamap.hpp.

10.167.3.57 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 989 of file datamap.hpp.

10.167.3.58 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 979 of file datamap.hpp.

10.167.3.59 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 999 of file datamap.hpp.

10.167.3.60 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1009 of file datamap.hpp.

10.167.3.61 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1019 of file datamap.hpp.

10.167.3.62 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1029 of file datamap.hpp.

10.167.3.63 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1039 of file datamap.hpp.

10.167.3.64 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1050 of file datamap.hpp.

10.167.3.65 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1061 of file datamap.hpp.

10.167.3.66 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1072 of file datamap.hpp.

10.167.3.67 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1083 of file datamap.hpp.

10.167.3.68 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1094 of file datamap.hpp.

10.167.3.69 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1105 of file datamap.hpp.

10.167.3.70 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1116 of file datamap.hpp.

10.167.3.71 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1127 of file datamap.hpp.

10.167.3.72 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1138 of file datamap.hpp.

10.167.3.73 bool coco::datamap::retrieve (const std::string & __s, bool & __v, bool __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 159 of file datamap.hpp.

10.167.3.74 bool coco::datamap::retrieve (const std::string & __s, int & __v, int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 181 of file datamap.hpp.

10.167.3.75 bool coco::datamap::retrieve (const std::string & __s, unsigned int & __v, unsigned int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 203 of file datamap.hpp.

10.167.3.76 bool coco::datamap::retrieve (const std::string & __s, double & __v, double __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 225 of file datamap.hpp.

10.167.3.77 bool coco::datamap::retrieve (const std::string & __s, interval & __v, const interval & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 247 of file datamap.hpp.

10.167.3.78 bool coco::datamap::retrieve (const std::string & __s, std::string & __is, const std::string & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 269 of file datamap.hpp.

10.167.3.79 bool coco::datamap::retrieve (const std::string & __s, num::Number & __is, const num::Number & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 291 of file datamap.hpp.

10.167.3.80 bool coco::datamap::retrieve (const std::string & __s, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 313 of file datamap.hpp.

10.167.3.81 bool coco::datamap::retrieve (const std::string & __s, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 335 of file datamap.hpp.

10.167.3.82 bool coco::datamap::retrieve (const std::string & __s, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 357 of file datamap.hpp.

10.167.3.83 bool coco::datamap::retrieve (const std::string & __s, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 379 of file datamap.hpp.

10.167.3.84 bool coco::datamap::retrieve (const std::string & __s, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 401 of file datamap.hpp.

10.167.3.85 bool coco::datamap::retrieve (const std::string & __s, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 423 of file datamap.hpp.

10.167.3.86 bool coco::datamap::retrieve (const std::string & __s, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 445 of file datamap.hpp.

10.167.3.87 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 467 of file datamap.hpp.

10.167.3.88 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 489 of file datamap.hpp.

10.167.3.89 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 511 of file datamap.hpp.

10.167.3.90 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 533 of file datamap.hpp.

10.167.3.91 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< interval > *& __v*, const vmtl::sparse_vector< interval > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 555 of file datamap.hpp.

10.167.3.92 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< std::string > *& __v*, const vmtl::sparse_vector< std::string > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 577 of file datamap.hpp.

10.167.3.93 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::sparse_vector< num::Number > *& __v*, const vmtl::sparse_vector< num::Number > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 599 of file datamap.hpp.

10.167.3.94 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::dense_matrix< double > *& __v*, const vmtl::dense_matrix< double > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 621 of file datamap.hpp.

10.167.3.95 bool coco::datamap::retrieve (const std::string & *__s*, const vmtl::dense_matrix< int > *& __v*, const vmtl::dense_matrix< int > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 643 of file datamap.hpp.

10.167.3.96 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 665 of file datamap.hpp.

10.167.3.97 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 687 of file datamap.hpp.

10.167.3.98 bool coco::datamap::retrieve (const std::string & __s, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 709 of file datamap.hpp.

10.167.3.99 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 731 of file datamap.hpp.

10.167.3.100 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 753 of file datamap.hpp.

10.167.3.101 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 775 of file datamap.hpp.

10.167.3.102 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 797 of file datamap.hpp.

10.167.3.103 bool coco::datamap::retrieve (const std::string & __s, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 819 of file datamap.hpp.

10.167.3.104 bool coco::datamap::retrieve (const char * __s, bool & __v, bool __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 834 of file datamap.hpp.

10.167.3.105 bool coco::datamap::retrieve (const char * __s, int & __v, int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 844 of file datamap.hpp.

10.167.3.106 bool coco::datamap::retrieve (const char * __s, unsigned int & __v, unsigned int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 854 of file datamap.hpp.

10.167.3.107 bool coco::datamap::retrieve (const char * __s, double & __v, double __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 864 of file datamap.hpp.

10.167.3.108 bool coco::datamap::retrieve (const char * __s, interval & __v, const interval & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 874 of file datamap.hpp.

10.167.3.109 bool coco::datamap::retrieve (const char * __s, std::string & __v, const std::string & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 884 of file datamap.hpp.

10.167.3.110 bool coco::datamap::retrieve (const char * __s, num::Number & __v, const num::Number & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 894 of file datamap.hpp.

10.167.3.111 bool coco::datamap::retrieve (const char * __s, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple

types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 904 of file datamap.hpp.

10.167.3.112 bool coco::datamap::retrieve (const char * __s, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 924 of file datamap.hpp.

10.167.3.113 bool coco::datamap::retrieve (const char * __s, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 914 of file datamap.hpp.

10.167.3.114 bool coco::datamap::retrieve (const char * __s, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 934 of file datamap.hpp.

10.167.3.115 bool coco::datamap::retrieve (const char * __s, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 944 of file datamap.hpp.

10.167.3.116 bool coco::datamap::retrieve (const char * __s, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 954 of file datamap.hpp.

10.167.3.117 bool coco::datamap::retrieve (const char * __s, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 964 of file datamap.hpp.

10.167.3.118 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 974 of file datamap.hpp.

10.167.3.119 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 994 of file datamap.hpp.

10.167.3.120 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 984 of file datamap.hpp.

10.167.3.121 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1004 of file datamap.hpp.

10.167.3.122 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1014 of file datamap.hpp.

10.167.3.123 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1024 of file datamap.hpp.

10.167.3.124 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1034 of file datamap.hpp.

10.167.3.125 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1044 of file datamap.hpp.

10.167.3.126 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1055 of file datamap.hpp.

10.167.3.127 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1066 of file datamap.hpp.

10.167.3.128 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1077 of file datamap.hpp.

10.167.3.129 bool coco::datamap::retrieve (const char * __s, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1088 of file datamap.hpp.

10.167.3.130 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1099 of file datamap.hpp.

10.167.3.131 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1110 of file datamap.hpp.

10.167.3.132 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1121 of file datamap.hpp.

10.167.3.133 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1132 of file datamap.hpp.

10.167.3.134 bool coco::datamap::retrieve (const char * __s, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s to the variable __v. It is returned whether the variable is set. If the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1143 of file datamap.hpp.

10.167.3.135 bool coco::datamap::retrieve_i (const std::string & __s, int i, bool & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1150 of file datamap.hpp.

10.167.3.136 bool coco::datamap::retrieve_i (const std::string & __s, int i, int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1170 of file datamap.hpp.

10.167.3.137 bool coco::datamap::retrieve_i (const std::string & __s, int i, unsigned int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1190 of file datamap.hpp.

10.167.3.138 bool coco::datamap::retrieve_i (const std::string & __s, int i, double & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1210 of file datamap.hpp.

10.167.3.139 bool coco::datamap::retrieve_i (const std::string & __s, int i, interval & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1230 of file datamap.hpp.

10.167.3.140 bool coco::datamap::retrieve_i (const std::string & __s, int i, std::string & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1250 of file datamap.hpp.

10.167.3.141 bool coco::datamap::retrieve_i (const std::string & __s, int i, num::Number & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1270 of file datamap.hpp.

10.167.3.142 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1290 of file datamap.hpp.

10.167.3.143 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1310 of file datamap.hpp.

10.167.3.144 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1330 of file datamap.hpp.

10.167.3.145 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1350 of file datamap.hpp.

10.167.3.146 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1370 of file datamap.hpp.

10.167.3.147 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1390 of file datamap.hpp.

10.167.3.148 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1410 of file datamap.hpp.

10.167.3.149 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1430 of file datamap.hpp.

10.167.3.150 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1450 of file datamap.hpp.

10.167.3.151 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1470 of file datamap.hpp.

10.167.3.152 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1490 of file datamap.hpp.

10.167.3.153 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1510 of file datamap.hpp.

10.167.3.154 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1530 of file datamap.hpp.

10.167.3.155 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1550 of file datamap.hpp.

10.167.3.156 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1570 of file datamap.hpp.

10.167.3.157 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1590 of file datamap.hpp.

10.167.3.158 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1610 of file datamap.hpp.

10.167.3.159 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1630 of file datamap.hpp.

10.167.3.160 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1650 of file datamap.hpp.

10.167.3.161 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1670 of file datamap.hpp.

10.167.3.162 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1690 of file datamap.hpp.

10.167.3.163 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1710 of file datamap.hpp.

10.167.3.164 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1730 of file datamap.hpp.

10.167.3.165 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1750 of file datamap.hpp.

10.167.3.166 bool coco::datamap::retrieve_i (const char * __s, int i, bool & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1770 of file datamap.hpp.

10.167.3.167 bool coco::datamap::retrieve_i (const char * __s, int i, int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1780 of file datamap.hpp.

10.167.3.168 bool coco::datamap::retrieve_i (const char * __s, int i, unsigned int & __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1790 of file datamap.hpp.

10.167.3.169 bool coco::datamap::retrieve_i (const char * __s, int i, double & __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1800 of file datamap.hpp.

10.167.3.170 bool coco::datamap::retrieve_i (const char * __s, int i, interval & __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1810 of file datamap.hpp.

10.167.3.171 bool coco::datamap::retrieve_i (const char * __s, int i, std::string & __is) const [inline, inherited]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1820 of file datamap.hpp.

10.167.3.172 bool coco::datamap::retrieve_i (const char * __s, int i, num::Number & __is) const [inline, inherited]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1830 of file datamap.hpp.

10.167.3.173 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable ___s with index i to the variable ___v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1840 of file datamap.hpp.

10.167.3.174 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1850 of file datamap.hpp.

10.167.3.175 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1860 of file datamap.hpp.

10.167.3.176 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1870 of file datamap.hpp.

10.167.3.177 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1880 of file datamap.hpp.

10.167.3.178 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1890 of file datamap.hpp.

10.167.3.179 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1900 of file datamap.hpp.

10.167.3.180 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< bool > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1910 of file datamap.hpp.

10.167.3.181 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1920 of file datamap.hpp.

10.167.3.182 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< unsigned int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1930 of file datamap.hpp.

10.167.3.183 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1940 of file datamap.hpp.

10.167.3.184 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1950 of file datamap.hpp.

10.167.3.185 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1960 of file datamap.hpp.

10.167.3.186 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1970 of file datamap.hpp.

10.167.3.187 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1980 of file datamap.hpp.

10.167.3.188 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1991 of file datamap.hpp.

10.167.3.189 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2002 of file datamap.hpp.

10.167.3.190 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2013 of file datamap.hpp.

10.167.3.191 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2024 of file datamap.hpp.

10.167.3.192 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< double > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2035 of file datamap.hpp.

10.167.3.193 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< int > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2046 of file datamap.hpp.

10.167.3.194 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< interval > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2057 of file datamap.hpp.

10.167.3.195 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< std::string > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2068 of file datamap.hpp.

10.167.3.196 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< num::Number > *& __v) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to the variable __v. It is returned whether the variable is set. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2079 of file datamap.hpp.

10.167.3.197 bool coco::datamap::retrieve_i (const std::string & __s, int i, bool & __v, bool __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1161 of file datamap.hpp.

10.167.3.198 bool coco::datamap::retrieve_i (const std::string & __s, int i, int & __v, int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1181 of file datamap.hpp.

10.167.3.199 bool coco::datamap::retrieve_i (const std::string & __s, int i, unsigned int & __v, unsigned int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1201 of file datamap.hpp.

10.167.3.200 bool coco::datamap::retrieve_i (const std::string & __s, int i, double & __v, double __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1221 of file datamap.hpp.

10.167.3.201 bool coco::datamap::retrieve_i (const std::string & __s, int i, interval & __v, const interval & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1241 of file datamap.hpp.

10.167.3.202 bool coco::datamap::retrieve_i (const std::string & __s, int i, std::string & __is, const std::string & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1261 of file datamap.hpp.

10.167.3.203 bool coco::datamap::retrieve_i (const std::string & __s, int i, num::Number & __is, const num::Number & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1281 of file datamap.hpp.

10.167.3.204 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1301 of file datamap.hpp.

10.167.3.205 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1321 of file datamap.hpp.

10.167.3.206 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1341 of file datamap.hpp.

10.167.3.207 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1361 of file datamap.hpp.

10.167.3.208 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1381 of file datamap.hpp.

10.167.3.209 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1401 of file datamap.hpp.

10.167.3.210 bool coco::datamap::retrieve_i (const std::string & __s, int i, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1421 of file datamap.hpp.

10.167.3.211 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1441 of file datamap.hpp.

10.167.3.212 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1461 of file datamap.hpp.

10.167.3.213 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1481 of file datamap.hpp.

10.167.3.214 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1501 of file datamap.hpp.

10.167.3.215 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1521 of file datamap.hpp.

10.167.3.216 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1541 of file datamap.hpp.

10.167.3.217 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1561 of file datamap.hpp.

10.167.3.218 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1581 of file datamap.hpp.

10.167.3.219 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1601 of file datamap.hpp.

10.167.3.220 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1621 of file datamap.hpp.

10.167.3.221 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1641 of file datamap.hpp.

10.167.3.222 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1661 of file datamap.hpp.

10.167.3.223 bool coco::datamap::retrieve_i (const std::string & __s, int i, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1681 of file datamap.hpp.

10.167.3.224 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_matrix< int > *&* *__v*, const vmtl::sparse_matrix< int > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1701 of file datamap.hpp.

10.167.3.225 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_matrix< interval > *&* *__v*, const vmtl::sparse_matrix< interval > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1721 of file datamap.hpp.

10.167.3.226 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_matrix< std::string > *&* *__v*, const vmtl::sparse_matrix< std::string > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1741 of file datamap.hpp.

10.167.3.227 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const vmtl::sparse_matrix< num::Number > *&* *__v*, const vmtl::sparse_matrix< num::Number > * *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1761 of file datamap.hpp.

10.167.3.228 bool coco::datamap::retrieve_i (const char * *__s*, int *i*, bool & *__v*, bool *__def*) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1775 of file datamap.hpp.

10.167.3.229 bool coco::datamap::retrieve_i (const char * __s, int i, int & __v, int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1785 of file datamap.hpp.

10.167.3.230 bool coco::datamap::retrieve_i (const char * __s, int i, unsigned int & __v, unsigned int __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1795 of file datamap.hpp.

10.167.3.231 bool coco::datamap::retrieve_i (const char * __s, int i, double & __v, double __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1805 of file datamap.hpp.

10.167.3.232 bool coco::datamap::retrieve_i (const char * __s, int i, interval & __v, const interval & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1815 of file datamap.hpp.

10.167.3.233 bool coco::datamap::retrieve_i (const char * __s, int i, std::string & __v, const std::string & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1825 of file datamap.hpp.

10.167.3.234 bool coco::datamap::retrieve_i (const char * __s, int i, num::Number & __v, const num::Number & __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the

simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1835 of file datamap.hpp.

10.167.3.235 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< bool > *& __v, const std::vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1845 of file datamap.hpp.

10.167.3.236 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< int > *& __v, const std::vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1855 of file datamap.hpp.

10.167.3.237 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< unsigned int > *& __v, const std::vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1865 of file datamap.hpp.

10.167.3.238 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< double > *& __v, const std::vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1875 of file datamap.hpp.

10.167.3.239 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< interval > *& __v, const std::vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1885 of file datamap.hpp.

10.167.3.240 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< std::string > *& __v, const std::vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1895 of file datamap.hpp.

10.167.3.241 bool coco::datamap::retrieve_i (const char * __s, int i, const std::vector< num::Number > *& __v, const std::vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1905 of file datamap.hpp.

10.167.3.242 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< bool > *& __v, const vmtl::sparse_vector< bool > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1915 of file datamap.hpp.

10.167.3.243 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< int > *& __v, const vmtl::sparse_vector< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1925 of file datamap.hpp.

10.167.3.244 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< unsigned int > *& __v, const vmtl::sparse_vector< unsigned int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1935 of file datamap.hpp.

10.167.3.245 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< double > *& __v, const vmtl::sparse_vector< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1945 of file datamap.hpp.

10.167.3.246 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< interval > *& __v, const vmtl::sparse_vector< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1955 of file datamap.hpp.

10.167.3.247 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< std::string > *& __v, const vmtl::sparse_vector< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1965 of file datamap.hpp.

10.167.3.248 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_vector< num::Number > *& __v, const vmtl::sparse_vector< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1975 of file datamap.hpp.

10.167.3.249 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< double > *& __v, const vmtl::dense_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1985 of file datamap.hpp.

10.167.3.250 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< int > *& __v, const vmtl::dense_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 1996 of file datamap.hpp.

10.167.3.251 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< interval > *& __v, const vmtl::dense_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2007 of file datamap.hpp.

10.167.3.252 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< std::string > *& __v, const vmtl::dense_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2018 of file datamap.hpp.

10.167.3.253 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::dense_matrix< num::Number > *& __v, const vmtl::dense_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2029 of file datamap.hpp.

10.167.3.254 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< double > *& __v, const vmtl::sparse_matrix< double > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2040 of file datamap.hpp.

10.167.3.255 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< int > *& __v, const vmtl::sparse_matrix< int > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2051 of file datamap.hpp.

10.167.3.256 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< interval > *& __v, const vmtl::sparse_matrix< interval > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2062 of file datamap.hpp.

10.167.3.257 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< std::string > *& __v, const vmtl::sparse_matrix< std::string > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2073 of file datamap.hpp.

10.167.3.258 bool coco::datamap::retrieve_i (const char * __s, int i, const vmtl::sparse_matrix< num::Number > *& __v, const vmtl::sparse_matrix< num::Number > * __def) const [inline, inherited]

The retrieve methods assign the value of the variable __s with index i to * the variable __v. It is returned whether the variable is set. If * the variable is not set, the value of __def is assigned to __v. Note that the simple types are assigned but for the complex types only pointers are set to the contents of the datamap, in order to reduce unnecessary big copy operations.

Definition at line 2084 of file datamap.hpp.

10.167.3.259 const basic_alltype & coco::datamap::sfind (const std::string & __s) const [inline, inherited]

This method returns the value (as basic_alltype) of the variable __s. If the variable is not set, an empty basic_alltype is returned.

Definition at line 81 of file datamap.hpp.

10.167.3.260 const basic_alltype & coco::datamap::sfind (const char * __s) const [inline, inherited]

This method returns the value (as basic_alltype) of the variable __s. If the variable is not set, an empty basic_alltype is returned.

Definition at line 90 of file datamap.hpp.

10.167.3.261 const basic_alltype & coco::datamap::sfind (const std::string & __s, int *i*) const [inline, inherited]

This method returns the value (as basic_alltype) of the variable __s with index i. If the variable is not set, an empty basic_alltype is returned.

Definition at line 95 of file datamap.hpp.

10.167.3.262 const basic_alltype & coco::datamap::sfind (const char * __s, int *i*) const [inline, inherited]

This method returns the value (as basic_alltype) of the variable __s with index i. If the variable is not set, an empty basic_alltype is returned.

Definition at line 104 of file datamap.hpp.

10.167.3.263 bool coco::datamap::sinsert (const std::string & __s, const basic_alltype & __h, bool *replace*) [inherited]

This method inserts the variable __s with value __h into the datamap. If replace is true, the value of an already defined variable __s would be replaced, if it is false the new value would not be set. The method returns whether the variable __s was indeed set to __h.

Definition at line 36 of file datamap.cc.

10.167.3.264 bool coco::datamap::sinsert (const char * __s, const basic_alltype & __h, bool *replace*) [inline, inherited]

This method inserts the variable __s with value __h into the datamap. If replace is true, the value of an already defined variable __s would be replaced, if it is false the new value would not be set. The method returns whether the variable __s was indeed set to __h.

Definition at line 36 of file datamap.hpp.

10.167.3.265 bool coco::datamap::sinsert (const std::string & __s, int *i*, const basic_alltype & __h, bool *replace*) [inline, inherited]

This method inserts the variable __s with index i with value __h into the datamap. If replace is true, the value of an already defined variable __s with index i would be replaced, if it is false the new value would not be set. The method returns whether the variable __s with index i was indeed set to __h.

Definition at line 42 of file datamap.hpp.

10.167.3.266 bool coco::datamap::sinsert (const char * __s, int *i*, const basic_alltype & __h, bool *replace*) [inline, inherited]

This method inserts the variable __s with index i with value __h into the datamap. If replace is true, the value of an already defined variable __s with index i would be replaced, if it is false the new value

would not be set. The method returns whether the variable __s with index i was indeed set to __h.

Definition at line 50 of file datamap.hpp.

10.167.3.267 bool coco::datamap::which (const std::string & __s, std::vector< int > & __idx) const [inline, inherited]

The which methods sets __idx to the list of indices defined for the variable with name __s. The methods return whether the variable __s is defined.

Definition at line 129 of file datamap.hpp.

10.167.3.268 bool coco::datamap::which (const char * __s, std::vector< int > & __idx) const [inline, inherited]

The which methods sets __idx to the list of indices defined for the variable with name __s. The methods return whether the variable __s is defined.

Definition at line 143 of file datamap.hpp.

The documentation for this class was generated from the following file:

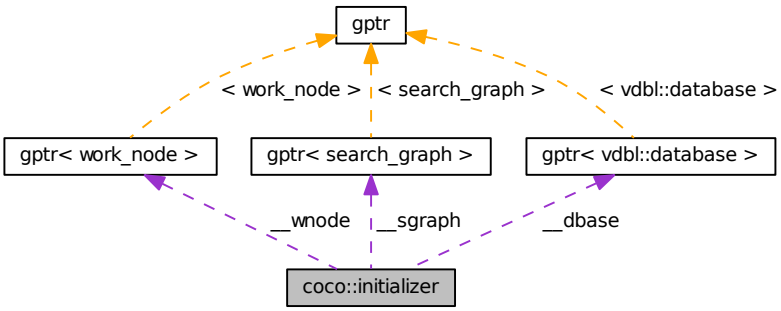
- info_contents.h

10.168 coco::initializer Class Reference

Initializer base class.

`#include <initializer.h>`

Collaboration diagram for coco::initializer:



Public Member Functions

- initializer (const std::string &__n, gptr< work_node > *&wnode, search_focus *&sfoc, gptr< search_graph > *&sgraph, gptr< vdbl::database > *&dbase)

- initializer (const std::string &__n, gptr< work_node > *&wnode, search_focus *&sfoc, gptr< search_graph > *&sgraph)
- initializer (const std::string &__n, gptr< work_node > *&wnode, search_focus *&sfoc, gptr< vdbl::database > *&dbase)
- initializer (const std::string &__n, gptr< work_node > *&wnode, gptr< search_graph > *&sgraph, gptr< vdbl::database > *&dbase)
- initializer (const std::string &__n, search_focus *&sfoc, gptr< search_graph > *&sgraph, gptr< vdbl::database > *&dbase)
- initializer (const std::string &__n, gptr< work_node > *&wnode, search_focus *&sfoc)
- initializer (const std::string &__n, gptr< work_node > *&wnode, gptr< vdbl::database > *&dbase)
- initializer (const std::string &__n, gptr< work_node > *&wnode, gptr< search_graph > *&sgraph)
- initializer (const std::string &__n, search_focus *&sfoc, gptr< search_graph > *&sgraph)
- initializer (const std::string &__n, search_focus *&sfoc, gptr< vdbl::database > *&dbase)
- initializer (const std::string &__n, gptr< search_graph > *&sgraph, gptr< vdbl::database > *&dbase)
- initializer (const std::string &__n, gptr< work_node > *&wnode)
- initializer (const std::string &__n, search_focus *&sfoc)
- initializer (const std::string &__n, gptr< search_graph > *&sgraph)
- initializer (const std::string &__n, gptr< vdbl::database > *&dbase)
- virtual ~initializer ()
- const std::string & get_name () const
- virtual ie_return_type initialize (const control_data &_c)

Protected Attributes

- std::string __name
- gptr< work_node > ** __wnode
- search_focus ** __sfocus
- gptr< vdbl::database > ** __dbase
- gptr< search_graph > ** __sgraph

10.168.1 Detailed Description

This is the base class of all COCONUT initializer modules. An initializer is a management module, which specializes in initializing data structures like the search graph or the search database.

10.168.2 Constructor & Destructor Documentation

10.168.2.1 coco::initializer::initializer (const std::string & *__n*, gptr< work_node > *& *wnode*, search_focus *& *sfoc*, gptr< search_graph > *& *sgraph*, gptr< vdbl::database > *& *dbase*) [inline]

This is the most general constructor for an initializer containing parameters for setting all internal variables. The parameter `__n` sets __name, `wnode` sets __wnode, `sfoc` sets __sfocus, `sgraph` sets __sgraph, and `dbase` sets __dbase.

Definition at line 94 of file initializer.h.

10.168.2.2 coco::initializer::initializer (const std::string & *__n*, gptr< work_node > *& *wnode*, search_focus *& *sfoc*, gptr< search_graph > *& *sgraph*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus and sgraph sets __sgraph.

Definition at line 107 of file initializer.h.

10.168.2.3 coco::initializer::initializer (const std::string & *__n*, gptr< work_node > *& *wnode*, search_focus *& *sfoc*, gptr< vdbl::database > *& *dbase*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus and dbase sets __dbase.

Definition at line 119 of file initializer.h.

10.168.2.4 coco::initializer::initializer (const std::string & *__n*, gptr< work_node > *& *wnode*, gptr< search_graph > *& *sgraph*, gptr< vdbl::database > *& *dbase*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sgraph sets __sgraph and dbase sets __dbase.

Definition at line 131 of file initializer.h.

10.168.2.5 coco::initializer::initializer (const std::string & *__n*, search_focus *& *sfoc*, gptr< search_graph > *& *sgraph*, gptr< vdbl::database > *& *dbase*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, sgraph sets __sgraph and dbase sets __dbase.

Definition at line 143 of file initializer.h.

10.168.2.6 coco::initializer::initializer (const std::string & *__n*, gptr< work_node > *& *wnode*, search_focus *& *sfoc*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus.

Definition at line 154 of file initializer.h.

10.168.2.7 coco::initializer::initializer (const std::string & *__n*, gptr< work_node > *& *wnode*, gptr< vdbl::database > *& *dbase*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, dbase sets __dbase.

Definition at line 164 of file initializer.h.

10.168.2.8 coco::initializer::initializer (const std::string & *__n*, gptr< work_node > *& *wnode*, gptr< search_graph > *& *sgraph*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sgraph sets __sgraph.

Definition at line 174 of file initializer.h.

10.168.2.9 coco::initializer::initializer (const std::string & *__n*, search_focus *& *sfoc*, gptr< search_graph > *& *sgraph*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, sgraph sets __sgraph.

Definition at line 184 of file initializer.h.

10.168.2.10 coco::initializer::initializer (const std::string & *__n*, search_focus *& *sfoc*, gptr< vdbl::database > *& *dbase*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, dbase sets __dbase.

Definition at line 194 of file initializer.h.

10.168.2.11 coco::initializer::initializer (const std::string & *__n*, gptr< search_graph > *& *sgraph*, gptr< vdbl::database > *& *dbase*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name, sgraph sets __sgraph, dbase sets __dbase.

Definition at line 204 of file initializer.h.

10.168.2.12 coco::initializer::initializer (const std::string & *__n*, gptr< work_node > *& *wnode*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name and wnode sets __wnode.

Definition at line 214 of file initializer.h.

10.168.2.13 coco::initializer::initializer (const std::string & *__n*, search_focus *& *sfoc*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name and sfoc sets __sfocus.

Definition at line 222 of file initializer.h.

10.168.2.14 coco::initializer::initializer (const std::string & *__n*, gptr< search_graph > *& *sgraph*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name and sgraph sets __sgraph.

Definition at line 230 of file initializer.h.

10.168.2.15 coco::initializer::initializer (const std::string & *__n*, gptr< vdbl::database > *& *dbase*) [inline]

This is a constructor for an initializer containing parameters for setting some internal variables. The parameter __n sets __name and dbase sets __dbase.

Definition at line 238 of file initializer.h.

10.168.2.16 virtual coco::initializer::~initializer ( ) [inline, virtual]

Standard Destructor

Definition at line 244 of file initializer.h.

10.168.3 Member Function Documentation

10.168.3.1 const std::string& coco::initializer::get_name ( ) const [inline]

The get_name method returns the identifier string of the initializer.

Definition at line 247 of file initializer.h.

10.168.3.2 virtual ie_return_type coco::initializer::initialize (const control_data & _c) [inline, virtual]

This method is the main method of an initializer. It is supposed to initialize various internal data structures. Status information is returned in a ie_return_type structure. This method is overloaded by the various subclasses. Service information and parameters are provided via the control_data structure __c.

Definition at line 254 of file initializer.h.

10.168.4 Member Data Documentation

10.168.4.1 gptr<vdbl::database>** coco::initializer::__dbase [protected]

This variable contains a pointer to a global pointer to the search database.

Definition at line 85 of file initializer.h.

10.168.4.2 std::string coco::initializer::__name [protected]

This is the identifier string for an initializer.

Definition at line 78 of file initializer.h.

10.168.4.3 search_focus** coco::initializer::__sfocus [protected]

This variable contains a pointer to a pointer to the search focus.

Definition at line 82 of file initializer.h.

10.168.4.4 gptr<search_graph>** coco::initializer::__sgraph [protected]

This variable contains a pointer to a global pointer to the search graph.

Definition at line 87 of file initializer.h.

10.168.4.5 gptr<work_node>** coco::initializer::__wnode [protected]

This variable contains a pointer to a global pointer to the work node.

Definition at line 80 of file initializer.h.

The documentation for this class was generated from the following file:

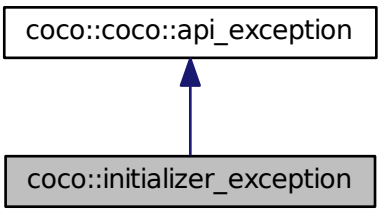
- initializer.h

10.169 coco::initializer_exception Class Reference

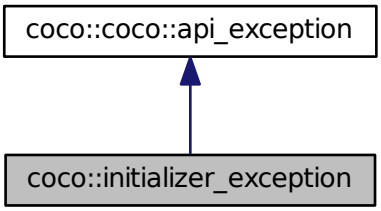
Initializer exception class.

`#include <initializer.h>`

Inheritance diagram for coco::initializer_exception:



Collaboration diagram for coco::initializer_exception:



Public Member Functions

- initializer_exception (const std::string &msg)

- initializer_exception (const char *msg)
- virtual ~initializer_exception () throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()

10.169.1 Detailed Description

This is the exceptions class thrown by initializer subclasses. Every properly coded COCONUT initializer module should only throw exceptions of this type.

10.169.2 Constructor & Destructor Documentation

10.169.2.1 coco::initializer_exception::initializer_exception (const std::string & *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 58 of file initializer.h.

10.169.2.2 coco::initializer_exception::initializer_exception (const char * *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 61 of file initializer.h.

10.169.2.3 virtual coco::initializer_exception::~initializer_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 65 of file initializer.h.

10.169.3 Member Function Documentation

10.169.3.1 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file expression.h.

10.169.3.2 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.169.3.3 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.169.3.4 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.169.3.5 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.169.3.6 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.169.3.7 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.169.3.8 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file expression.h.

10.169.3.9 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.169.3.10 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.169.3.11 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.169.3.12 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.169.3.13 const char * coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

Definition at line 57 of file api_exception.cc.

10.169.3.14 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.169.3.15 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.169.3.16 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.169.3.17 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.169.3.18 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file expression.h.

10.169.3.19 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.169.3.20 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following file:

- initializer.h

10.170 coco::coco::coco::interval Class Reference

Public Member Functions

- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const

- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const

- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()

- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator*= (const interval &a)
- interval & operator*= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const

Static Public Member Functions

- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)

Friends

- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)

- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)

- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)

10.170.1 Detailed Description

Definition at line 113 of file search_graph.cc.

10.170.2 Constructor & Destructor Documentation

10.170.2.1 coco::coco::coco::interval::interval (double *lo*, double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.170.2.2 coco::coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.170.2.3 coco::coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.170.2.4 coco::coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.170.2.5 coco::coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.170.2.6 coco::coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.170.2.7 coco::coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.170.2.8 coco::coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.170.2.9 coco::coco::coco::interval::interval (double *lo*, double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.170.2.10 coco::coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.170.2.11 coco::coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.170.2.12 coco::coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.170.2.13 coco::coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.170.2.14 coco::coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.170.2.15 coco::coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.170.2.16 coco::coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.170.3 Member Function Documentation

10.170.3.1 __Base coco::coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.170.3.2 __Base coco::coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.170.3.3 bool coco::coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.170.3.4 bool coco::coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.170.3.5 double coco::coco::coco::interval::diam ( ) const

10.170.3.6 double coco::coco::coco::interval::diam ( ) const

10.170.3.7 bool coco::coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.170.3.8 bool coco::coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.170.3.9 double coco::coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.170.3.10 double coco::coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.170.3.11 static interval coco::coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.170.3.12 static interval coco::coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.170.3.13 bool coco::coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.170.3.14 bool coco::coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.170.3.15 void coco::coco::coco::interval::get_bounds (double & *a*, double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.170.3.16 void coco::coco::coco::interval::get_bounds (double & *a*, double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.170.3.17 interval coco::coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.170.3.18 interval coco::coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.170.3.19 interval coco::coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.170.3.20 interval coco::coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.170.3.21 interval& coco::coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.170.3.22 interval& coco::coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.170.3.23 interval& coco::coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.170.3.24 interval& coco::coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.170.3.25 interval& coco::coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and `x`.

10.170.3.26 interval& coco::coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and `x`.

10.170.3.27 interval& coco::coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and `x`.

10.170.3.28 interval& coco::coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and `x`.

10.170.3.29 double coco::coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.170.3.30 double coco::coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.170.3.31 interval coco::coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.170.3.32 interval coco::coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.170.3.33 interval& coco::coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.170.3.34 interval& coco::coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.170.3.35 interval& coco::coco::coco::interval::intersect_invcos_wc (const interval & *__i*, double *__l*, double *__d*)

10.170.3.36 interval& coco::coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.170.3.37 interval& coco::coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.170.3.38 interval& coco::coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.170.3.39 interval& coco::coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.170.3.40 interval& coco::coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.170.3.41 interval& coco::coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.170.3.42 interval& coco::coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.170.3.43 interval& coco::coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.170.3.44 interval& coco::coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.170.3.45 interval& coco::coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.170.3.46 interval& coco::coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.170.3.47 interval& coco::coco::coco::interval::intersect_power (const interval & __i, int __n)

10.170.3.48 interval& coco::coco::coco::interval::intersect_power (const interval & __i, int __n)

10.170.3.49 interval& coco::coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.170.3.50 interval& coco::coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.170.3.51 interval& coco::coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.170.3.52 interval& coco::coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.170.3.53 bool coco::coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.170.3.54 bool coco::coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.170.3.55 bool coco::coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.170.3.56 bool coco::coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.170.3.57 bool coco::coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.170.3.58 bool coco::coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.170.3.59 bool coco::coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.170.3.60 bool coco::coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.170.3.61 bool coco::coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.170.3.62 bool coco::coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.170.3.63 bool coco::coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.170.3.64 bool coco::coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.170.3.65 double coco::coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.170.3.66 double coco::coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.170.3.67 double coco::coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.170.3.68 double coco::coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.170.3.69 double coco::coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.170.3.70 double coco::coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.170.3.71 interval& coco::coco::coco::interval::operator*= (const interval & *a*)

10.170.3.72 interval& coco::coco::coco::interval::operator*= (const interval & *a*)

10.170.3.73 interval& coco::coco::coco::interval::operator*= (double *a*)

10.170.3.74 interval& coco::coco::coco::interval::operator*= (double *a*)

10.170.3.75 interval& coco::coco::coco::interval::operator+= (const interval & *a*)

10.170.3.76 interval& coco::coco::coco::interval::operator+= (const interval & *a*)

10.170.3.77 interval& coco::coco::coco::interval::operator+= (double *a*)

10.170.3.78 interval& coco::coco::coco::interval::operator+= (double *a*)

10.170.3.79 interval coco::coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.170.3.80 interval coco::coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.170.3.81 interval& coco::coco::coco::interval::operator-= (const interval & *a*)

10.170.3.82 interval& coco::coco::coco::interval::operator-= (const interval & *a*)

10.170.3.83 interval& coco::coco::coco::interval::operator-= (double *a*)

10.170.3.84 interval& coco::coco::coco::interval::operator-= (double *a*)

10.170.3.85 interval& coco::coco::coco::interval::operator/= (const interval & *a*)

10.170.3.86 interval& coco::coco::coco::interval::operator/= (const interval & *a*)

10.170.3.87 interval& coco::coco::coco::interval::operator/= (double *a*)

10.170.3.88 interval& coco::coco::coco::interval::operator/= (double *a*)

10.170.3.89 interval& coco::coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.170.3.90 interval& coco::coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.170.3.91 interval& coco::coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.170.3.92 interval& coco::coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.170.3.93 interval& coco::coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.170.3.94 interval& coco::coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.170.3.95 interval& coco::coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.170.3.96 interval& coco::coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.170.3.97 interval& coco::coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.170.3.98 interval& coco::coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.170.3.99 interval& coco::coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.170.3.100 interval& coco::coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.170.3.101 static int const& coco::coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.170.3.102 static int const& coco::coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.170.3.103 static int coco::coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.170.3.104 static int coco::coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.170.3.105 double coco::coco::coco::interval::project (double __*d*) const

10.170.3.106 double coco::coco::interval::project (double __*d*) const [inline]

Definition at line 831 of file expression.h.

10.170.3.107 bool coco::coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.170.3.108 bool coco::coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.170.3.109 bool coco::coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.170.3.110 bool coco::coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.170.3.111 double coco::coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.170.3.112 double coco::coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.170.3.113 double coco::coco::coco::interval::rel_width ( ) const

10.170.3.114 double coco::coco::coco::interval::rel_width ( ) const

10.170.3.115 double coco::coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.170.3.116 double coco::coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.170.3.117 interval& coco::coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.170.3.118 interval& coco::coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.170.3.119 double coco::coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.170.3.120 double coco::coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.170.3.121 void coco::coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.170.3.122 void coco::coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.170.3.123 void coco::coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.170.3.124 void coco::coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.170.3.125 void coco::coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.170.3.126 void coco::coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.170.3.127 void coco::coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.170.3.128 void coco::coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.170.3.129 void coco::coco::coco::interval::setpair (double & *lo*, double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.170.3.130 void coco::coco::coco::interval::setpair (double & *lo*, double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.170.3.131 bool coco::coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.170.3.132 bool coco::coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.170.3.133 double coco::coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.170.3.134 double coco::coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.170.3.135 bool coco::coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.170.3.136 bool coco::coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.170.3.137 double coco::coco::coco::interval::width ( ) const

10.170.3.138 double coco::coco::coco::interval::width ( ) const

10.170.4 Friends And Related Function Documentation

10.170.4.1 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.170.4.2 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.170.4.3 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.170.4.4 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.170.4.5 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.170.4.6 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.170.4.7 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.170.4.8 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.170.4.9 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.170.4.10 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.170.4.11 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.170.4.12 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.170.4.13 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.170.4.14 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.170.4.15 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.170.4.16 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.170.4.17 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.170.4.18 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.170.4.19 interval cancel (const interval & *a,* const interval & *b*) [friend]

10.170.4.20 interval cancel (const interval & *a,* const interval & *b*) [friend]

10.170.4.21 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.170.4.22 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.170.4.23 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.170.4.24 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.170.4.25 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.170.4.26 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.170.4.27 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.170.4.28 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.170.4.29 interval division_part1 (const interval & *x,* const interval & *y,* bool & *b*) [friend]

10.170.4.30 interval division_part1 (const interval & *x,* const interval & *y,* bool & *b*) [friend]

10.170.4.31 interval division_part2 (const interval & *x,* const interval & *y,* bool *b*) [friend]

10.170.4.32 interval division_part2 (const interval & *x,* const interval & *y,* bool *b*) [friend]

10.170.4.33 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.170.4.34 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.170.4.35 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.170.4.36 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.170.4.37 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.170.4.38 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.170.4.39 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.170.4.40 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.170.4.41 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.170.4.42 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.170.4.43 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.170.4.44 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.170.4.45 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.170.4.46 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.170.4.47 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.170.4.48 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.170.4.49 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.170.4.50 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.170.4.51 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.170.4.52 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.170.4.53 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

10.170.4.54 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

10.170.4.55 interval operator∗ (const interval & *a*, double *b*) [friend]

10.170.4.56 interval operator∗ (const interval & *a*, double *b*) [friend]

10.170.4.57 interval operator∗ (double *b*, const interval & *a*) [friend]

10.170.4.58 interval operator∗ (double *b*, const interval & *a*) [friend]

10.170.4.59 interval operator+ (const interval & *a*, const interval & *b*) [friend]

10.170.4.60 interval operator+ (const interval & *a*, const interval & *b*) [friend]

10.170.4.61 interval operator+ (const interval & *a*, double *b*) [friend]

10.170.4.62 interval operator+ (const interval & *a*, double *b*) [friend]

10.170.4.63 interval operator+ (double *b*, const interval & *a*) [friend]

10.170.4.64 interval operator+ (double *b*, const interval & *a*) [friend]

10.170.4.65 interval operator- (const interval & *a*, const interval & *b*) [friend]

10.170.4.66 interval operator- (const interval & *a*, const interval & *b*) [friend]

10.170.4.67 interval operator- (const interval & *a*, double *b*) [friend]

10.170.4.68 interval operator- (const interval & *a*, double *b*) [friend]

10.170.4.69 interval operator- (double *b*, const interval & *a*) [friend]

10.170.4.70 interval operator- (double *b*, const interval & *a*) [friend]

10.170.4.71 interval operator/ (const interval & *a*, const interval & *b*) [friend]

10.170.4.72 interval operator/ (const interval & *a*, const interval & *b*) [friend]

10.170.4.73 interval operator/ (const interval & *a*, double *b*) [friend]

10.170.4.74 interval operator/ (const interval & *a*, double *b*) [friend]

10.170.4.75 interval operator/ (double *b*, const interval & *a*) [friend]

10.170.4.76 interval operator/ (double *b*, const interval & *a*) [friend]

10.170.4.77 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.170.4.78 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.170.4.79 interval pow (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.170.4.80 interval pow (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.170.4.81 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.170.4.82 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.170.4.83 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.170.4.84 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.170.4.85 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.170.4.86 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.170.4.87 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.170.4.88 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.170.4.89 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.170.4.90 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.170.4.91 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.170.4.92 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.170.4.93 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.170.4.94 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

The documentation for this class was generated from the following file:

- interval_boost.h

10.171 coco::interval Class Reference

Interval wrapper class.

`#include <interval_boost.h>`

Public Member Functions

- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()

- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const __Base &x)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double sup () const
- double inf () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const

- bool disjoint (const interval &x) const
- bool interior (const interval &x) const
- bool seq (const interval &x) const
- bool sne (const interval &x) const
- bool sge (const interval &x) const
- bool sgt (const interval &x) const
- bool sle (const interval &x) const
- bool slt (const interval &x) const
- bool ceq (const interval &x) const
- bool cne (const interval &x) const
- bool cge (const interval &x) const
- bool cgt (const interval &x) const
- bool cle (const interval &x) const
- bool clt (const interval &x) const
- bool peq (const interval &x) const
- bool pne (const interval &x) const
- bool pge (const interval &x) const
- bool pgt (const interval &x) const
- bool ple (const interval &x) const
- bool plt (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const __Base &_x)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)

- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const __Base &x)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const __Base &_x)

- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const

- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)

- double diam () const
- double width () const

- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)

- double diam () const
- double width () const

Static Public Member Functions

- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)

Friends

- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)

- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- bool operator== (const interval &__i, const interval &__d)
- template<class _TC >
  bool operator== (const interval &__i, const _TC &__d)
- bool operator!= (const interval &__i, const interval &__d)
- template<class _TC >
  bool operator!= (const interval &__i, const _TC &__d)
- std::ostream & operator<< (std::ostream &, const interval &)
- interval tanh (const interval &x)
- interval tan (const interval &x)
- interval sqrt (const interval &x)
- interval sqr (const interval &x)
- interval sinh (const interval &x)
- interval sin (const interval &x)
- interval pow (const interval &x, const interval &y)
- interval power (const interval &x, int n)
- interval log2 (const interval &x)
- interval log1p (const interval &x)
- interval log10 (const interval &x)
- interval log (const interval &x)
- interval expm1 (const interval &x)
- interval exp2 (const interval &x)
- interval exp10 (const interval &x)
- interval exp (const interval &x)
- interval coth (const interval &x)
- interval cot (const interval &x)
- interval cosh (const interval &x)
- interval cos (const interval &x)
- interval atanh (const interval &x)
- interval atan (const interval &x)
- interval asinh (const interval &x)
- interval asin (const interval &x)
- interval acoth (const interval &x)
- interval acot (const interval &x)
- interval acosh (const interval &x)

- interval abs (const interval &x)
- interval acos (const interval &x)
- double dist (const interval &x, const interval &y)
- double mag (const interval &)
- double mig (const interval &)
- double rad (const interval &)
- double relDiam (const interval &)
- double width (const interval &)
- double diam (const interval &)
- double mid (const interval &)
- interval operator/ (double a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (const interval &a, const interval &b)
- interval operator∗ (double a, const interval &b)
- interval operator∗ (const interval &a, double b)
- interval operator∗ (const interval &a, const interval &b)
- interval cancel (const interval &a, const interval &b)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double a, const interval &b)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- double upward_divides (const double &a, const double &b)
- double downward_divides (const double &a, const double &b)
- double upward_multiplies (const double &a, const double &b)
- double downward_multiplies (const double &a, const double &b)
- double upward_minus (const double &a, const double &b)
- double downward_minus (const double &a, const double &b)
- double upward_plus (const double &a, const double &b)
- double downward_plus (const double &a, const double &b)
- interval imin (const interval &x, const interval &y)
- interval imax (const interval &x, const interval &y)

10.171.1 Detailed Description

This is a wrapper class to make the COCONUT environment independent from a <it>specific</it> interval library. Currently, the FILIB++ library works underneath.

This is a wrapper class to make the COCONUT environment independent from a <it>specific</it> interval library.

Definition at line 72 of file interval_filib.h.

10.171.2 Constructor & Destructor Documentation

10.171.2.1 coco::interval::interval (double *lo*, double *up*) [inline]

Definition at line 134 of file interval_boost.h.

10.171.2.2 coco::interval::interval (double *d* = 0) [inline]

Definition at line 135 of file interval_boost.h.

10.171.2.3 coco::interval::interval (int *d*) [inline]

Definition at line 136 of file interval_boost.h.

10.171.2.4 coco::interval::interval (unsigned *d*) [inline]

Definition at line 137 of file interval_boost.h.

10.171.2.5 coco::interval::interval (long *d*) [inline]

Definition at line 138 of file interval_boost.h.

10.171.2.6 coco::interval::interval (unsigned long *d*) [inline]

Definition at line 139 of file interval_boost.h.

10.171.2.7 coco::interval::interval (const interval & *x*) [inline]

Definition at line 140 of file interval_boost.h.

10.171.2.8 coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 141 of file interval_boost.h.

10.171.2.9 coco::interval::interval (double *lo,* double *up*) [inline]

Constructor, setting the interval to [lo,up].

Definition at line 86 of file interval_filib.h.

10.171.2.10 coco::interval::interval (double *d* = 0) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 88 of file interval_filib.h.

10.171.2.11 coco::interval::interval (int *d*) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 90 of file interval_filib.h.

10.171.2.12 coco::interval::interval (unsigned *d*) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 92 of file interval_filib.h.

10.171.2.13 coco::interval::interval (long *d*) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 94 of file interval_filib.h.

10.171.2.14 coco::interval::interval (unsigned long *d*) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 96 of file interval_filib.h.

10.171.2.15 coco::interval::interval (const __Base & *x*) [inline, explicit]

Constructor, setting the interval from a base class interval.

Definition at line 98 of file interval_filib.h.

10.171.2.16 coco::interval::interval (const interval & *x*) [inline]

Standard Copy Constructor

Definition at line 100 of file interval_filib.h.

10.171.2.17 coco::interval::interval (const interval_st & *x*) [inline]

Constructor, setting the interval from a interval_st interval storage.

Definition at line 102 of file interval_filib.h.

10.171.2.18 coco::interval::interval (double *lo*, double *up*) [inline]

Constructor, setting the interval to [lo,up].

Definition at line 68 of file interval_profil.h.

10.171.2.19 coco::interval::interval (double *d* = 0) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 70 of file interval_profil.h.

10.171.2.20 coco::interval::interval (int *d*) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 72 of file interval_profil.h.

10.171.2.21 coco::interval::interval (unsigned *d*) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 74 of file interval_profil.h.

10.171.2.22 coco::interval::interval (long *d*) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 76 of file interval_profil.h.

10.171.2.23 coco::interval::interval (unsigned long *d*) [inline]

Constructor, setting the interval to the thin interval [d,d].

Definition at line 78 of file interval_profil.h.

10.171.2.24 coco::interval::interval (const __Base & *x*) [inline, explicit]

Constructor, setting the interval from a base class interval.

Definition at line 80 of file interval_profil.h.

10.171.2.25 coco::interval::interval (const interval & *x*) [inline]

Standard Copy Constructor

Definition at line 82 of file interval_profil.h.

10.171.2.26 coco::interval::interval (const interval_st & *x*) [inline]

Constructor, setting the interval from a interval_st interval storage.

Definition at line 84 of file interval_profil.h.

10.171.3 Member Function Documentation

10.171.3.1 __Base coco::interval::base ( ) const [inline]

This method returns a base class representing the same interval.

Definition at line 93 of file interval_profil.h.

10.171.3.2 __Base coco::interval::base ( ) const [inline]

This method returns a base class representing the same interval.

Definition at line 116 of file interval_filib.h.

10.171.3.3 __Base coco::interval::base ( ) const [inline]

Definition at line 152 of file interval_boost.h.

10.171.3.4 bool coco::interval::ceq (const interval & *x*) const [inline]

Perform ceq comparison

Definition at line 176 of file interval_filib.h.

10.171.3.5 bool coco::interval::cge (const interval & *x*) const [inline]

Perform cge comparison

Definition at line 182 of file interval_filib.h.

10.171.3.6 bool coco::interval::cgt (const interval & *x*) const [inline]

Perform cgt comparison

Definition at line 185 of file interval_filib.h.

10.171.3.7 bool coco::interval::cle (const interval & *x*) const [inline]

Perform cle comparison

Definition at line 188 of file interval_filib.h.

10.171.3.8 bool coco::interval::clt (const interval & *x*) const [inline]

Perform clt comparison

Definition at line 191 of file interval_filib.h.

10.171.3.9 bool coco::interval::cne (const interval & *x*) const [inline]

Perform cne comparison

Definition at line 179 of file interval_filib.h.

10.171.3.10 bool coco::interval::contains (const interval & *x*) const [inline]

Check whether this interval contains `x`.

Definition at line 202 of file interval_profil.h.

10.171.3.11 bool coco::interval::contains (const interval & *x*) const [inline]

Definition at line 234 of file interval_boost.h.

10.171.3.12 bool coco::interval::contains (const interval & *x*) const [inline]

Check whether this interval contains `x`.

Definition at line 274 of file interval_filib.h.

10.171.3.13 double coco::interval::diam ( ) const

10.171.3.14 double coco::interval::diam ( ) const

Returns an upper bound for the diameter (width) of this interval, i.e.

a.diam() == a.sup()-a.inf()

Special cases in the extended system:

- a.diam() == NaN for a == [EMPTY]
- a.diam() == +INF for any infinite interval

10.171.3.15 double coco::interval::diam ( ) const

Returns an upper bound for the diameter (width) of this interval, i.e.

a.diam() == a.sup()-a.inf()

Special cases in the extended system:

- a.diam() == NaN for a == [EMPTY]
- a.diam() == +INF for any infinite interval

10.171.3.16 bool coco::interval::disjoint (const interval & *x*) const [inline]

This method checks whether this interval and x are disjoint.

Definition at line 133 of file interval_profil.h.

10.171.3.17 bool coco::interval::disjoint (const interval & *x*) const [inline]

This method checks whether this interval and x are disjoint.

Definition at line 151 of file interval_filib.h.

10.171.3.18 bool coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 173 of file interval_boost.h.

10.171.3.19 double coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.171.3.20 double coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.171.3.21 double coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.171.3.22 static interval coco::interval::EMPTY ( ) [inline, static]

This returns a new empty interval.

Definition at line 97 of file interval_profil.h.

10.171.3.23 bool coco::interval::empty ( ) const [inline]

This method checks whether the interval is empty.

Definition at line 100 of file interval_profil.h.

10.171.3.24 static interval coco::interval::EMPTY ( ) [inline, static]

This returns a new empty interval.

Definition at line 118 of file interval_filib.h.

10.171.3.25 bool coco::interval::empty ( ) const [inline]

This method checks whether the interval is empty.

Definition at line 121 of file interval_filib.h.

10.171.3.26 static interval coco::interval::EMPTY ( ) [inline, static]

Definition at line 153 of file interval_boost.h.

10.171.3.27 bool coco::interval::empty ( ) const [inline]

Definition at line 156 of file interval_boost.h.

10.171.3.28 void coco::interval::get_bounds (double & *a*, double & *b*) const [inline]

Set the pair `a` and `b` to the bounds of the interval.

Definition at line 213 of file interval_profil.h.

10.171.3.29 void coco::interval::get_bounds (double & *a*, double & *b*) const [inline]

Definition at line 243 of file interval_boost.h.

10.171.3.30 void coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Set the pair `a` and `b` to the bounds of the interval.

Definition at line 285 of file interval_filib.h.

10.171.3.31 interval coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 180 of file interval_profil.h.

10.171.3.32 interval coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 213 of file interval_boost.h.

10.171.3.33 interval coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 251 of file interval_filib.h.

10.171.3.34 interval coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 187 of file interval_profil.h.

10.171.3.35 interval coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 220 of file interval_boost.h.

10.171.3.36 interval coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 258 of file interval_filib.h.

10.171.3.37 interval& coco::interval::hulldiffwith (const interval & *x*) [inline]

`y.hulldiffwith(x)` constructs the hull of the set difference of `y` and `x`.

Definition at line 157 of file interval_profil.h.

10.171.3.38 interval& coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 192 of file interval_boost.h.

10.171.3.39 interval& coco::interval::hulldiffwith (const interval & *x*) [inline]

`y.hulldiffwith(x)` constructs the hull of the set difference of `y` and `x`.

Definition at line 229 of file interval_filib.h.

10.171.3.40 interval& coco::interval::hullwith (const interval & *x*) [inline]

`y.hullwith(x)` builds the hull of `y` with `x`.

Definition at line 150 of file interval_profil.h.

10.171.3.41 interval& coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 186 of file interval_boost.h.

10.171.3.42 interval& coco::interval::hullwith (const interval & *x*) [inline]

`y.hullwith(x)` builds the hull of `y` with `x`.

Definition at line 222 of file interval_filib.h.

10.171.3.43 interval& coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and `x`.

10.171.3.44 interval& coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and `x`.

10.171.3.45 interval& coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and `x`.

10.171.3.46 interval& coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and `x`.

10.171.3.47 interval& coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and `x`.

10.171.3.48 interval& coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and `x`.

10.171.3.49 double coco::interval::inf ( ) const [inline]

This method returns the lower bound of the interval.

Definition at line 113 of file interval_filib.h.

10.171.3.50 double coco::interval::inf ( ) const [inline]

Definition at line 150 of file interval_boost.h.

10.171.3.51 bool coco::interval::interior (const interval & *x*) const [inline]

This method checks whether this interval is contained in the interior of `x`.

Definition at line 154 of file interval_filib.h.

10.171.3.52 interval coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 172 of file interval_profil.h.

10.171.3.53 interval coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 207 of file interval_boost.h.

10.171.3.54 interval coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 245 of file interval_filib.h.

10.171.3.55 interval& coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

`i.intersect_div(j,k)` computes the interval hull of the set $i \cap (j/k)$.

10.171.3.56 interval& coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.171.3.57 interval& coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

`i.intersect_div(j,k)` computes the interval hull of the set $i \cap (j/k)$.

10.171.3.58 interval& coco::interval::intersect_invcos_wc (const interval & *__i*, double *__l*, double *__d*)

`i.intersect_infcos_wc(j,l,d)` computes the interval hull of the set $i \cap f(j,l,n)$, where f is the inverse image of j under the function $x \mapsto \cos(lx+d)$.

10.171.3.59 interval& coco::interval::intersect_invcos_wc (const interval & *__i*, double *__l*, double *__d*)

10.171.3.60 interval& coco::interval::intersect_invcos_wc (const interval & *__i*, double *__l*, double *__d*)

`i.intersect_infcos_wc(j,l,d)` computes the interval hull of the set $i \cap f(j,l,n)$, where f is the inverse image of j under the function $x \mapsto \cos(lx+d)$.

10.171.3.61 interval& coco::interval::intersect_invcosh_wc (const interval & *__i*, double *__l*, double *__d*)

10.171.3.62 interval& coco::interval::intersect_invcosh_wc (const interval & *__i*, double *__l*, double *__d*)

`i.intersect_infcosh_wc(j,l,n)` computes the interval hull of the set $i \cap f(j,l,n)$, where f is the inverse image of j under the function $x \mapsto cosh(lx+d)$.

10.171.3.63 interval& coco::interval::intersect_invgauss_wc (const interval & *__i*, double *__l*, double *__m*, double *__s*)

`i.intersect_infgauss_wc(j,l,m,s)` computes the interval hull of the set $i \cap f(j,l,m,s)$, where f is the inverse image of j under the function $x \mapsto e^{-(\frac{lx-m}{s})^2}$.

10.171.3.64 interval& coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.171.3.65 interval& coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

`i.intersect_infgauss_wc(j,l,m,s)` computes the interval hull of the set $i \cap f(j,l,m,s)$, where f is the inverse image of j under the function $x \mapsto e^{-(\frac{lx-m}{s})^2}$.

10.171.3.66 interval& coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

`i.intersect_infpower_wc(j,l,n)` computes the interval hull of the set $i \cap f(j,l,n)$, where f is the inverse image of j under the function $x \mapsto (lx)^n$.

10.171.3.67 interval& coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.171.3.68 interval& coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

`i.intersect_infpower_wc(j,l,n)` computes the interval hull of the set $i \cap f(j,l,n)$, where f is the inverse image of j under the function $x \mapsto (lx)^n$.

10.171.3.69 interval& coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

`i.intersect_infsin_wc(j,l,d)` computes the interval hull of the set $i \cap f(j,l,n)$, where f is the inverse image of j under the function $x \mapsto \sin(lx+d)$.

10.171.3.70 interval& coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.171.3.71 interval& coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

`i.intersect_infsin_wc(j,l,d)` computes the interval hull of the set $i \cap f(j,l,n)$, where f is the inverse image of j under the function $x \mapsto \sin(lx+d)$.

10.171.3.72 interval& coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

`i.intersect_infsqr_wc(j,l,d)` computes the interval hull of the set $i \cap f(j,l,d)$, where f is the inverse image of j under the function $x \mapsto (lx+d)^2$.

10.171.3.73 interval& coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.171.3.74 interval& coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

`i.intersect_infsqr_wc(j,l,d)` computes the interval hull of the set $i \cap f(j,l,d)$, where f is the inverse image of j under the function $x \mapsto (lx+d)^2$.

10.171.3.75 interval& coco::interval::intersect_power (const interval & __i, int __n)

`i.intersect_power(j,n)` computes the interval hull of the set $i \cap j^n$.

10.171.3.76 interval& coco::interval::intersect_power (const interval & __i, int __n)

10.171.3.77 interval& coco::interval::intersect_power (const interval & __i, int __n)

`i.intersect_power(j,n)` computes the interval hull of the set $i \cap j^n$.

10.171.3.78 interval& coco::interval::intersectwith (const interval & x) [inline]

`y.intersectwith(x)` intersects `y` with `x`.

Definition at line 142 of file interval_profil.h.

10.171.3.79 interval& coco::interval::intersectwith (const interval & x) [inline]

Definition at line 179 of file interval_boost.h.

10.171.3.80 interval& coco::interval::intersectwith (const interval & x) [inline]

`y.intersectwith(x)` intersects `y` with `x`.

Definition at line 216 of file interval_filib.h.

10.171.3.81 interval& coco::interval::ipow (int n)

The method changes this interval to the `n-th` power.

10.171.3.82 interval& coco::interval::ipow (int n)

The method changes this interval to the `n-th` power.

10.171.3.83 interval& coco::interval::ipow (int n)

The method changes this interval to the `n-th` power.

10.171.3.84 bool coco::interval::is_bounded ( ) const [inline]

This method checks whether the interval is bounded.

Definition at line 118 of file interval_profil.h.

10.171.3.85 bool coco::interval::is_bounded ( ) const [inline]

This method checks whether the interval is bounded.

Definition at line 136 of file interval_filib.h.

10.171.3.86 bool coco::interval::is_bounded ( ) const [inline]

Definition at line 162 of file interval_boost.h.

10.171.3.87 bool coco::interval::is_empty ( ) const [inline]

This method checks whether the interval is empty.

Definition at line 102 of file interval_profil.h.

10.171.3.88 bool coco::interval::is_empty ( ) const [inline]

This method checks whether the interval is empty.

Definition at line 123 of file interval_filib.h.

10.171.3.89 bool coco::interval::is_empty ( ) const [inline]

Definition at line 157 of file interval_boost.h.

10.171.3.90 bool coco::interval::is_entire ( ) const [inline]

This method checks whether the interval is all $\mathbb{R}$.

Definition at line 116 of file interval_profil.h.

10.171.3.91 bool coco::interval::is_entire ( ) const [inline]

This method checks whether the interval is all $\mathbb{R}$.

Definition at line 134 of file interval_filib.h.

10.171.3.92 bool coco::interval::is_entire ( ) const [inline]

Definition at line 161 of file interval_boost.h.

10.171.3.93 bool coco::interval::is_thin ( ) const [inline]

This method checks whether the interval is thin (contains only one point).

Definition at line 106 of file interval_profil.h.

10.171.3.94 bool coco::interval::is_thin ( ) const [inline]

This method checks whether the interval is thin (contains only one point).

Definition at line 126 of file interval_filib.h.

10.171.3.95 bool coco::interval::is_thin ( ) const [inline]

Definition at line 158 of file interval_boost.h.

10.171.3.96 bool coco::interval::is_unbounded_above ( ) const [inline]

This method checks whether the interval is unbounded above ($[a, \infty]$).

Definition at line 114 of file interval_profil.h.

10.171.3.97 bool coco::interval::is_unbounded_above ( ) const [inline]

This method checks whether the interval is unbounded above ($[a, \infty]$).

Definition at line 132 of file interval_filib.h.

10.171.3.98 bool coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 160 of file interval_boost.h.

10.171.3.99 bool coco::interval::is_unbounded_below ( ) const [inline]

This method checks whether the interval is unbounded below ($[-\infty, a]$).

Definition at line 111 of file interval_profil.h.

10.171.3.100 bool coco::interval::is_unbounded_below ( ) const [inline]

This method checks whether the interval is unbounded below ($[-\infty, a]$).

Definition at line 129 of file interval_filib.h.

10.171.3.101 bool coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 159 of file interval_boost.h.

10.171.3.102 double coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.171.3.103 double coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.171.3.104 double coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.171.3.105 double coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.171.3.106 double coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.171.3.107 double coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.171.3.108 double coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.171.3.109 double coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in x }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.171.3.110 double coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in x }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.171.3.111 interval& coco::interval::operator∗= (const interval & *a*)

unary operator∗=

10.171.3.112 interval& coco::interval::operator∗= (const interval & *a*)

10.171.3.113 interval& coco::interval::operator∗= (double *a*)

unary operator∗= for number

10.171.3.114 interval& coco::interval::operator∗= (double *a*)

10.171.3.115 interval& coco::interval::operator∗= (const interval & *a*)

unary operator∗=

10.171.3.116 interval& coco::interval::operator∗= (double *a*)

unary operator∗= for number

10.171.3.117 interval& coco::interval::operator+= (const interval & *a*)

unary +=

10.171.3.118 interval& coco::interval::operator+= (double *a*)

unary += for number

10.171.3.119 interval& coco::interval::operator+= (const interval & *a*)

10.171.3.120 interval& coco::interval::operator+= (double *a*)

10.171.3.121 interval& coco::interval::operator+= (const interval & *a*)

unary +=

10.171.3.122 interval& coco::interval::operator+= (double *a*)

unary += for number

10.171.3.123 interval coco::interval::operator- ( ) const [inline]

Unary interval minus operation

Definition at line 234 of file interval_profil.h.

10.171.3.124 interval coco::interval::operator- ( ) const [inline]

Definition at line 257 of file interval_boost.h.

10.171.3.125 interval coco::interval::operator- ( ) const [inline]

Unary interval minus operation

Definition at line 306 of file interval_filib.h.

10.171.3.126 interval& coco::interval::operator-= (const interval & *a*)

unary -=

10.171.3.127 interval& coco::interval::operator-= (double *a*)

unary -= for number

10.171.3.128 interval& coco::interval::operator-= (const interval & *a*)

10.171.3.129 interval& coco::interval::operator-= (double *a*)

10.171.3.130 interval& coco::interval::operator-= (const interval & *a*)

unary -=

10.171.3.131 interval& coco::interval::operator-= (double *a*)

unary -= for number

10.171.3.132 interval& coco::interval::operator/= (const interval & *a*)

unary operator/=

10.171.3.133 interval& coco::interval::operator/= (const interval & *a*)

10.171.3.134 interval& coco::interval::operator/= (double *a*)

unary operator/= for number

10.171.3.135 interval& coco::interval::operator/= (double *a*)

10.171.3.136 interval& coco::interval::operator/= (const interval & *a*)

unary operator/=

10.171.3.137 interval& coco::interval::operator/= (double *a*)

unary operator/= for number

10.171.3.138 interval& coco::interval::operator= (double *d*) [inline]

Assign the thin interval [d,d].

Definition at line 223 of file interval_profil.h.

10.171.3.139 interval& coco::interval::operator= (int *d*) [inline]

Assign the thin interval [d,d].

Definition at line 224 of file interval_profil.h.

10.171.3.140 interval& coco::interval::operator= (unsigned *d*) [inline]

Assign the thin interval [d,d].

Definition at line 225 of file interval_profil.h.

10.171.3.141 interval& coco::interval::operator= (long *d*) [inline]

Assign the thin interval [d,d].

Definition at line 226 of file interval_profil.h.

10.171.3.142 interval& coco::interval::operator= (unsigned long *d*) [inline]

Assign the thin interval [d,d].

Definition at line 227 of file interval_profil.h.

10.171.3.143 interval& coco::interval::operator= (const __Base & *_x*) [inline]

Assignment from the base interval class

Definition at line 230 of file interval_profil.h.

10.171.3.144 interval& coco::interval::operator= (const interval & *x*) [inline]

Definition at line 248 of file interval_boost.h.

10.171.3.145 interval& coco::interval::operator= (double *d*) [inline]

Definition at line 249 of file interval_boost.h.

10.171.3.146 interval& coco::interval::operator= (int *d*) [inline]

Definition at line 250 of file interval_boost.h.

10.171.3.147 interval& coco::interval::operator= (unsigned *d*) [inline]

Definition at line 251 of file interval_boost.h.

10.171.3.148 interval& coco::interval::operator= (long *d*) [inline]

Definition at line 252 of file interval_boost.h.

10.171.3.149 interval& coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 253 of file interval_boost.h.

10.171.3.150 interval& coco::interval::operator= (double *d*) [inline]

Assign the thin interval [d,d].

Definition at line 295 of file interval_filib.h.

10.171.3.151 interval& coco::interval::operator= (int *d*) [inline]

Assign the thin interval [d,d].

Definition at line 296 of file interval_filib.h.

10.171.3.152 interval& coco::interval::operator= (unsigned *d*) [inline]

Assign the thin interval [d,d].

Definition at line 297 of file interval_filib.h.

10.171.3.153 interval& coco::interval::operator= (long *d*) [inline]

Assign the thin interval [d,d].

Definition at line 298 of file interval_filib.h.

10.171.3.154 interval& coco::interval::operator= (unsigned long *d*) [inline]

Assign the thin interval [d,d].

Definition at line 299 of file interval_filib.h.

10.171.3.155 interval& coco::interval::operator= (const __Base & *_x*) [inline]

Assignment from the base interval class

Definition at line 302 of file interval_filib.h.

10.171.3.156 bool coco::interval::peq (const interval & *x*) const [inline]

Perform peq comparison

Definition at line 194 of file interval_filib.h.

10.171.3.157 bool coco::interval::pge (const interval & *x*) const [inline]

Perform pge comparison

Definition at line 200 of file interval_filib.h.

10.171.3.158 bool coco::interval::pgt (const interval & *x*) const [inline]

Perform pgt comparison

Definition at line 203 of file interval_filib.h.

10.171.3.159 bool coco::interval::ple (const interval & *x*) const [inline]

Perform ple comparison

Definition at line 206 of file interval_filib.h.

10.171.3.160 bool coco::interval::plt (const interval & *x*) const [inline]

Perform plt comparison

Definition at line 209 of file interval_filib.h.

10.171.3.161 bool coco::interval::pne (const interval & *x*) const [inline]

Perform pne comparison

Definition at line 197 of file interval_filib.h.

10.171.3.162 static int const& coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.171.3.163 static int coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.171.3.164 static int const& coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.171.3.165 static int coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.171.3.166 static int const& coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.171.3.167 static int coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.171.3.168 double coco::interval::project (double __*d*) const

This method returns the projection of __d to the interval, i.e. to the double number in the interval which is closest to __d.

10.171.3.169 double coco::interval::project (double __*d*) const

10.171.3.170 double coco::interval::project (double __*d*) const

This method returns the projection of __d to the interval, i.e. to the double number in the interval which is closest to __d.

10.171.3.171 bool coco::interval::proper_subset (const interval & *x*) const [inline]

This method checks whether this interval is a proper subset of x.

Definition at line 127 of file interval_profil.h.

10.171.3.172 bool coco::interval::proper_subset (const interval & *x*) const [inline]

This method checks whether this interval is a proper subset of x.

Definition at line 145 of file interval_filib.h.

10.171.3.173 bool coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 169 of file interval_boost.h.

10.171.3.174 bool coco::interval::proper_superset (const interval & *x*) const [inline]

This method checks whether this interval is a proper superset of x.

Definition at line 130 of file interval_profil.h.

10.171.3.175 bool coco::interval::proper_superset (const interval & *x*) const [inline]

This method checks whether this interval is a proper superset of x.

Definition at line 148 of file interval_filib.h.

10.171.3.176 bool coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 171 of file interval_boost.h.

10.171.3.177 double coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.171.3.178 double coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.171.3.179 double coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.171.3.180 double coco::interval::rel_width ( ) const

`i.rel_width()` computes the relative width (diameter) of `i`.

10.171.3.181 double coco::interval::rel_width ( ) const

10.171.3.182 double coco::interval::rel_width ( ) const

`i.rel_width()` computes the relative width (diameter) of `i`.

10.171.3.183 double coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.171.3.184 double coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.171.3.185 double coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.171.3.186 interval& coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.171.3.187 interval& coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.171.3.188 double coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.171.3.189 double coco::interval::safeguarded_mid ( ) const

Returns the safeguarded midpoint of this interval, i.e.

a.safeguarded_mid() ==

- 0 if a contains 0,
- $2 * \overline{a}$ if a is unbounded below and does not contain 0,
- $2 * \underline{a}$ if a is unbounded above and does not contain 0,
- the geometric mean of the endpoints if contained in a,
- the arithmetic mean of the endpoints otherwise.

10.171.3.190 double coco::interval::safeguarded_mid ( ) const

Returns the safeguarded midpoint of this interval, i.e.

a.safeguarded_mid() ==

- 0 if a contains 0,
- $2 * \overline{a}$ if a is unbounded below and does not contain 0,
- $2 * \underline{a}$ if a is unbounded above and does not contain 0,
- the geometric mean of the endpoints if contained in a,
- the arithmetic mean of the endpoints otherwise.

10.171.3.191 bool coco::interval::seq (const interval & *x*) const [inline]

Perform seq comparison

Definition at line 158 of file interval_filib.h.

10.171.3.192 void coco::interval::set (double *lo*, double *up*) [inline]

`i.set(lo,up)` sets `i` to [lo,up].

Definition at line 87 of file interval_profil.h.

10.171.3.193 void coco::interval::set (double *lo*, double *up*) [inline]

`i.set(lo,up)` sets `i` to [lo,up].

Definition at line 105 of file interval_filib.h.

10.171.3.194 void coco::interval::set (double *lo*, double *up*) [inline]

Definition at line 144 of file interval_boost.h.

10.171.3.195 void coco::interval::set_bounds (double *a*, double *b*) [inline]

Set the bounds of the interval to a and b.

Definition at line 215 of file interval_profil.h.

10.171.3.196 void coco::interval::set_bounds (double *a*, double *b*) [inline]

Definition at line 244 of file interval_boost.h.

10.171.3.197 void coco::interval::set_bounds (double *a*, double *b*) [inline]

Set the bounds of the interval to `a` and `b`.

Definition at line 287 of file interval_filib.h.

10.171.3.198 void coco::interval::set_lb (double *d*) [inline]

Set the lower bound of the interval to `d`.

Definition at line 217 of file interval_profil.h.

10.171.3.199 void coco::interval::set_lb (double *d*) [inline]

Definition at line 245 of file interval_boost.h.

10.171.3.200 void coco::interval::set_lb (double *d*) [inline]

Set the lower bound of the interval to `d`.

Definition at line 289 of file interval_filib.h.

10.171.3.201 void coco::interval::set_ub (double *d*) [inline]

Set the upper bound of the interval to `d`.

Definition at line 219 of file interval_profil.h.

10.171.3.202 void coco::interval::set_ub (double *d*) [inline]

Definition at line 246 of file interval_boost.h.

10.171.3.203 void coco::interval::set_ub (double *d*) [inline]

Set the upper bound of the interval to `d`.

Definition at line 291 of file interval_filib.h.

10.171.3.204 void coco::interval::setpair (double & *lo*, double & *up*) const [inline]

Set the pair `lo` and `up` to the bounds of the interval.

Definition at line 211 of file interval_profil.h.

10.171.3.205 void coco::interval::setpair (double & *lo*, double & *up*) const [inline]

Definition at line 242 of file interval_boost.h.

10.171.3.206 void coco::interval::setpair (double & *lo*, double & *up*) const [inline]

Set the pair `lo` and `up` to the bounds of the interval.

Definition at line 283 of file interval_filib.h.

10.171.3.207 bool coco::interval::sge (const interval & *x*) const [inline]

Perform sge comparison

Definition at line 164 of file interval_filib.h.

10.171.3.208 bool coco::interval::sgt (const interval & *x*) const [inline]

Perform sgt comparison

Definition at line 167 of file interval_filib.h.

10.171.3.209 bool coco::interval::sle (const interval & *x*) const [inline]

Perform sle comparison

Definition at line 170 of file interval_filib.h.

10.171.3.210 bool coco::interval::slt (const interval & *x*) const [inline]

Perform slt comparison

Definition at line 173 of file interval_filib.h.

10.171.3.211 bool coco::interval::sne (const interval & *x*) const [inline]

Perform sne comparison

Definition at line 161 of file interval_filib.h.

10.171.3.212 bool coco::interval::subset (const interval & *x*) const [inline]

This method checks whether this interval is a subset of `x`.

Definition at line 121 of file interval_profil.h.

10.171.3.213 bool coco::interval::subset (const interval & *x*) const [inline]

This method checks whether this interval is a subset of `x`.

Definition at line 139 of file interval_filib.h.

10.171.3.214 bool coco::interval::subset (const interval & *x*) const [inline]

Definition at line 165 of file interval_boost.h.

10.171.3.215 double coco::interval::sup ( ) const [inline]

This method returns the upper bound of the interval.

Definition at line 111 of file interval_filib.h.

10.171.3.216 double coco::interval::sup ( ) const [inline]

Definition at line 151 of file interval_boost.h.

10.171.3.217 bool coco::interval::superset (const interval & *x*) const [inline]

This method checks whether this interval is a superset of x.

Definition at line 124 of file interval_profil.h.

10.171.3.218 bool coco::interval::superset (const interval & *x*) const [inline]

This method checks whether this interval is a superset of x.

Definition at line 142 of file interval_filib.h.

10.171.3.219 bool coco::interval::superset (const interval & *x*) const [inline]

Definition at line 167 of file interval_boost.h.

10.171.3.220 double coco::interval::width ( ) const

10.171.3.221 double coco::interval::width ( ) const

Returns an upper bound for the diameter (width) of this interval, i.e.

a.diam() == a.sup()-a.inf()

Special cases in the extended system:

- a.diam() == NaN for a == [EMPTY]
- a.diam() == +INF for any infinite interval

10.171.3.222 double coco::interval::width ( ) const

Returns an upper bound for the diameter (width) of this interval, i.e.

a.diam() == a.sup()-a.inf()

Special cases in the extended system:

- a.diam() == NaN for a == [EMPTY]
- a.diam() == +INF for any infinite interval

10.171.4 Friends And Related Function Documentation

10.171.4.1 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.171.4.2 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.171.4.3 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.171.4.4 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.171.4.5 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.171.4.6 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.171.4.7 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.171.4.8 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.171.4.9 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.171.4.10 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.171.4.11 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.171.4.12 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.171.4.13 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.171.4.14 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.171.4.15 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.171.4.16 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.171.4.17 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.171.4.18 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.171.4.19 interval cancel (const interval & *a,* const interval & *b*) [friend]

10.171.4.20 interval cancel (const interval & *a,* const interval & *b*) [friend]

10.171.4.21 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.171.4.22 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.171.4.23 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.171.4.24 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.171.4.25 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.171.4.26 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.171.4.27 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.171.4.28 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.171.4.29 double diam (const interval &) [friend]

diam(a) == a.diam()

10.171.4.30 double dist (const interval & *x*, const interval & *y*) [friend]

Same as x.dist(y)

10.171.4.31 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.171.4.32 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.171.4.33 double downward_divides (const double & *a*, const double & *b*) [friend]

This function divides `a` by `b`, rounding downwards.

10.171.4.34 double downward_minus (const double & *a*, const double & *b*) [friend]

This function subtracts `b` from `a`, rounding downwards.

10.171.4.35 double downward_multiplies (const double & *a*, const double & *b*) [friend]

This function multiplies `a` by `b`, rounding downwards.

10.171.4.36 double downward_plus (const double & *a*, const double & *b*) [friend]

This function adds `a` and `b`, rounding downwards.

10.171.4.37 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.171.4.38 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.171.4.39 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.171.4.40 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.171.4.41 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.171.4.42 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.171.4.43 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.171.4.44 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.171.4.45 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.171.4.46 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.171.4.47 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.171.4.48 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.171.4.49 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.171.4.50 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.171.4.51 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.171.4.52 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.171.4.53 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.171.4.54 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.171.4.55 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.171.4.56 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.171.4.57 double mag (const interval &) [friend]

mag(a) == a.mag()

10.171.4.58 double mid (const interval &) [friend]

mid(a) == a.mid()

10.171.4.59 double mig (const interval &) [friend]

mig(a) == a.mig()

10.171.4.60 bool operator!= (const interval & *__i*, const interval & *__d*) [friend]

Comparison operator (not equal)

Definition at line 519 of file expression.h.

10.171.4.61 template<class _TC > bool operator!= (const interval & *__i*, const _TC & *__d*) [friend]

10.171.4.62 interval operator* (const interval & *a*, const interval & *b*) [friend]

This operator computes a*b as interval.

10.171.4.63 interval operator* (const interval & *a*, double *b*) [friend]

10.171.4.64 interval operator* (double *b*, const interval & *a*) [friend]

10.171.4.65 interval operator* (double *a*, const interval & *b*) [friend]

10.171.4.66 interval operator* (const interval & *a*, double *b*) [friend]

10.171.4.67 interval operator* (const interval & *a*, const interval & *b*) [friend]

This operator computes a*b as interval.

10.171.4.68 interval operator+ (const interval & *a*, const interval & *b*) [friend]

This operator computes a+b as interval.

10.171.4.69 interval operator+ (const interval & *a*, double *b*) [friend]

10.171.4.70 interval operator+ (double *b*, const interval & *a*) [friend]

10.171.4.71 interval operator+ (const interval & *a*, const interval & *b*) [friend]

This operator computes a+b as interval.

10.171.4.72 interval operator+ (const interval & *a*, double *b*) [friend]

10.171.4.73 interval operator+ (double *b*, const interval & *a*) [friend]

10.171.4.74 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.171.4.75 interval operator- (const interval & *a*, double *b*) [friend]

10.171.4.76 interval operator- (double *b*, const interval & *a*) [friend]

10.171.4.77 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.171.4.78 interval operator- (const interval & *a*, double *b*) [friend]

10.171.4.79 interval operator- (double *a*, const interval & *b*) [friend]

10.171.4.80 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.171.4.81 interval operator/ (const interval & *a*, double *b*) [friend]

10.171.4.82 interval operator/ (double *b*, const interval & *a*) [friend]

10.171.4.83 interval operator/ (double *a*, const interval & *b*) [friend]

10.171.4.84 interval operator/ (const interval & *a*, double *b*) [friend]

10.171.4.85 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.171.4.86 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.171.4.87 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.171.4.88 bool operator== (const interval & *__i*, const interval & *__d*) [friend]

friends

Comparison operator (equal)

Definition at line 515 of file expression.h.

10.171.4.89 template<class _TC > bool operator== (const interval & __i, const _TC & __d) [friend]

10.171.4.90 interval pow (const interval & x, const interval & y) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.171.4.91 interval pow (const interval & x, const interval & y) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.171.4.92 interval power (const interval & x, int n) [friend]

Returns an interval enclosure of x^n.

10.171.4.93 interval power (const interval & x, int n) [friend]

Returns an interval enclosure of x^n.

10.171.4.94 double rad (const interval &) [friend]

rad(a) == a.rad()

10.171.4.95 double relDiam (const interval &) [friend]

relDiam(a) == a.relDiam()

10.171.4.96 interval sin (const interval & x) [friend]

Returns an interval enclosure of the sine of the interval x.

10.171.4.97 interval sin (const interval & x) [friend]

Returns an interval enclosure of the sine of the interval x.

10.171.4.98 interval sinh (const interval & x) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.171.4.99 interval sinh (const interval & x) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.171.4.100 interval sqr (const interval & x) [friend]

Returns an interval enclosure of x^2.

10.171.4.101 interval sqr (const interval & x) [friend]

Returns an interval enclosure of x^2.

10.171.4.102 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.171.4.103 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.171.4.104 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.171.4.105 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.171.4.106 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.171.4.107 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.171.4.108 double upward_divides (const double & *a*, const double & *b*) [friend]

This function divides a by b, rounding upwards.

10.171.4.109 double upward_minus (const double & *a*, const double & *b*) [friend]

This function subtracts b from a, rounding upwards.

10.171.4.110 double upward_multiplies (const double & *a*, const double & *b*) [friend]

This function multiplies a by b, rounding upwards.

10.171.4.111 double upward_plus (const double & *a*, const double & *b*) [friend]

This function adds a and b, rounding upwards.

10.171.4.112 double width (const interval &) [friend]

The documentation for this class was generated from the following files:

- interval_boost.h
- interval_filib.h
- interval_profil.h

10.172 coco::coco::interval Class Reference

```
#include <expression.h>
```

Public Member Functions

- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()

- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const

- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const

- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)

- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)

- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)

- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)

- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)

- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator*= (const interval &a)
- interval & operator*= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)

- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)

- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator*= (const interval &a)
- interval & operator*= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const

- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)

- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)

- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const

- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator*= (const interval &a)
- interval & operator*= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)

- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)

- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const
- interval (double lo, double up)
- interval (double d=0)
- interval (int d)
- interval (unsigned d)
- interval (long d)
- interval (unsigned long d)
- interval (const interval &x)
- interval (const interval_st &x)
- void set (double lo, double up)
- double inf () const
- double sup () const
- __Base base () const
- bool empty () const
- bool is_empty () const
- bool is_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool subset (const interval &x) const
- bool superset (const interval &x) const
- bool proper_subset (const interval &x) const
- bool proper_superset (const interval &x) const
- bool disjoint (const interval &x) const
- double rel_width () const
- interval & intersectwith (const interval &x)
- interval & hullwith (const interval &x)
- interval & hulldiffwith (const interval &x)
- interval intersect (const interval &x) const
- interval hull (const interval &x) const
- interval hulldiff (const interval &x) const
- bool contains (const interval &x) const
- void setpair (double &lo, double &up) const
- void get_bounds (double &a, double &b) const

- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- interval & operator= (const interval &x)
- interval & operator= (double d)
- interval & operator= (int d)
- interval & operator= (unsigned d)
- interval & operator= (long d)
- interval & operator= (unsigned long d)
- interval operator- () const
- interval & ipow (int n)
- interval & imin (const interval &x)
- interval & imax (const interval &x)
- interval & round_to_integer ()
- interval & intersect_div (const interval &__i, const interval &__j)
- interval & intersect_power (const interval &__i, int __n)
- interval & intersect_invsqr_wc (const interval &__i, double __l, double __d)
- interval & intersect_invpower_wc (const interval &__i, double __l, int __n)
- interval & intersect_invsin_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcos_wc (const interval &__i, double __l, double __d)
- interval & intersect_invcosh_wc (const interval &__i, double __l, double __d)
- interval & intersect_invgauss_wc (const interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- interval & operator+= (const interval &a)
- interval & operator+= (double a)
- interval & operator-= (const interval &a)
- interval & operator-= (double a)
- interval & operator∗= (const interval &a)
- interval & operator∗= (double a)
- interval & operator/= (const interval &a)
- interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const interval &x) const
- double safeguarded_mid () const

Static Public Member Functions

- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()

- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)
- static interval EMPTY ()
- static int const & precision ()
- static int precision (int const &p)

Friends

- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator∗ (const interval &a, const interval &b)
- interval operator∗ (const interval &a, double b)
- interval operator∗ (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)

- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)

- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)

- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)

- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)

- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)

- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)

- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator∗ (const interval &a, const interval &b)
- interval operator∗ (const interval &a, double b)
- interval operator∗ (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)

- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)

- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator* (const interval &a, const interval &b)
- interval operator* (const interval &a, double b)
- interval operator* (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)

- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)
- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator∗ (const interval &a, const interval &b)
- interval operator∗ (const interval &a, double b)
- interval operator∗ (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)
- std::ostream & operator<< (std::ostream &s, const interval &a)

- interval operator+ (const interval &a, const interval &b)
- interval operator+ (const interval &a, double b)
- interval operator+ (double b, const interval &a)
- interval operator- (const interval &a, const interval &b)
- interval operator- (const interval &a, double b)
- interval operator- (double b, const interval &a)
- interval cancel (const interval &a, const interval &b)
- interval operator∗ (const interval &a, const interval &b)
- interval operator∗ (const interval &a, double b)
- interval operator∗ (double b, const interval &a)
- interval operator/ (const interval &a, const interval &b)
- interval operator/ (const interval &a, double b)
- interval operator/ (double b, const interval &a)
- interval acos (const interval &x)
- interval abs (const interval &x)
- interval acosh (const interval &x)
- interval acot (const interval &x)
- interval acoth (const interval &x)
- interval asin (const interval &x)
- interval asinh (const interval &x)
- interval atan (const interval &x)
- interval atanh (const interval &x)
- interval cos (const interval &x)
- interval cosh (const interval &x)
- interval cot (const interval &x)
- interval coth (const interval &x)
- interval exp (const interval &x)
- interval exp10 (const interval &x)
- interval exp2 (const interval &x)
- interval expm1 (const interval &x)
- interval log (const interval &x)
- interval log10 (const interval &x)
- interval log1p (const interval &x)
- interval log2 (const interval &x)
- interval power (const interval &x, int n)
- interval pow (const interval &x, const interval &y)
- interval sin (const interval &x)
- interval sinh (const interval &x)
- interval sqr (const interval &x)
- interval sqrt (const interval &x)
- interval tan (const interval &x)
- interval tanh (const interval &x)
- interval imax (const interval &x, const interval &y)
- interval imin (const interval &x, const interval &y)
- interval division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b)

10.172.1 Detailed Description

Definition at line 113 of file expression.h.

10.172.2 Constructor & Destructor Documentation

10.172.2.1 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file expression.h.

10.172.2.2 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file expression.h.

10.172.2.3 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file expression.h.

10.172.2.4 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file expression.h.

10.172.2.5 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file expression.h.

10.172.2.6 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file expression.h.

10.172.2.7 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file expression.h.

10.172.2.8 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file expression.h.

10.172.2.9 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file expression.h.

10.172.2.10 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file expression.h.

10.172.2.11 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file expression.h.

10.172.2.12 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file expression.h.

10.172.2.13 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file expression.h.

10.172.2.14 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file expression.h.

10.172.2.15 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file expression.h.

10.172.2.16 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file expression.h.

10.172.2.17 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.18 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.19 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.20 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.21 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.22 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.23 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.24 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.25 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.26 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.27 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.28 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.29 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.30 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.31 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.32 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.33 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.34 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.35 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.36 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.37 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.38 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.39 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.40 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.41 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.42 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.43 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.44 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.45 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.46 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.47 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.48 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.49 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.50 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.51 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.52 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.53 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.54 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.55 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.56 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.57 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.58 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.59 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.60 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.61 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.62 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.63 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.64 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.65 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.66 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.67 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.68 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.69 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.70 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.71 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.72 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.73 coco::coco::interval::interval (double *lo*, double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.74 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.75 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.76 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.77 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.78 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.79 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.80 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.81 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.82 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.83 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.84 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.85 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.86 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.87 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.88 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.2.89 coco::coco::interval::interval (double *lo,* double *up*) [inline]

Definition at line 135 of file search_graph.cc.

10.172.2.90 coco::coco::interval::interval (double *d* = 0) [inline]

Definition at line 136 of file search_graph.cc.

10.172.2.91 coco::coco::interval::interval (int *d*) [inline]

Definition at line 137 of file search_graph.cc.

10.172.2.92 coco::coco::interval::interval (unsigned *d*) [inline]

Definition at line 138 of file search_graph.cc.

10.172.2.93 coco::coco::interval::interval (long *d*) [inline]

Definition at line 139 of file search_graph.cc.

10.172.2.94 coco::coco::interval::interval (unsigned long *d*) [inline]

Definition at line 140 of file search_graph.cc.

10.172.2.95 coco::coco::interval::interval (const interval & *x*) [inline]

Definition at line 141 of file search_graph.cc.

10.172.2.96 coco::coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 142 of file search_graph.cc.

10.172.3 Member Function Documentation

10.172.3.1 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file expression.h.

10.172.3.2 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file expression.h.

10.172.3.3 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.4 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.5 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.6 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.7 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.8 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.9 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.10 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.11 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.12 __Base coco::coco::interval::base ( ) const [inline]

Definition at line 153 of file search_graph.cc.

10.172.3.13 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.14 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.15 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.16 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file expression.h.

10.172.3.17 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.18 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.19 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.20 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.21 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file expression.h.

10.172.3.22 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.23 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.24 bool coco::coco::interval::contains (const interval & *x*) const [inline]

Definition at line 235 of file search_graph.cc.

10.172.3.25 double coco::coco::interval::diam ( ) const

10.172.3.26 double coco::coco::interval::diam ( ) const

10.172.3.27 double coco::coco::interval::diam ( ) const

10.172.3.28 double coco::coco::interval::diam ( ) const

10.172.3.29 double coco::coco::interval::diam ( ) const

10.172.3.30 double coco::coco::interval::diam ( ) const

10.172.3.31 double coco::coco::interval::diam ( ) const

10.172.3.32 double coco::coco::interval::diam ( ) const

10.172.3.33 double coco::coco::interval::diam ( ) const

10.172.3.34 double coco::coco::interval::diam ( ) const

10.172.3.35 double coco::coco::interval::diam ( ) const

10.172.3.36 double coco::coco::interval::diam ( ) const

10.172.3.37 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file expression.h.

10.172.3.38 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.39 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.40 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.41 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.42 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.43 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.44 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file expression.h.

10.172.3.45 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.46 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.47 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.48 bool coco::coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 174 of file search_graph.cc.

10.172.3.49 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.50 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.51 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.52 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.53 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.54 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.55 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.56 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.57 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.58 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.59 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.60 double coco::coco::interval::dist (const interval & *x*) const

Returns an upper bound for the Hausdorff distance of this interval and the interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

10.172.3.61 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file expression.h.

10.172.3.62 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.63 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.64 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.65 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file expression.h.

10.172.3.66 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.67 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.68 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.69 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.70 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.71 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.72 static interval coco::coco::interval::EMPTY ( ) [inline, static]

Definition at line 154 of file search_graph.cc.

10.172.3.73 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.74 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.75 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.76 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file expression.h.

10.172.3.77 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.78 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.79 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.80 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.81 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file expression.h.

10.172.3.82 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.83 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.84 bool coco::coco::interval::empty ( ) const [inline]

Definition at line 157 of file search_graph.cc.

10.172.3.85 void coco::coco::interval::get_bounds (double & *a*, double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.86 void coco::coco::interval::get_bounds (double & *a*, double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.87 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.88 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.89 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.90 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.91 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file expression.h.

10.172.3.92 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.93 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file expression.h.

10.172.3.94 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.95 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.96 void coco::coco::interval::get_bounds (double & *a,* double & *b*) const [inline]

Definition at line 244 of file search_graph.cc.

10.172.3.97 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.98 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.99 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.100 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.101 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file expression.h.

10.172.3.102 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.103 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.104 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file expression.h.

10.172.3.105 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.106 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.107 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.108 interval coco::coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 214 of file search_graph.cc.

10.172.3.109 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.110 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.111 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.112 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.113 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.114 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.115 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file expression.h.

10.172.3.116 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.117 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file expression.h.

10.172.3.118 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.119 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.120 interval coco::coco::interval::hulldiff (const interval & *x*) const [inline]

`y.hulldiff(x)` returns the hull of the set difference of `y` and `x`.

Definition at line 221 of file search_graph.cc.

10.172.3.121 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.122 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.123 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.124 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.125 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.126 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file expression.h.

10.172.3.127 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.128 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file expression.h.

10.172.3.129 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.130 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.131 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.132 interval& coco::coco::interval::hulldiffwith (const interval & *x*) [inline]

Definition at line 193 of file search_graph.cc.

10.172.3.133 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.134 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.135 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.136 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.137 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file expression.h.

10.172.3.138 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.139 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file expression.h.

10.172.3.140 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.141 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.142 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.143 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.144 interval& coco::coco::interval::hullwith (const interval & *x*) [inline]

Definition at line 187 of file search_graph.cc.

10.172.3.145 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.146 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.147 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.148 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.149 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.150 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.151 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.152 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.153 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.154 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.155 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.156 interval& coco::coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and x.

10.172.3.157 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.158 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.159 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.160 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.161 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.162 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.163 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.164 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.165 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.166 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.167 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.168 interval& coco::coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and x.

10.172.3.169 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.170 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.171 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file expression.h.

10.172.3.172 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.173 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.174 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.175 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.176 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file expression.h.

10.172.3.177 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.178 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.179 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.180 double coco::coco::interval::inf ( ) const [inline]

Definition at line 151 of file search_graph.cc.

10.172.3.181 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.182 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.183 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.184 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.185 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.186 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file expression.h.

10.172.3.187 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.188 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file expression.h.

10.172.3.189 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.190 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.191 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.192 interval coco::coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 208 of file search_graph.cc.

10.172.3.193 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.194 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.195 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.196 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.197 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.198 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.199 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.200 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.201 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.202 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.203 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.204 interval& coco::coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.172.3.205 interval& coco::coco::interval::intersect_invcos_wc (const interval & *__i*, double *__l*, double *__d*)

10.172.3.206 interval& coco::coco::interval::intersect_invcos_wc (const interval & *__i*, double *__l*, double *__d*)

10.172.3.207 interval& coco::coco::interval::intersect_invcos_wc (const interval & *__i*, double *__l*, double *__d*)

10.172.3.208 interval& coco::coco::interval::intersect_invcos_wc (const interval & *__i*, double *__l*, double *__d*)

10.172.3.209 interval& coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.172.3.210 interval& coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.172.3.211 interval& coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.172.3.212 interval& coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.172.3.213 interval& coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.172.3.214 interval& coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.172.3.215 interval& coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.172.3.216 interval& coco::coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.172.3.217 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.218 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.219 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.220 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.221 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.222 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.223 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.224 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.225 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.226 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.227 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.228 interval& coco::coco::interval::intersect_invcosh_wc (const interval & __i, double __l, double __d)

10.172.3.229 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.230 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.231 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.232 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.233 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.234 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.235 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.236 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.237 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.238 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.239 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.240 interval& coco::coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.172.3.241 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.242 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.243 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.244 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.245 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.246 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.247 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.248 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.249 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.250 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.251 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.252 interval& coco::coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.172.3.253 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.254 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.255 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.256 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.257 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.258 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.259 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.260 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.261 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.262 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.263 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.264 interval& coco::coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.172.3.265 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.266 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.267 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.268 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.269 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.270 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.271 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.272 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.273 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.274 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.275 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.276 interval& coco::coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.172.3.277 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.278 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.279 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.280 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.281 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.282 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.283 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.284 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.285 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.286 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.287 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.288 interval& coco::coco::interval::intersect_power (const interval & __i, int __n)

10.172.3.289 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.290 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.291 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.292 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file expression.h.

10.172.3.293 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.294 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.295 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.296 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file expression.h.

10.172.3.297 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.298 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.299 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.300 interval& coco::coco::interval::intersectwith (const interval & *x*) [inline]

Definition at line 180 of file search_graph.cc.

10.172.3.301 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.302 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.303 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.304 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.305 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.306 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.307 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.308 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.309 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.310 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.311 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.312 interval& coco::coco::interval::ipow (int *n*)

The method changes this interval to the `n-th` power.

10.172.3.313 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.314 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.315 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.316 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.317 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file expression.h.

10.172.3.318 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.319 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.320 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.321 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file expression.h.

10.172.3.322 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.323 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.324 bool coco::coco::interval::is_bounded ( ) const [inline]

Definition at line 163 of file search_graph.cc.

10.172.3.325 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.326 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.327 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.328 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file expression.h.

10.172.3.329 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.330 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.331 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.332 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.333 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.334 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.335 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file search_graph.cc.

10.172.3.336 bool coco::coco::interval::is_empty ( ) const [inline]

Definition at line 158 of file expression.h.

10.172.3.337 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.338 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.339 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.340 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.341 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file expression.h.

10.172.3.342 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.343 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file expression.h.

10.172.3.344 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.345 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.346 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.347 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.348 bool coco::coco::interval::is_entire ( ) const [inline]

Definition at line 162 of file search_graph.cc.

10.172.3.349 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file expression.h.

10.172.3.350 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.351 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.352 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.353 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file expression.h.

10.172.3.354 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.355 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.356 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.357 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.358 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.359 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.360 bool coco::coco::interval::is_thin ( ) const [inline]

Definition at line 159 of file search_graph.cc.

10.172.3.361 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.362 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.363 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.364 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.365 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file expression.h.

10.172.3.366 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.367 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.368 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.369 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.370 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.371 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file search_graph.cc.

10.172.3.372 bool coco::coco::interval::is_unbounded_above ( ) const [inline]

Definition at line 161 of file expression.h.

10.172.3.373 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.374 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.375 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.376 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.377 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file expression.h.

10.172.3.378 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.379 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.380 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.381 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.382 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.383 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file search_graph.cc.

10.172.3.384 bool coco::coco::interval::is_unbounded_below ( ) const [inline]

Definition at line 160 of file expression.h.

10.172.3.385 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.386 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.387 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.388 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.389 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.390 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.391 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.392 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.393 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.394 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.395 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.396 double coco::coco::interval::mag ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.397 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.398 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.399 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.400 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.401 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.402 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.403 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.404 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.405 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.406 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.407 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.408 double coco::coco::interval::mid ( ) const

Returns the midpoint of this interval, i.e.

a.mid() == (a.inf()+a.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

10.172.3.409 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.410 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.411 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.412 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.413 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.414 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.415 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.416 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.417 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.418 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.419 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.420 double coco::coco::interval::mig ( ) const

Returns the mignitude of this interval, i.e.

a.mig() == min{abs(y) : y in a }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

10.172.3.421 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.422 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.423 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.424 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.425 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.426 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.427 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.428 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.429 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.430 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.431 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.432 interval& coco::coco::interval::operator*= (const interval & *a*)

10.172.3.433 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.434 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.435 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.436 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.437 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.438 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.439 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.440 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.441 interval& coco::coco::interval::operator*= (double *a*)

10.172.3.442 interval& coco::coco::interval::operator∗= (double *a*)

10.172.3.443 interval& coco::coco::interval::operator∗= (double *a*)

10.172.3.444 interval& coco::coco::interval::operator∗= (double *a*)

10.172.3.445 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.446 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.447 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.448 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.449 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.450 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.451 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.452 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.453 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.454 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.455 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.456 interval& coco::coco::interval::operator+= (const interval & *a*)

10.172.3.457 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.458 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.459 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.460 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.461 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.462 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.463 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.464 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.465 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.466 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.467 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.468 interval& coco::coco::interval::operator+= (double *a*)

10.172.3.469 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.470 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file expression.h.

10.172.3.471 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.472 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.473 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.474 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.475 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.476 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.477 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file expression.h.

10.172.3.478 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.479 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.480 interval coco::coco::interval::operator- ( ) const [inline]

Definition at line 258 of file search_graph.cc.

10.172.3.481 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.482 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.483 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.484 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.485 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.486 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.487 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.488 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.489 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.490 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.491 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.492 interval& coco::coco::interval::operator-= (const interval & *a*)

10.172.3.493 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.494 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.495 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.496 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.497 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.498 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.499 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.500 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.501 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.502 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.503 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.504 interval& coco::coco::interval::operator-= (double *a*)

10.172.3.505 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.506 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.507 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.508 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.509 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.510 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.511 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.512 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.513 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.514 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.515 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.516 interval& coco::coco::interval::operator/= (const interval & *a*)

10.172.3.517 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.518 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.519 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.520 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.521 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.522 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.523 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.524 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.525 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.526 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.527 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.528 interval& coco::coco::interval::operator/= (double *a*)

10.172.3.529 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.530 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.531 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.532 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.533 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.534 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.535 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.536 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file expression.h.

10.172.3.537 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.538 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.539 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file search_graph.cc.

10.172.3.540 interval& coco::coco::interval::operator= (const interval & *x*) [inline]

Definition at line 249 of file expression.h.

10.172.3.541 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.542 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file expression.h.

10.172.3.543 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.544 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.545 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.546 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.547 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.548 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.549 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.550 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file expression.h.

10.172.3.551 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.552 interval& coco::coco::interval::operator= (double *d*) [inline]

Definition at line 250 of file search_graph.cc.

10.172.3.553 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.554 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file expression.h.

10.172.3.555 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.556 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.557 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.558 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.559 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.560 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.561 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file expression.h.

10.172.3.562 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.563 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.564 interval& coco::coco::interval::operator= (int *d*) [inline]

Definition at line 251 of file search_graph.cc.

10.172.3.565 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.566 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.567 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.568 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.569 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file expression.h.

10.172.3.570 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.571 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.572 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file expression.h.

10.172.3.573 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.574 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.575 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.576 interval& coco::coco::interval::operator= (unsigned *d*) [inline]

Definition at line 252 of file search_graph.cc.

10.172.3.577 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.578 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file expression.h.

10.172.3.579 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.580 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.581 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.582 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.583 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.584 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.585 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file expression.h.

10.172.3.586 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.587 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.588 interval& coco::coco::interval::operator= (long *d*) [inline]

Definition at line 253 of file search_graph.cc.

10.172.3.589 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file expression.h.

10.172.3.590 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.591 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.592 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.593 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.594 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.595 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.596 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.597 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file expression.h.

10.172.3.598 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.599 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.600 interval& coco::coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 254 of file search_graph.cc.

10.172.3.601 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.602 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.603 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.604 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.605 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.606 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.607 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.608 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.609 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.610 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.611 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.612 static int const& coco::coco::interval::precision ( ) [static]

This method returns the output precision that is used by the output operator <<.

10.172.3.613 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.614 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.615 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.616 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.617 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.618 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.619 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.620 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.621 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.622 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.623 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.624 static int coco::coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.172.3.625 double coco::coco::interval::project (double *__d*) const

10.172.3.626 double coco::interval::project (double *__d*) const [inline]

Definition at line 830 of file interval_boost.h.

10.172.3.627 double coco::coco::interval::project (double *__d*) const

10.172.3.628 double coco::coco::interval::project (double *__d*) const

10.172.3.629 double coco::coco::interval::project (double *__d*) const

10.172.3.630 double coco::coco::interval::project (double *__d*) const

10.172.3.631 double coco::coco::interval::project (double *__d*) const

10.172.3.632 double coco::coco::interval::project (double *__d*) const

10.172.3.633 double coco::coco::interval::project (double *__d*) const

10.172.3.634 double coco::coco::interval::project (double *__d*) const

10.172.3.635 double coco::coco::interval::project (double *__d*) const

10.172.3.636 double coco::coco::interval::project (double *__d*) const

10.172.3.637 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.638 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.639 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.640 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.641 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file expression.h.

10.172.3.642 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.643 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.644 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.645 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.646 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.647 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file expression.h.

10.172.3.648 bool coco::coco::interval::proper_subset (const interval & *x*) const [inline]

Definition at line 170 of file search_graph.cc.

10.172.3.649 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file expression.h.

10.172.3.650 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.651 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.652 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.653 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file expression.h.

10.172.3.654 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.655 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.656 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.657 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.658 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.659 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.660 bool coco::coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 172 of file search_graph.cc.

10.172.3.661 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.662 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.663 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.664 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.665 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.666 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.667 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.668 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.669 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.670 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.671 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.672 double coco::coco::interval::rad ( ) const

Returns an upper bound for the radius of this interval, i.e.

a.rad() = (a.sup() - a.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite interval

10.172.3.673 double coco::coco::interval::rel_width ( ) const

10.172.3.674 double coco::coco::interval::rel_width ( ) const

10.172.3.675 double coco::coco::interval::rel_width ( ) const

10.172.3.676 double coco::coco::interval::rel_width ( ) const

10.172.3.677 double coco::coco::interval::rel_width ( ) const

10.172.3.678 double coco::coco::interval::rel_width ( ) const

10.172.3.679 double coco::coco::interval::rel_width ( ) const

10.172.3.680 double coco::coco::interval::rel_width ( ) const

10.172.3.681 double coco::coco::interval::rel_width ( ) const

10.172.3.682 double coco::coco::interval::rel_width ( ) const

10.172.3.683 double coco::coco::interval::rel_width ( ) const

10.172.3.684 double coco::coco::interval::rel_width ( ) const

10.172.3.685 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.686 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.687 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.688 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.689 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.690 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.691 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.692 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.693 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.694 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.695 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.696 double coco::coco::interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this interval, i.e.

a.relDiam == a.diam() if a.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite interval

10.172.3.697 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.698 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.699 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.700 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.701 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.702 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.703 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.704 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.705 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.706 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.707 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.708 interval& coco::coco::interval::round_to_integer ( )

This method rounds the interval inward to integer borders.

10.172.3.709 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.710 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.711 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.712 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.713 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.714 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.715 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.716 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.717 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.718 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.719 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.720 double coco::coco::interval::safeguarded_mid ( ) const

Returns the magnitude of this interval, i.e.

a.mag() == max{abs(y) : y in a }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite interval

10.172.3.721 void coco::coco::interval::set (double *lo*, double *up*) [inline]

Definition at line 145 of file expression.h.

10.172.3.722 void coco::coco::interval::set (double *lo*, double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.723 void coco::coco::interval::set (double *lo*, double *up*) [inline]

Definition at line 145 of file expression.h.

10.172.3.724 void coco::coco::interval::set (double *lo*, double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.725 void coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.726 void coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.727 void coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.728 void coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.729 void coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.730 void coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.731 void coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.732 void coco::coco::interval::set (double *lo,* double *up*) [inline]

Definition at line 145 of file search_graph.cc.

10.172.3.733 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file expression.h.

10.172.3.734 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.735 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.736 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.737 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file expression.h.

10.172.3.738 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.739 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.740 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.741 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.742 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.743 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.744 void coco::coco::interval::set_bounds (double *a,* double *b*) [inline]

Definition at line 245 of file search_graph.cc.

10.172.3.745 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.746 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.747 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.748 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.749 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.750 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.751 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.752 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.753 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.754 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file expression.h.

10.172.3.755 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file search_graph.cc.

10.172.3.756 void coco::coco::interval::set_lb (double *d*) [inline]

Definition at line 246 of file expression.h.

10.172.3.757 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.758 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.759 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.760 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file expression.h.

10.172.3.761 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.762 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.763 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.764 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.765 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.766 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.767 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file expression.h.

10.172.3.768 void coco::coco::interval::set_ub (double *d*) [inline]

Definition at line 247 of file search_graph.cc.

10.172.3.769 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.770 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.771 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.772 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.773 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file expression.h.

10.172.3.774 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.775 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.776 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.777 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.778 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.779 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file search_graph.cc.

10.172.3.780 void coco::coco::interval::setpair (double & *lo,* double & *up*) const [inline]

Definition at line 243 of file expression.h.

10.172.3.781 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.782 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.783 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.784 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.785 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file expression.h.

10.172.3.786 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.787 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.788 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.789 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.790 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.791 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file search_graph.cc.

10.172.3.792 bool coco::coco::interval::subset (const interval & *x*) const [inline]

Definition at line 166 of file expression.h.

10.172.3.793 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.794 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.795 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file expression.h.

10.172.3.796 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.797 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.798 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.799 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.800 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.801 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.802 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.803 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file expression.h.

10.172.3.804 double coco::coco::interval::sup ( ) const [inline]

Definition at line 152 of file search_graph.cc.

10.172.3.805 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.806 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.807 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file expression.h.

10.172.3.808 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.809 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.810 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.811 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.812 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.813 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.814 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file expression.h.

10.172.3.815 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.816 bool coco::coco::interval::superset (const interval & *x*) const [inline]

Definition at line 168 of file search_graph.cc.

10.172.3.817 double coco::coco::interval::width ( ) const

10.172.3.818 double coco::coco::interval::width ( ) const

10.172.3.819 double coco::coco::interval::width ( ) const

10.172.3.820 double coco::coco::interval::width ( ) const

10.172.3.821 double coco::coco::interval::width ( ) const

10.172.3.822 double coco::coco::interval::width ( ) const

10.172.3.823 double coco::coco::interval::width ( ) const

10.172.3.824 double coco::coco::interval::width ( ) const

10.172.3.825 double coco::coco::interval::width ( ) const

10.172.3.826 double coco::coco::interval::width ( ) const

10.172.3.827 double coco::coco::interval::width ( ) const

10.172.3.828 double coco::coco::interval::width ( ) const

10.172.4 Friends And Related Function Documentation

10.172.4.1 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.2 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.3 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.4 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.5 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.6 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.7 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.8 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.9 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.10 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.11 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.12 interval abs (const interval & *x*) [friend]

Returns an interval enclosure of the absolute value of the interval x.

10.172.4.13 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.14 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.15 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.16 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.17 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.18 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.19 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.20 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.21 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.22 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.23 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.24 interval acos (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cosine of the interval x.

10.172.4.25 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.26 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.27 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.28 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.29 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.30 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.31 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.32 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.33 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.34 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.35 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.36 interval acosh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cosine of the interval x.

10.172.4.37 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.38 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.39 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.40 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.41 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.42 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.43 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.44 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.45 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.46 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.47 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.48 interval acot (const interval & *x*) [friend]

Returns an interval enclosure of the inverse cotangent of the interval x.

10.172.4.49 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.50 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.51 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.52 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.53 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.54 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.55 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.56 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.57 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.58 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.59 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.60 interval acoth (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic cotangent of the interval x.

10.172.4.61 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.62 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.63 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.64 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.65 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.66 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.67 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.68 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.69 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.70 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.71 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.72 interval asin (const interval & *x*) [friend]

Returns an interval enclosure of the inverse sine of the interval x.

10.172.4.73 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.74 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.75 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.76 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.77 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.78 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.79 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.80 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.81 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.82 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.83 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.84 interval asinh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic sine of the interval x.

10.172.4.85 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.86 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.87 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.88 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.89 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.90 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.91 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.92 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.93 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.94 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.95 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.96 interval atan (const interval & *x*) [friend]

Returns an interval enclosure of the inverse tangent of the interval x.

10.172.4.97 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.98 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.99 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.100 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.101 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.102 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.103 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.104 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.105 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.106 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.107 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.108 interval atanh (const interval & *x*) [friend]

Returns an interval enclosure of the inverse hyperbolic tangent of the interval x.

10.172.4.109 interval cancel (const interval & *a,* const interval & *b*) [friend]

10.172.4.110 interval cancel (const interval & *a,* const interval & *b*) [friend]

10.172.4.111 interval cancel (const interval & *a,* const interval & *b*) [friend]

10.172.4.112 interval cancel (const interval & *a,* const interval & *b*) [friend]

10.172.4.113 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.172.4.114 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.172.4.115 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.172.4.116 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.172.4.117 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.172.4.118 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.172.4.119 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.172.4.120 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.172.4.121 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.122 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.123 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.124 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.125 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.126 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.127 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.128 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.129 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.130 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.131 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.132 interval cos (const interval & *x*) [friend]

Returns an interval enclosure of the cosine of the interval x.

10.172.4.133 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.134 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.135 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.136 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.137 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.138 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.139 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.140 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.141 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.142 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.143 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.144 interval cosh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cosine of the interval x.

10.172.4.145 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.146 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.147 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.148 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.149 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.150 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.151 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.152 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.153 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.154 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.155 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.156 interval cot (const interval & *x*) [friend]

Returns an interval enclosure of the cotangent of the interval x.

10.172.4.157 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.158 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.159 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.160 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.161 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.162 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.163 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.164 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.165 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.166 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.167 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.168 interval coth (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic cotangent of the interval x.

10.172.4.169 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.170 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.171 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.172 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.173 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval,

division_part1 returns this interval and sets the boolean reference **to** false. However, if the result needs two intervals, division_part1 returns the negative part and sets the boolean reference to true; a call to division_part2 is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be true, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.174 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, division_part1 returns this interval and sets the boolean reference **to** false. However, if the result needs two intervals, division_part1 returns the negative part and sets the boolean reference to true; a call to division_part2 is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be true, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.175 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, division_part1 returns this interval and sets the boolean reference **to** false. However, if the result needs two intervals, division_part1 returns the negative part and sets the boolean reference to true; a call to division_part2 is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be true, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.176 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, division_part1 returns this interval and sets the boolean reference **to** false. However, if the result needs two intervals, division_part1 returns the negative part and sets the boolean reference to true; a call to division_part2 is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be true, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.177 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, division_part1 returns this interval and sets the boolean reference **to** false. However, if the result needs two intervals, division_part1 returns the negative part and sets the boolean reference to true; a call to division_part2 is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be true, and the behavior of the function is then undetermined if the division does not produce a second

interval.

10.172.4.178 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.179 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.180 interval division_part1 (const interval & *x*, const interval & *y*, bool & *b*) [friend]

The division_part? functions return the result of x/y in the following sense, even if the smallest set containing the result is a union of two intervals. For example, the narrowest closed set containg $[2,3]/[-2,1]$ is not $]-\infty,\infty[$ but $]-\infty,-1]\cup[2,\infty[$. When the result of the division is representable by only one interval, `division_part1` returns this interval and sets the boolean reference **to** `false`. However, if the result needs two intervals, `division_part1` returns the negative part and sets the boolean reference to `true`; a call to `division_part2` is now needed to get the positive part. This second function can take the boolean returned by the first function as last argument. If this bool is not given, its value is assumed to be `true`, and the behavior of the function is then undetermined if the division does not produce a second interval.

10.172.4.181 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.182 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.183 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.184 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.185 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.186 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.187 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.188 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.189 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.190 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.191 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.192 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.172.4.193 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.194 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.195 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.196 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.197 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.198 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.199 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.200 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.201 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.202 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.203 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.204 interval exp (const interval & *x*) [friend]

Returns an interval enclosure of the exponential of the interval x.

10.172.4.205 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.206 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.207 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.208 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.209 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.210 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.211 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.212 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.213 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.214 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.215 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.216 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.172.4.217 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.218 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.219 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.220 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.221 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.222 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.223 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.224 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.225 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.226 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.227 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.228 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.172.4.229 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.230 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.231 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.232 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.233 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.234 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.235 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.236 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.237 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.238 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.239 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.240 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.172.4.241 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.242 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.243 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.244 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.245 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.246 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.247 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.248 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.249 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.250 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.251 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.252 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.172.4.253 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.254 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.255 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.256 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.257 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.258 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.259 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.260 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.261 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.262 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.263 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.264 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.172.4.265 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.266 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.267 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.268 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.269 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.270 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.271 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.272 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.273 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.274 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.275 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.276 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.172.4.277 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.278 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.279 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.280 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.281 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.282 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.283 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.284 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.285 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.286 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.287 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.288 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.172.4.289 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.290 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.291 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.292 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.293 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.294 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.295 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.296 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.297 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.298 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.299 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.300 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.172.4.301 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.302 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.303 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.304 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.305 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.306 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.307 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.308 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.309 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.310 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.311 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.312 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.172.4.313 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.314 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.315 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.316 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.317 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.318 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.319 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.320 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.321 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.322 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.323 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.324 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.172.4.325 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.326 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.327 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.328 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.329 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.330 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.331 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.332 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.333 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.334 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.335 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.336 interval operator∗ (const interval & *a*, double *b*) [friend]

10.172.4.337 interval operator∗ (double *b*, const interval & *a*) [friend]

10.172.4.338 interval operator∗ (double *b*, const interval & *a*) [friend]

10.172.4.339 interval operator∗ (double *b*, const interval & *a*) [friend]

10.172.4.340 interval operator∗ (double *b*, const interval & *a*) [friend]

10.172.4.341 interval operator∗ (double *b*, const interval & *a*) [friend]

10.172.4.342 interval operator∗ (double *b*, const interval & *a*) [friend]

10.172.4.343 interval operator∗ (double *b,* const interval & *a*) [friend]

10.172.4.344 interval operator∗ (double *b,* const interval & *a*) [friend]

10.172.4.345 interval operator∗ (double *b,* const interval & *a*) [friend]

10.172.4.346 interval operator∗ (double *b,* const interval & *a*) [friend]

10.172.4.347 interval operator∗ (double *b,* const interval & *a*) [friend]

10.172.4.348 interval operator∗ (double *b,* const interval & *a*) [friend]

10.172.4.349 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.350 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.351 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.352 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.353 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.354 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.355 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.356 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.357 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.358 interval operator+ (const interval & *a,* const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.359 interval operator+ (const interval & *a*, const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.360 interval operator+ (const interval & *a*, const interval & *b*) [friend]

This operator computes a+b as interval.

10.172.4.361 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.362 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.363 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.364 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.365 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.366 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.367 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.368 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.369 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.370 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.371 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.372 interval operator+ (const interval & *a*, double *b*) [friend]

10.172.4.373 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.374 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.375 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.376 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.377 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.378 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.379 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.380 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.381 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.382 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.383 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.384 interval operator+ (double *b*, const interval & *a*) [friend]

10.172.4.385 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.386 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.387 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.388 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.389 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.390 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.391 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.392 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.393 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.394 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.395 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.396 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.172.4.397 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.398 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.399 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.400 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.401 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.402 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.403 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.404 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.405 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.406 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.407 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.408 interval operator- (const interval & *a*, double *b*) [friend]

10.172.4.409 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.410 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.411 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.412 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.413 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.414 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.415 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.416 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.417 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.418 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.419 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.420 interval operator- (double *b*, const interval & *a*) [friend]

10.172.4.421 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.422 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.423 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.424 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.425 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.426 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.427 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.428 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.429 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.430 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.431 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.432 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.172.4.433 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.434 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.435 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.436 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.437 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.438 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.439 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.440 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.441 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.442 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.443 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.444 interval operator/ (const interval & *a*, double *b*) [friend]

10.172.4.445 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.446 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.447 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.448 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.449 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.450 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.451 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.452 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.453 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.454 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.455 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.456 interval operator/ (double *b*, const interval & *a*) [friend]

10.172.4.457 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.172.4.458 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.172.4.459 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.172.4.460 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.172.4.461 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.172.4.462 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.172.4.463 std::ostream& operator<< (std::ostream & *s,* const interval & *a*) [friend]

10.172.4.464 std::ostream& operator<< (std::ostream & *s,* const interval & *a*) [friend]

10.172.4.465 std::ostream& operator<< (std::ostream & *s,* const interval & *a*) [friend]

10.172.4.466 std::ostream& operator<< (std::ostream & *s,* const interval & *a*) [friend]

10.172.4.467 std::ostream& operator<< (std::ostream & *s,* const interval & *a*) [friend]

10.172.4.468 std::ostream& operator<< (std::ostream & *s,* const interval & *a*) [friend]

10.172.4.469 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.470 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.471 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.472 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.473 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.474 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.475 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.476 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.477 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.478 interval pow (const interval & *x,* const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.479 interval pow (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.480 interval pow (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.172.4.481 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.482 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.483 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.484 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.485 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.486 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.487 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.488 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.489 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.490 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.491 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.492 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.172.4.493 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.494 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.495 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.496 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.497 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.498 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.499 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.500 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.501 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.502 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.503 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.504 interval sin (const interval & *x*) [friend]

Returns an interval enclosure of the sine of the interval x.

10.172.4.505 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.506 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.507 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.508 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.509 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.510 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.511 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.512 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.513 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.514 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.515 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.516 interval sinh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic sine of the interval x.

10.172.4.517 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.518 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.519 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.520 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.521 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.522 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.523 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.524 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.525 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.526 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.527 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.528 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.172.4.529 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.530 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.531 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.532 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.533 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.534 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.535 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.536 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.537 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.538 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.539 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.540 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.172.4.541 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.542 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.543 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.544 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.545 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.546 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.547 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.548 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.549 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.550 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.551 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.552 interval tan (const interval & *x*) [friend]

Returns an interval enclosure of the tangent of the interval x.

10.172.4.553 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.554 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.555 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.556 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.557 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.558 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.559 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.560 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.561 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.562 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.563 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

10.172.4.564 interval tanh (const interval & *x*) [friend]

Returns an interval enclosure of the hyperbolic tangent of the interval x.

The documentation for this class was generated from the following files:

- interval_boost.h
- interval_filib.h
- interval_profil.h

10.173 coco::interval_eval Class Reference

Forward function range evaluation.

`#include <int_evaluator.h>`

Inheritance diagram for coco::interval_eval:

Collaboration diagram for coco::interval_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- interval_eval (const std::vector< interval > &__x, const variable_indicator &__v, const model &_-_m, std::vector< interval > *__c, bool do_i=true)
- interval_eval (const interval_eval &__v)
- ~interval_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_box (const std::vector< interval > &__x, const variable_indicator &__v)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const interval &__rval)
- int update (const expression_node &__data, const interval &__rval)
- interval calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.173.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural interval extension.

10.173.2 Member Typedef Documentation

10.173.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.173.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.173.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.173.3 Constructor & Destructor Documentation

10.173.3.1 coco::interval_eval::interval_eval (const std::vector< interval > & __x, const variable_indicator & __v, const model & __m, std::vector< interval > * __c, bool do_i = true) [inline]

Constructor: ___x is the box over which the function range shall be evaluated, ___v is a variable indicator specifying the variables that have changed value since the last evaluation, ___m specifies the DAG, and ___c the cache (may be NULL). The boolean do_i determines whether the natural interval extension will use known node ranges for reducing the ranges.

Definition at line 141 of file int_evaluator.h.

10.173.3.2 coco::interval_eval::interval_eval (const interval_eval & __v) [inline]

Standard Copy Constructor

Definition at line 155 of file int_evaluator.h.

10.173.3.3 coco::interval_eval::~interval_eval ( ) [inline]

Standard Destructor

Definition at line 158 of file int_evaluator.h.

10.173.4 Member Function Documentation

10.173.4.1 void coco::interval_eval::calculate (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 238 of file int_evaluator.h.

10.173.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.173.4.3 interval coco::interval_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 751 of file int_evaluator.h.

10.173.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.173.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.173.4.6 void coco::interval_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 175 of file int_evaluator.h.

10.173.4.7 int coco::interval_eval::initialize (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 177 of file int_evaluator.h.

10.173.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.173.4.9 bool coco::interval_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the range for this node is already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 86 of file int_evaluator.h.

10.173.4.10 void coco::interval_eval::new_box (const std::vector< interval > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation box to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 167 of file int_evaluator.h.

10.173.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.173.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.173.4.13 void coco::interval_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 249 of file int_evaluator.h.

10.173.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.173.4.15 expression_const_walker coco::interval_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 161 of file int_evaluator.h.

10.173.4.16 int coco::interval_eval::update (const interval & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 264 of file int_evaluator.h.

10.173.4.17 int coco::interval_eval::update (const expression_node & __data, const interval & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 270 of file int_evaluator.h.

10.173.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.173.4.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.173.4.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.173.4.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.173.4.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.173.4.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

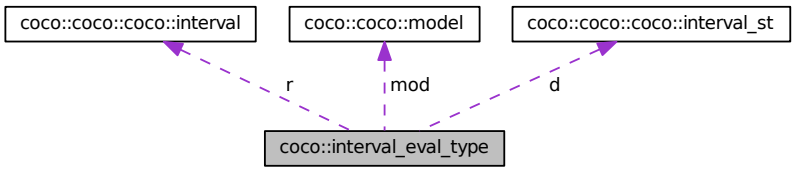
- int_evaluator.h

10.174 coco::interval_eval_type Struct Reference

Visitor data for interval_eval.

`#include <int_evaluator.h>`

Collaboration diagram for coco::interval_eval_type:



Public Attributes

- const std::vector< interval > * x
- std::vector< interval > * cache
- const model * mod
- union {
  void * p
  interval_st d
  int info
  } u
- interval r
- unsigned int n
- bool do_intersect

10.174.1 Detailed Description

This class is the visitor data type for the interval_eval evaluator.

10.174.2 Member Data Documentation

10.174.2.1 std::vector<interval>* coco::interval_eval_type::cache

the function range cache

Definition at line 59 of file int_evaluator.h.

10.174.2.2 interval_st coco::interval_eval_type::d

Definition at line 61 of file int_evaluator.h.

10.174.2.3 bool coco::interval_eval_type::do_intersect

compute an intersection with the node range?

Definition at line 65 of file int_evaluator.h.

10.174.2.4 int coco::interval_eval_type::info

Definition at line 61 of file int_evaluator.h.

10.174.2.5 const model* coco::interval_eval_type::mod

the DAG

Definition at line 60 of file int_evaluator.h.

10.174.2.6 unsigned int coco::interval_eval_type::n

children counter

Definition at line 64 of file int_evaluator.h.

10.174.2.7 void* coco::interval_eval_type::p

Definition at line 61 of file int_evaluator.h.

10.174.2.8 interval coco::interval_eval_type::r

return value

Definition at line 63 of file int_evaluator.h.

10.174.2.9 union { ... } coco::interval_eval_type::u

additional data for complex nodes

10.174.2.10 const std::vector<interval>* coco::interval_eval_type::x

the box

Definition at line 58 of file int_evaluator.h.

The documentation for this struct was generated from the following file:

- int_evaluator.h

10.175 coco::interval_set Class Reference

`#include <interval_set.h>`

Public Member Functions

- interval_set (void)
- interval_set (double a)
- interval_set (double a, double b)
- interval_set (const interval &a)
- interval_set (const interval &a, const interval &b)

- interval_set (const interval_set &a)
- virtual ~interval_set ()
- interval_set & operator= (const interval_set &a)
- void setSorting (bool s)
- bool getSorting () const
- int getLength () const
- unsigned int get_maxlen () const
- void set_maxlen (unsigned int m)
- unsigned int get_fillext () const
- void set_fillext (unsigned int m)
- bool is_empty () const
- bool is_subset (const interval_set &a) const
- bool is_subset (const interval &a) const
- bool is_proper_subset (const interval_set &a) const
- bool is_proper_subset (const interval &a) const
- bool is_disjoint (const interval_set &a) const
- bool is_disjoint (const interval &a) const
- interval & operator[ ] (int pos) const
- void add (const interval &a)
- void add (const interval_set &a)
- interval remove (int pos)
- interval removeFirst ()
- void trimToSize ()
- void trimintervaloSize ()
- int find (const interval &a)
- bool contains (double a) const
- bool contains (const interval &a) const
- void conc (const interval &a)
- interval_set remove (const interval &a)
- interval minimum () const
- interval maximum () const
- interval hull () const
- double width () const
- double volume () const
- double mig () const
- double mag () const
- double safeguarded_mid () const
- int num_of_gaps () const
- interval_set & round_to_integer ()
- interval_set & intersectwith (const interval_set &a)
- interval_set intersect (const interval_set &a)
- void intersect_power (const interval_set &a, int n)
- void intersect_div (const interval_set &a, const interval_set &b)
- void intersect_invsqr_wc (const interval_set &a, double b, double c)
- void intersect_invpower_wc (const interval_set &a, double b, int n)
- void intersect_invsin_wc (const interval_set &a, double b, double c)
- void intersect_invcos_wc (const interval_set &a, double b, double c)
- void intersect_invcosh_wc (const interval_set &a, double b, double c)
- void intersect_invgauss_wc (const interval_set &a, double b, double c, double d)
- interval_set & operator+= (const interval_set &a)
- interval_set & operator-= (const interval_set &a)
- interval_set & operator*= (const interval_set &a)
- interval_set & operator/= (const interval_set &a)

10.175.1 Constructor & Destructor Documentation

10.175.1.1 coco::interval_set::interval_set (void) [inline]

Definition at line 206 of file interval_set.h.

10.175.1.2 coco::interval_set::interval_set (double *a*) [inline]

Definition at line 211 of file interval_set.h.

10.175.1.3 coco::interval_set::interval_set (double *a*, double *b*) [inline]

Definition at line 217 of file interval_set.h.

10.175.1.4 coco::interval_set::interval_set (const interval & *a*) [inline]

Definition at line 223 of file interval_set.h.

10.175.1.5 coco::interval_set::interval_set (const interval & *a*, const interval & *b*) [inline]

Definition at line 229 of file interval_set.h.

10.175.1.6 coco::interval_set::interval_set (const interval_set & *a*) [inline]

Definition at line 235 of file interval_set.h.

10.175.1.7 coco::interval_set::~interval_set ( ) [inline, virtual]

Definition at line 242 of file interval_set.h.

10.175.2 Member Function Documentation

10.175.2.1 void coco::interval_set::add (const interval & *a*)

Definition at line 134 of file interval_set.cc.

10.175.2.2 void coco::interval_set::add (const interval_set & *a*) [inline]

Definition at line 285 of file interval_set.h.

10.175.2.3 void coco::interval_set::conc (const interval & *a*) [inline]

Definition at line 322 of file interval_set.h.

10.175.2.4 bool coco::interval_set::contains (double *a*) const [inline]

Definition at line 317 of file interval_set.h.

10.175.2.5 bool coco::interval_set::contains (const interval & *a*) const [inline]

Definition at line 311 of file interval_set.h.

10.175.2.6 int coco::interval_set::find (const interval & *a*) [inline]

Definition at line 303 of file interval_set.h.

10.175.2.7 unsigned int coco::interval_set::get_fillext ( ) const [inline]

Definition at line 264 of file interval_set.h.

10.175.2.8 unsigned int coco::interval_set::get_maxlen ( ) const [inline]

Definition at line 256 of file interval_set.h.

10.175.2.9 int coco::interval_set::getLength ( ) const [inline]

Definition at line 247 of file interval_set.h.

10.175.2.10 bool coco::interval_set::getSorting ( ) const [inline]

Definition at line 252 of file interval_set.h.

10.175.2.11 interval coco::interval_set::hull ( ) const

Definition at line 389 of file interval_set.cc.

10.175.2.12 interval_set coco::interval_set::intersect (const interval_set & *a*)

Definition at line 462 of file interval_set.cc.

10.175.2.13 void coco::interval_set::intersect_div (const interval_set & *a*, const interval_set & *b*)

Definition at line 510 of file interval_set.cc.

10.175.2.14 void coco::interval_set::intersect_invcos_wc (const interval_set & *a*, double *b*, double *c*)

Definition at line 600 of file interval_set.cc.

10.175.2.15 void coco::interval_set::intersect_invcosh_wc (const interval_set & *a*, double *b*, double *c*)

Definition at line 547 of file interval_set.cc.

10.175.2.16 void coco::interval_set::intersect_invgauss_wc (const interval_set & *a*, double *b*, double *c*, double *d*)

Definition at line 616 of file interval_set.cc.

10.175.2.17 void coco::interval_set::intersect_invpower_wc (const interval_set & *a*, double *b*, int *n*)

Definition at line 568 of file interval_set.cc.

10.175.2.18 void coco::interval_set::intersect_invsin_wc (const interval_set & *a*, double *b*, double *c*)

Definition at line 584 of file interval_set.cc.

10.175.2.19 void coco::interval_set::intersect_invsqr_wc (const interval_set & *a,* double *b,* double *c*)

Definition at line 526 of file interval_set.cc.

10.175.2.20 void coco::interval_set::intersect_power (const interval_set & *a,* int *n*)

Definition at line 494 of file interval_set.cc.

10.175.2.21 interval_set & coco::interval_set::intersectwith (const interval_set & *a*)

Definition at line 478 of file interval_set.cc.

10.175.2.22 bool coco::interval_set::is_disjoint (const interval_set & *a*) const

Definition at line 115 of file interval_set.cc.

10.175.2.23 bool coco::interval_set::is_disjoint (const interval & *a*) const

Definition at line 129 of file interval_set.cc.

10.175.2.24 bool coco::interval_set::is_empty ( ) const [inline]

Definition at line 272 of file interval_set.h.

10.175.2.25 bool coco::interval_set::is_proper_subset (const interval_set & *a*) const

Definition at line 104 of file interval_set.cc.

10.175.2.26 bool coco::interval_set::is_proper_subset (const interval & *a*) const

Definition at line 110 of file interval_set.cc.

10.175.2.27 bool coco::interval_set::is_subset (const interval_set & *a*) const

Definition at line 81 of file interval_set.cc.

10.175.2.28 bool coco::interval_set::is_subset (const interval & *a*) const

Definition at line 99 of file interval_set.cc.

10.175.2.29 double coco::interval_set::mag ( ) const

Definition at line 410 of file interval_set.cc.

10.175.2.30 interval coco::interval_set::maximum ( ) const

Definition at line 372 of file interval_set.cc.

10.175.2.31 double coco::interval_set::mig ( ) const

Definition at line 402 of file interval_set.cc.

10.175.2.32 interval coco::interval_set::minimum ( ) const

Definition at line 354 of file interval_set.cc.

10.175.2.33 int coco::interval_set::num_of_gaps ( ) const [inline]

Definition at line 112 of file interval_set.h.

10.175.2.34 interval_set & coco::interval_set::operator*= (const interval_set & *a*)

Definition at line 644 of file interval_set.cc.

10.175.2.35 interval_set & coco::interval_set::operator+= (const interval_set & *a*)

Definition at line 632 of file interval_set.cc.

10.175.2.36 interval_set & coco::interval_set::operator-= (const interval_set & *a*)

Definition at line 638 of file interval_set.cc.

10.175.2.37 interval_set & coco::interval_set::operator/= (const interval_set & *a*)

Definition at line 650 of file interval_set.cc.

10.175.2.38 interval_set & coco::interval_set::operator= (const interval_set & *a*)

Definition at line 59 of file interval_set.cc.

10.175.2.39 interval & coco::interval_set::operator[] (int *pos*) const [inline]

Definition at line 276 of file interval_set.h.

10.175.2.40 interval coco::interval_set::remove (int *pos*)

Definition at line 292 of file interval_set.cc.

10.175.2.41 interval_set coco::interval_set::remove (const interval & *a*)

Definition at line 317 of file interval_set.cc.

10.175.2.42 interval coco::interval_set::removeFirst ( ) [inline]

Definition at line 291 of file interval_set.h.

10.175.2.43 interval_set & coco::interval_set::round_to_integer ( )

This method rounds the interval inward to integer borders.

Definition at line 445 of file interval_set.cc.

10.175.2.44 double coco::interval_set::safeguarded_mid ( ) const

Definition at line 418 of file interval_set.cc.

10.175.2.45 void coco::interval_set::set_fillext (unsigned int *m*) [inline]

Definition at line 268 of file interval_set.h.

10.175.2.46 void coco::interval_set::set_maxlen (unsigned int *m*) [inline]

Definition at line 260 of file interval_set.h.

10.175.2.47 void coco::interval_set::setSorting (bool *s*)

Definition at line 74 of file interval_set.cc.

10.175.2.48 void coco::interval_set::trimintervaloSize ( ) [inline]

10.175.2.49 void coco::interval_set::trimToSize ( ) [inline]

Definition at line 295 of file interval_set.h.

10.175.2.50 double coco::interval_set::volume ( ) const

Definition at line 436 of file interval_set.cc.

10.175.2.51 double coco::interval_set::width ( ) const

Definition at line 397 of file interval_set.cc.

The documentation for this class was generated from the following files:

- interval_set.h
- interval_set.cc

10.176 coco::coco::interval_st Struct Reference

`#include <expression.h>`

Public Member Functions

- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)

Public Attributes

- double l
- double u

10.176.1 Detailed Description

Definition at line 106 of file expression.h.

10.176.2 Member Function Documentation

10.176.2.1 interval_st & coco::interval_st::operator= (const interval & __i) [inline]

Definition at line 753 of file interval_boost.h.

10.176.2.2 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.3 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.4 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.5 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.6 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.7 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.8 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.9 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.10 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.11 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.2.12 interval_st& coco::coco::interval_st::operator= (const interval & __i)

10.176.3 Member Data Documentation

10.176.3.1 double coco::interval_st::l

Definition at line 108 of file expression.h.

10.176.3.2 double coco::interval_st::u

Definition at line 108 of file expression.h.

The documentation for this struct was generated from the following files:

- interval_boost.h
- interval_filib.h
- interval_profil.h

10.177 coco::coco::coco::interval_st Struct Reference

Public Member Functions

- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)

Public Attributes

- double l
- double u

10.177.1 Detailed Description

Definition at line 106 of file search_graph.cc.

10.177.2 Member Function Documentation

10.177.2.1 interval_st & coco::coco::interval_st::operator= (const interval & __i) [inline]

Definition at line 754 of file expression.h.

10.177.2.2 interval_st& coco::coco::coco::interval_st::operator= (const interval & __i)

10.177.3 Member Data Documentation

10.177.3.1 double coco::coco::interval_st::l

Definition at line 108 of file search_graph.cc.

10.177.3.2 double coco::coco::interval_st::u

Definition at line 108 of file search_graph.cc.

The documentation for this struct was generated from the following file:

- interval_boost.h

10.178 coco::interval_st Struct Reference

Constructor-free interval.

```
#include <interval_boost.h>
```

Public Member Functions

- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)
- interval_st & operator= (const interval &__i)

Public Attributes

- double l
- double u

10.178.1 Detailed Description

This is a constructor-free interval data type for storing intervals within unions.

Definition at line 55 of file interval_filib.h.

10.178.2 Member Function Documentation

10.178.2.1 interval_st& coco::interval_st::operator= (const interval & __i)

10.178.2.2 interval_st& coco::interval_st::operator= (const interval & __i)

10.178.2.3 interval_st& coco::interval_st::operator= (const interval & __i)

10.178.3 Member Data Documentation

10.178.3.1 double coco::interval_st::l

Definition at line 107 of file interval_boost.h.

10.178.3.2 double coco::interval_st::u

Definition at line 107 of file interval_boost.h.

The documentation for this struct was generated from the following files:

- interval_boost.h
- interval_filib.h
- interval_profil.h

10.179 Islope Class Reference

Class for computing intervalslopes up to order 2.

`#include <Islope.h>`

Public Member Functions

- Islope (void)
- Islope (int n)
- Islope (double a, int n)
- Islope (double a, double b, int n)
- Islope (const Islope &a)

- virtual ~Islope ()
- Islope & operator= (const Islope &a)
- I & operator[ ] (int pos) const
- void setDeg (int n)
- Islope & operator+= (const Islope &b)
- Islope & operator+= (const double &b)
- Islope & operator-= (const Islope &b)
- Islope & operator-= (const double &b)
- Islope & operator*= (const Islope &b)
- Islope & operator*= (const double &b)
- Islope & operator/= (const Islope &b)
- Islope & operator/= (const double &b)

Static Public Member Functions

- static Islope getVar (const double &x, int n)
- static Islope getVar (const double &x, const double &y, int n)
- static Islope getVar (const I &x, int n)
- static Islope getVar (const double &x, const double &y, const double &z, int n)

Public Attributes

- int order
- double * minmaxval

10.179.1 Constructor & Destructor Documentation

10.179.1.1 Islope::Islope (void) [inline]

Constructor for the slope of the constant function f(x)=0. The slope order is 1.

Definition at line 136 of file Islope.h.

10.179.1.2 Islope::Islope (int *n*) [inline]

Constructor for the slope of order n of the constant function f(x)=0. n has to be 2.

Definition at line 149 of file Islope.h.

10.179.1.3 Islope::Islope (double *a*, int *n*) [inline]

Constructor for the slope of order n of the constant function f(x)=a. n has to be 2.

Definition at line 181 of file Islope.h.

10.179.1.4 Islope::Islope (double *a*, double *b*, int *n*) [inline]

Constructor for the slope of order n of a constant function with f(x) in [a,b], (a+b)/2 is used as middlepoint. n has to be 2.

Definition at line 214 of file Islope.h.

10.179.1.5 Islope::Islope (const Islope & *a*) [inline]

Copy Constructor

Definition at line 246 of file Islope.h.

10.179.1.6 Islope::∼Islope ( ) [virtual]

Deconstructor

Definition at line 9 of file Islope.cc.

10.179.2 Member Function Documentation

10.179.2.1 Islope Islope::getVar (const double & *x*, int *n*) [inline, static]

Constructs a slope of order n for the function f(x)=x at a specific point x. n has to be 2.

Definition at line 288 of file Islope.h.

10.179.2.2 Islope Islope::getVar (const double & *x*, const double & *y*, int *n*) [inline, static]

Constructs a slope of order n for the function f(x)=x in a specific interval [x,y]. (x+y)/2 is used as middlepoint. n has to be 2.

Definition at line 321 of file Islope.h.

10.179.2.3 Islope Islope::getVar (const I & *x*, int *n*) [inline, static]

Constructs a slope of order n for the function f(x)=x in a specific interval [x,y]. (x+y)/2 is used as middlepoint. n has to be 2.

Definition at line 354 of file Islope.h.

10.179.2.4 Islope Islope::getVar (const double & *x*, const double & *y*, const double & *z*, int *n*) [inline, static]

Constructs a slope of order n for the function f(x)=x in a specific interval [x,y]. z is used as middlepoint. n has to be 2.

Definition at line 387 of file Islope.h.

10.179.2.5 Islope & Islope::operator∗= (const Islope & *b*) [inline]

Operator ∗=. Multiplicates slope a with slope b.

Definition at line 905 of file Islope.h.

10.179.2.6 Islope & Islope::operator∗= (const double & *b*) [inline]

Operator ∗=. Multiplicates slope a with double b.

Definition at line 932 of file Islope.h.

10.179.2.7 Islope & Islope::operator+= (const Islope & *b*) [inline]

Operator +=. Adds slope a and b.

Definition at line 819 of file Islope.h.

10.179.2.8 Islope & Islope::operator+= (const double & *b*) [inline]

Operator +=. Adds slope a and double b.

Definition at line 846 of file Islope.h.

10.179.2.9 Islope & Islope::operator-= (const Islope & *b*) [inline]

Operator -=. Subtracts slope b from slope a.

Definition at line 862 of file Islope.h.

10.179.2.10 Islope & Islope::operator-= (const double & *b*) [inline]

Operator -=. Subtracts double b from slope a.

Definition at line 889 of file Islope.h.

10.179.2.11 Islope & Islope::operator/= (const Islope & *b*) [inline]

Operator /=. Divides the slope a by slope b.

Definition at line 952 of file Islope.h.

10.179.2.12 Islope & Islope::operator/= (const double & *b*) [inline]

Operator /=. Divides the slope a by double b.

Definition at line 983 of file Islope.h.

10.179.2.13 Islope & Islope::operator= (const Islope & *a*) [inline]

Operator = for the slope class.

Definition at line 257 of file Islope.h.

10.179.2.14 I & Islope::operator[ ] (int *pos*) const [inline]

Definition at line 279 of file Islope.h.

10.179.2.15 void Islope::setDeg (int *n*) [inline]

Sets the order of the slope to n. If order is set to 1, data[3]-data[6] is deleted. If order is set to 2 data[3]-data[6] is initialized as 0.

Definition at line 420 of file Islope.h.

10.179.3 Member Data Documentation

10.179.3.1 double* Islope::minmaxval

here is the data stored we need for computing the function value of x_min and x_max. x_min and x_max are the bounds of the variable x.

Definition at line 26 of file Islope.h.

10.179.3.2 int Islope::order

the order of the slope. it can be 1 or 2.

Definition at line 24 of file Islope.h.

The documentation for this class was generated from the following files:

- Islope.h
- Islope.cc

10.180 coco::Islope_eval Class Reference

Forward function range evaluation.

`#include <Islope_evaluator.h>`

Inheritance diagram for coco::Islope_eval:

Collaboration diagram for coco::Islope_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- Islope_eval (const std::vector< Islope > &__x, const variable_indicator &__v, const model &__m, std::vector< Islope > *__c, int __deg)
- Islope_eval (const Islope_eval &__v)

- ~Islope_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_interval (const std::vector< Islope > &__x, const variable_indicator &__v)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const Islope &__rval)
- int update (const expression_node &__data, const Islope &__rval)
- Islope calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.180.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural Islope extension.

10.180.2 Member Typedef Documentation

10.180.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.180.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.180.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.180.3 Constructor & Destructor Documentation

10.180.3.1 coco::Islope_eval::Islope_eval (const std::vector< Islope > & __x, const variable_indicator & __v, const model & __m, std::vector< Islope > * __c, int __deg) [inline]

Constructor: ___x is the box over which the function range shall be evaluated, ___v is a variable indicator specifying the variables that have changed value since the last evaluation, ___m specifies the DAG, and ___c the cache (may be NULL). The boolean do_i determines whether the natural Islope extension will use known node ranges for reducing the ranges.

Definition at line 143 of file Islope_evaluator.h.

10.180.3.2 coco::Islope_eval::Islope_eval (const Islope_eval & __v) [inline]

Standard Copy Constructor

Definition at line 158 of file Islope_evaluator.h.

10.180.3.3 coco::Islope_eval::~Islope_eval ( ) [inline]

Standard Destructor

Definition at line 161 of file Islope_evaluator.h.

10.180.4 Member Function Documentation

10.180.4.1 void coco::Islope_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 245 of file Islope_evaluator.h.

10.180.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.180.4.3 Islope coco::Islope_eval::calculate_value (bool eval_all) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 703 of file Islope_evaluator.h.

10.180.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.180.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.180.4.6 void coco::Islope_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 180 of file Islope_evaluator.h.

10.180.4.7 int coco::Islope_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 182 of file Islope_evaluator.h.

10.180.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The ___data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.180.4.9 bool coco::Islope_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the range for this node is already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 88 of file Islope_evaluator.h.

10.180.4.10 void coco::Islope_eval::new_interval (const std::vector< Islope > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation point to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 172 of file Islope_evaluator.h.

10.180.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.180.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.180.4.13 void coco::Islope_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 252 of file Islope_evaluator.h.

10.180.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.180.4.15 expression_const_walker coco::Islope_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 164 of file Islope_evaluator.h.

10.180.4.16 int coco::Islope_eval::update (const Islope & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 257 of file Islope_evaluator.h.

10.180.4.17 int coco::Islope_eval::update (const expression_node & __data, const Islope & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 264 of file Islope_evaluator.h.

10.180.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.180.4.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.180.4.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.180.4.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.180.4.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.180.4.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

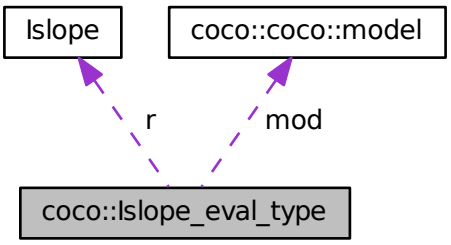
- Islope_evaluator.h

10.181 coco::Islope_eval_type Struct Reference

Visitor data for Islope_eval.

`#include <Islope_evaluator.h>`

Collaboration diagram for coco::Islope_eval_type:



Public Attributes

- const std::vector< Islope > * x
- std::vector< Islope > * cache
- const model * mod
- void * p
- Islope r
- unsigned int n
- int deg

10.181.1 Detailed Description

This class is the visitor data type for the Islope_eval evaluator.

10.181.2 Member Data Documentation

10.181.2.1 std::vector<Islope>* coco::Islope_eval_type::cache

function range the function range cache

Definition at line 60 of file Islope_evaluator.h.

10.181.2.2 int coco::Islope_eval_type::deg

degree of slope

Definition at line 65 of file Islope_evaluator.h.

10.181.2.3 const model* coco::Islope_eval_type::mod

the DAG

Definition at line 61 of file Islope_evaluator.h.

10.181.2.4 unsigned int coco::Islope_eval_type::n

children counter

Definition at line 64 of file Islope_evaluator.h.

10.181.2.5 void* coco::Islope_eval_type::p

additional data for

Definition at line 62 of file Islope_evaluator.h.

10.181.2.6 Islope coco::Islope_eval_type::r

return value

Definition at line 63 of file Islope_evaluator.h.

10.181.2.7 const std::vector<Islope>* coco::Islope_eval_type::x

Definition at line 59 of file Islope_evaluator.h.

The documentation for this struct was generated from the following file:

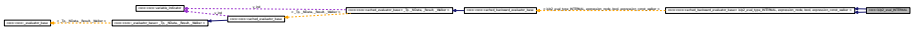
- Islope_evaluator.h

10.182 coco::islp2_eval_INTERNAL Class Reference

`#include <islp2_evaluator.h>`

Inheritance diagram for coco::islp2_eval_INTERNAL:

Collaboration diagram for coco::islp2_eval_INTERNAL:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- islp2_eval_INTERNAL (std::vector< std::vector< interval > > *__c_data, std::vector< std::vector< interval > > *__d_data, std::vector< std::vector< interval > > *__h_data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__c, std::vector< std::vector< interval > > *__d, std::vector< std::vector< interval > > *__h, std::vector< interval > *__cslp, std::vector< interval > *__islp, std::vector< interval > *__islp2)
- islp2_eval_INTERNAL (const islp2_eval_INTERNAL &__he)
- ~islp2_eval_INTERNAL ()
- void newPoint (std::vector< std::vector< interval > > *__c_data, std::vector< std::vector< interval > > *__d_data, std::vector< std::vector< interval > > *__h_data, const variable_indicator &__v)
- void newResult (std::vector< interval > *__islp, std::vector< interval > *__islp2)
- void set_mult (interval scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)
- islp2_eval_INTERNAL (std::vector< std::vector< interval > > *__c_data, std::vector< std::vector< interval > > *__d_data, std::vector< std::vector< interval > > *__h_data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__c, std::vector< std::vector< interval > > *__d, std::vector< std::vector< interval > > *__h, std::vector< interval > *__cslp, std::vector< interval > *__islp, std::vector< interval > *__islp2)
- islp2_eval_INTERNAL (const islp2_eval_INTERNAL &__he)
- ~islp2_eval_INTERNAL ()
- void newPoint (std::vector< std::vector< interval > > *__c_data, std::vector< std::vector< interval > > *__d_data, std::vector< std::vector< interval > > *__h_data, const variable_indicator &__v)
- void newResult (std::vector< interval > *__islp, std::vector< interval > *__islp2)
- void set_mult (interval scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)
- bool is_cached (const node_data_type &__data)

10.182.1 Detailed Description

Definition at line 59 of file islp2_evaluator.h.

10.182.2 Member Typedef Documentation

10.182.2.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.182.2.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.182.2.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.182.3 Constructor & Destructor Documentation

10.182.3.1 coco::islp2_eval_INTERNAL::islp2_eval_INTERNAL (std::vector< std::vector< interval > > * __c_data, std::vector< std::vector< interval > > * __d_data, std::vector< std::vector< interval > > * __h_data, variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > * __c, std::vector< std::vector< interval > > * __d, std::vector< std::vector< interval > > * __h, std::vector< interval > * __cslp, std::vector< interval > * __islp, std::vector< interval > * __islp2) [inline]

Definition at line 79 of file islp2_evaluator.h.

10.182.3.2 coco::islp2_eval_INTERNAL::islp2_eval_INTERNAL (const islp2_eval_INTERNAL & __he) [inline]

Definition at line 126 of file islp2_evaluator.h.

10.182.3.3 coco::islp2_eval_INTERNAL::~islp2_eval_INTERNAL ( ) [inline]

Definition at line 128 of file islp2_evaluator.h.

10.182.3.4 coco::islp2_eval_INTERNAL::islp2_eval_INTERNAL (std::vector< std::vector< interval > > * __c_data, std::vector< std::vector< interval > > * __d_data, std::vector< std::vector< interval > > * __h_data, variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > * __c, std::vector< std::vector< interval > > * __d, std::vector< std::vector< interval > > * __h, std::vector< interval > * __cslp, std::vector< interval > * __islp, std::vector< interval > * __islp2) [inline]

Definition at line 79 of file islp2_evaluator.h.

10.182.3.5 coco::islp2_eval_INTERNAL::islp2_eval_INTERNAL (const islp2_eval_INTERNAL & __he) [inline]

Definition at line 126 of file islp2_evaluator.h.

10.182.3.6 coco::islp2_eval_INTERNAL::~islp2_eval_INTERNAL ( ) [inline]

Definition at line 128 of file islp2_evaluator.h.

10.182.4 Member Function Documentation

10.182.4.1 int coco::islp2_eval_INTERNAL::calculate (const expression_node & __data) [inline]

Definition at line 171 of file islp2_evaluator.h.

10.182.4.2 int coco::islp2_eval_INTERNAL::calculate (const expression_node & __data) [inline]

Definition at line 171 of file islp2_evaluator.h.

10.182.4.3 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.182.4.4 bool coco::islp2_eval_INTERNAL::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp2_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 412 of file islp2_evaluator.h.

10.182.4.5 bool coco::islp2_eval_INTERNAL::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp2_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 412 of file islp2_evaluator.h.

10.182.4.6 void coco::islp2_eval_INTERNAL::cleanup (const expression_node & *__data*) [inline]

Definition at line 311 of file islp2_evaluator.h.

10.182.4.7 void coco::islp2_eval_INTERNAL::cleanup (const expression_node & *__data*) [inline]

Definition at line 311 of file islp2_evaluator.h.

10.182.4.8 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.182.4.9 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.182.4.10 void coco::islp2_eval_INTERNAL::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp2_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 160 of file islp2_evaluator.h.

10.182.4.11 void coco::islp2_eval_INTERNAL::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp2_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 160 of file islp2_evaluator.h.

10.182.4.12 bool coco::islp2_eval_INTERNAL::is_cached (const node_data_type & *__data*) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp2_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 68 of file islp2_evaluator.h.

10.182.4.13 bool coco::islp2_eval_INTERNAL::is_cached (const node_data_type & *__data*) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp2_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 68 of file islp2_evaluator.h.

10.182.4.14 void coco::islp2_eval_INTERNAL::newPoint (std::vector< std::vector< interval > > * *__c_data*, std::vector< std::vector< interval > > * *__d_data*, std::vector< std::vector< interval > > * *__h_data*, const variable_indicator & *__v*) [inline]

Definition at line 130 of file islp2_evaluator.h.

10.182.4.15 void coco::islp2_eval_INTERNAL::newPoint (std::vector< std::vector< interval > > * *__c_data*, std::vector< std::vector< interval > > * *__d_data*, std::vector< std::vector< interval > > * *__h_data*, const variable_indicator & *__v*) [inline]

Definition at line 130 of file islp2_evaluator.h.

10.182.4.16 void coco::islp2_eval_INTERNAL::newResult (std::vector< interval > * *__islp*, std::vector< interval > * *__islp2*) [inline]

Definition at line 142 of file islp2_evaluator.h.

10.182.4.17 void coco::islp2_eval_INTERNAL::newResult (std::vector< interval > * __islp, std::vector< interval > * __islp2) [inline]

Definition at line 142 of file islp2_evaluator.h.

10.182.4.18 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.182.4.19 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.182.4.20 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.182.4.21 void coco::islp2_eval_INTERNAL::set_mult (interval scal) [inline]

Definition at line 148 of file islp2_evaluator.h.

10.182.4.22 void coco::islp2_eval_INTERNAL::set_mult (interval scal) [inline]

Definition at line 148 of file islp2_evaluator.h.

10.182.4.23 expression_const_walker coco::islp2_eval_INTERNAL::short_cut_to (const expression_node & __data) [inline]

Definition at line 156 of file islp2_evaluator.h.

10.182.4.24 expression_const_walker coco::islp2_eval_INTERNAL::short_cut_to (const expression_node & __data) [inline]

Definition at line 156 of file islp2_evaluator.h.

10.182.4.25 int coco::islp2_eval_INTERNAL::update (const bool & __rval) [inline]

Definition at line 349 of file islp2_evaluator.h.

10.182.4.26 int coco::islp2_eval_INTERNAL::update (const bool & *__rval*) [inline]

Definition at line 349 of file islp2_evaluator.h.

10.182.4.27 int coco::islp2_eval_INTERNAL::update (const expression_node & *__data,* const bool & *__rval*) [inline]

Definition at line 359 of file islp2_evaluator.h.

10.182.4.28 int coco::islp2_eval_INTERNAL::update (const expression_node & *__data,* const bool & *__rval*) [inline]

Definition at line 359 of file islp2_evaluator.h.

10.182.4.29 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.182.4.30 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & *__data,* const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.182.4.31 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 892 of file search_graph.cc.

10.182.4.32 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.182.4.33 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.182.4.34 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

The documentation for this class was generated from the following file:

- islp2_evaluator_tmpl.h

10.183 islp2_eval_INTERNAL Class Reference

`#include <islp2_evaluator_tmpl.h>`

Public Member Functions

- islp2_eval_INTERNAL (std::vector< std::vector< interval > > *__c_data, std::vector< std::vector< interval > > *__d_data, std::vector< std::vector< interval > > *__h_data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__c, std::vector< std::vector< interval > > *__d, std::vector< std::vector< interval > > *__h, std::vector< interval > *__cslp, std::vector< interval > *__islp, std::vector< interval > *__islp2)
- islp2_eval_INTERNAL (const islp2_eval_INTERNAL &__he)
- ~islp2_eval_INTERNAL ()
- void newPoint (std::vector< std::vector< interval > > *__c_data, std::vector< std::vector< interval > > *__d_data, std::vector< std::vector< interval > > *__h_data, const variable_indicator &__v)
- void newResult (std::vector< interval > *__islp, std::vector< interval > *__islp2)
- void set_mult (interval scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.183.1 Constructor & Destructor Documentation

10.183.1.1 islp2_eval_INTERNAL::islp2_eval_INTERNAL (std::vector< std::vector< interval > > * __c_data, std::vector< std::vector< interval > > * __d_data, std::vector< std::vector< interval > > * __h_data, variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > * __c, std::vector< std::vector< interval > > * __d, std::vector< std::vector< interval > > * __h, std::vector< interval > * __cslp, std::vector< interval > * __islp, std::vector< interval > * __islp2) [inline]

Definition at line 78 of file islp2_evaluator_tmpl.h.

10.183.1.2 islp2_eval_INTERNAL::islp2_eval_INTERNAL (const islp2_eval_INTERNAL & __he) [inline]

Definition at line 125 of file islp2_evaluator_tmpl.h.

10.183.1.3 islp2_eval_INTERNAL::~islp2_eval_INTERNAL ( ) [inline]

Definition at line 127 of file islp2_evaluator_tmpl.h.

10.183.2 Member Function Documentation

10.183.2.1 int islp2_eval_INTERNAL::calculate (const expression_node & __data) [inline]

Definition at line 170 of file islp2_evaluator_tmpl.h.

10.183.2.2 bool islp2_eval_INTERNAL::calculate_value (bool eval_all) [inline]

Definition at line 411 of file islp2_evaluator_tmpl.h.

10.183.2.3 void islp2_eval_INTERNAL::cleanup (const expression_node & __data) [inline]

Definition at line 310 of file islp2_evaluator_tmpl.h.

10.183.2.4 void islp2_eval_INTERNAL::initialize ( ) [inline]

Definition at line 159 of file islp2_evaluator_tmpl.h.

10.183.2.5 bool islp2_eval_INTERNAL::is_cached (const node_data_type & __data) [inline, protected]

Definition at line 67 of file islp2_evaluator_tmpl.h.

10.183.2.6 void islp2_eval_INTERNAL::newPoint (std::vector< std::vector< interval > > * __c_data, std::vector< std::vector< interval > > * __d_data, std::vector< std::vector< interval > > * __h_data, const variable_indicator & __v) [inline]

Definition at line 129 of file islp2_evaluator_tmpl.h.

10.183.2.7 void islp2_eval_INTERNAL::newResult (std::vector< interval > * *__islp*, std::vector< interval > * *__islp2*) [inline]

Definition at line 141 of file islp2_evaluator_tmpl.h.

10.183.2.8 void islp2_eval_INTERNAL::set_mult (interval *scal*) [inline]

Definition at line 147 of file islp2_evaluator_tmpl.h.

10.183.2.9 expression_const_walker islp2_eval_INTERNAL::short_cut_to (const expression_node & *__data*) [inline]

Definition at line 155 of file islp2_evaluator_tmpl.h.

10.183.2.10 int islp2_eval_INTERNAL::update (const bool & *__rval*) [inline]

Definition at line 348 of file islp2_evaluator_tmpl.h.

10.183.2.11 int islp2_eval_INTERNAL::update (const expression_node & *__data*, const bool & *__rval*) [inline]

Definition at line 358 of file islp2_evaluator_tmpl.h.

The documentation for this class was generated from the following file:

- islp2_evaluator_tmpl.h

10.184 islp2_eval_type_INTERNAL Struct Reference

`#include <islp2_evaluator_tmpl.h>`

Public Attributes

- std::vector< std::vector < interval > > * c_data
- std::vector< std::vector < interval > > * d_data
- std::vector< std::vector < interval > > * h_data
- std::vector< std::vector < interval > > * c_cache
- std::vector< std::vector < interval > > * d_cache
- std::vector< std::vector < interval > > * h_cache
- std::vector< interval > * cslp_vec
- std::vector< interval > * islp_vec
- std::vector< interval > * islp2_vec
- bool calc_islp
- const model * mod
- interval c_mult
- interval d_mult
- interval islp2_of_current_node
- bool islp2_zero
- interval c_mult_trans
- interval d_mult_trans

- interval h_mult
- interval h_mult_trans
- bool is_linear
- unsigned int child_n

10.184.1 Member Data Documentation

10.184.1.1 std::vector<std::vector<interval> >* islp2_eval_type_INTERNAL::c_cache

Definition at line 32 of file islp2_evaluator_tmpl.h.

10.184.1.2 std::vector<std::vector<interval> >* islp2_eval_type_INTERNAL::c_data

Definition at line 29 of file islp2_evaluator_tmpl.h.

10.184.1.3 interval islp2_eval_type_INTERNAL::c_mult

Definition at line 44 of file islp2_evaluator_tmpl.h.

10.184.1.4 interval islp2_eval_type_INTERNAL::c_mult_trans

Definition at line 48 of file islp2_evaluator_tmpl.h.

10.184.1.5 bool islp2_eval_type_INTERNAL::calc_islp

Definition at line 42 of file islp2_evaluator_tmpl.h.

10.184.1.6 unsigned int islp2_eval_type_INTERNAL::child_n

Definition at line 53 of file islp2_evaluator_tmpl.h.

10.184.1.7 std::vector<interval>* islp2_eval_type_INTERNAL::cslp_vec

Definition at line 39 of file islp2_evaluator_tmpl.h.

10.184.1.8 std::vector<std::vector<interval> >* islp2_eval_type_INTERNAL::d_cache

Definition at line 33 of file islp2_evaluator_tmpl.h.

10.184.1.9 std::vector<std::vector<interval> >* islp2_eval_type_INTERNAL::d_data

Definition at line 30 of file islp2_evaluator_tmpl.h.

10.184.1.10 interval islp2_eval_type_INTERNAL::d_mult

Definition at line 45 of file islp2_evaluator_tmpl.h.

10.184.1.11 interval islp2_eval_type_INTERNAL::d_mult_trans

Definition at line 49 of file islp2_evaluator_tmpl.h.

10.184.1.12 std::vector<std::vector<interval> >* islp2_eval_type_INTERNAL::h_cache

Definition at line 34 of file islp2_evaluator_tmpl.h.

10.184.1.13 std::vector<std::vector<interval> >* islp2_eval_type_INTERNAL::h_data

Definition at line 31 of file islp2_evaluator_tmpl.h.

10.184.1.14 interval islp2_eval_type_INTERNAL::h_mult

Definition at line 50 of file islp2_evaluator_tmpl.h.

10.184.1.15 interval islp2_eval_type_INTERNAL::h_mult_trans

Definition at line 51 of file islp2_evaluator_tmpl.h.

10.184.1.16 bool islp2_eval_type_INTERNAL::is_linear

Definition at line 52 of file islp2_evaluator_tmpl.h.

10.184.1.17 interval islp2_eval_type_INTERNAL::islp2_of_current_node

Definition at line 46 of file islp2_evaluator_tmpl.h.

10.184.1.18 std::vector<interval>* islp2_eval_type_INTERNAL::islp2_vec

Definition at line 41 of file islp2_evaluator_tmpl.h.

10.184.1.19 bool islp2_eval_type_INTERNAL::islp2_zero

Definition at line 47 of file islp2_evaluator_tmpl.h.

10.184.1.20 std::vector<interval>* islp2_eval_type_INTERNAL::islp_vec

Definition at line 40 of file islp2_evaluator_tmpl.h.

10.184.1.21 const model* islp2_eval_type_INTERNAL::mod

Definition at line 43 of file islp2_evaluator_tmpl.h.

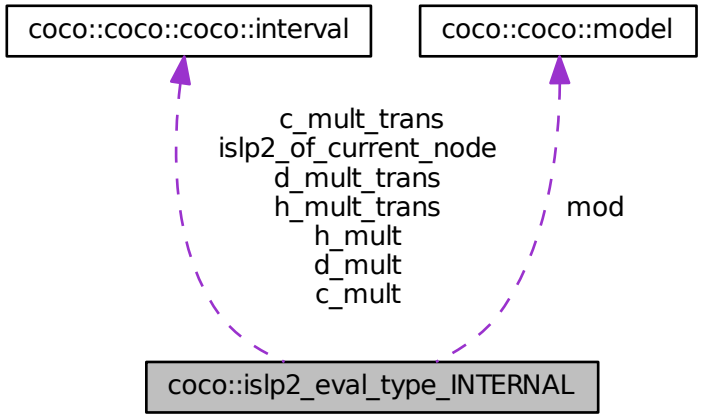
The documentation for this struct was generated from the following file:

- islp2_evaluator_tmpl.h

10.185 coco::islp2_eval_type_INTERNAL Struct Reference

```
#include <islp2_evaluator.h>
```

Collaboration diagram for coco::islp2_eval_type_INTERNAL:



Public Attributes

- std::vector< std::vector < interval > > * c_data
- std::vector< std::vector < interval > > * d_data
- std::vector< std::vector < interval > > * h_data
- std::vector< std::vector < interval > > * c_cache
- std::vector< std::vector < interval > > * d_cache
- std::vector< std::vector < interval > > * h_cache
- std::vector< interval > * cslp_vec
- std::vector< interval > * islp_vec
- std::vector< interval > * islp2_vec
- bool calc_islp
- const model * mod
- interval c_mult
- interval d_mult
- interval islp2_of_current_node
- bool islp2_zero
- interval c_mult_trans
- interval d_mult_trans
- interval h_mult
- interval h_mult_trans
- bool is_linear
- unsigned int child_n

10.185.1 Detailed Description

Definition at line 28 of file islp2_evaluator.h.

10.185.2 Member Data Documentation

10.185.2.1 std::vector< std::vector< interval > > * islp2_eval_type_INTERNAL::c_cache

Definition at line 33 of file islp2_evaluator.h.

10.185.2.2 std::vector< std::vector< interval > > * islp2_eval_type_INTERNAL::c_data

Definition at line 30 of file islp2_evaluator.h.

10.185.2.3 interval islp2_eval_type_INTERNAL::c_mult

Definition at line 45 of file islp2_evaluator.h.

10.185.2.4 interval islp2_eval_type_INTERNAL::c_mult_trans

Definition at line 49 of file islp2_evaluator.h.

10.185.2.5 bool islp2_eval_type_INTERNAL::calc_islp

Definition at line 43 of file islp2_evaluator.h.

10.185.2.6 unsigned int islp2_eval_type_INTERNAL::child_n

Definition at line 54 of file islp2_evaluator.h.

10.185.2.7 std::vector< interval > * islp2_eval_type_INTERNAL::cslp_vec

Definition at line 40 of file islp2_evaluator.h.

10.185.2.8 std::vector< std::vector< interval > > * islp2_eval_type_INTERNAL::d_cache

Definition at line 34 of file islp2_evaluator.h.

10.185.2.9 std::vector< std::vector< interval > > * islp2_eval_type_INTERNAL::d_data

Definition at line 31 of file islp2_evaluator.h.

10.185.2.10 interval islp2_eval_type_INTERNAL::d_mult

Definition at line 46 of file islp2_evaluator.h.

10.185.2.11 interval islp2_eval_type_INTERNAL::d_mult_trans

Definition at line 50 of file islp2_evaluator.h.

10.185.2.12 std::vector< std::vector< interval > > * islp2_eval_type_INTERNAL::h_cache

Definition at line 35 of file islp2_evaluator.h.

10.185.2.13 std::vector< std::vector< interval > > * islp2_eval_type_INTERNAL::h_data

Definition at line 32 of file islp2_evaluator.h.

10.185.2.14 interval islp2_eval_type_INTERNAL::h_mult

Definition at line 51 of file islp2_evaluator.h.

10.185.2.15 interval islp2_eval_type_INTERNAL::h_mult_trans

Definition at line 52 of file islp2_evaluator.h.

10.185.2.16 bool islp2_eval_type_INTERNAL::is_linear

Definition at line 53 of file islp2_evaluator.h.

10.185.2.17 interval islp2_eval_type_INTERNAL::islp2_of_current_node

Definition at line 47 of file islp2_evaluator.h.

10.185.2.18 std::vector< interval > * islp2_eval_type_INTERNAL::islp2_vec

Definition at line 42 of file islp2_evaluator.h.

10.185.2.19 bool islp2_eval_type_INTERNAL::islp2_zero

Definition at line 48 of file islp2_evaluator.h.

10.185.2.20 std::vector< interval > * islp2_eval_type_INTERNAL::islp_vec

Definition at line 41 of file islp2_evaluator.h.

10.185.2.21 const model * islp2_eval_type_INTERNAL::mod

Definition at line 44 of file islp2_evaluator.h.

The documentation for this struct was generated from the following file:

- islp2_evaluator_tmpl.h

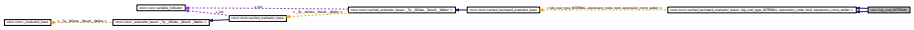
10.186 coco::islp_eval_INTERNAL Class Reference

Backward first order slope evaluation with prepared first order slope data.

`#include <islp_evaluator.h>`

Inheritance diagram for coco::islp_eval_INTERNAL:

Collaboration diagram for coco::islp_eval_INTERNAL:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- islp_eval_INTERNAL (std::vector< std::vector< interval > > &__slp_data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__d, std::vector< interval > &__islp)
- islp_eval_INTERNAL (const islp_eval_INTERNAL &__d)
- ~islp_eval_INTERNAL ()
- void new_point (std::vector< std::vector< interval > > &__slp_data, const variable_indicator &__v)
- void new_result (std::vector< interval > &__islp)
- void set_mult (interval scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- islp_eval_INTERNAL (std::vector< std::vector< interval > > &__slp_data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__d, std::vector< interval > &__islp)
- islp_eval_INTERNAL (const islp_eval_INTERNAL &__d)
- ~islp_eval_INTERNAL ()
- void new_point (std::vector< std::vector< interval > > &__slp_data, const variable_indicator &__v)
- void new_result (std::vector< interval > &__islp)
- void set_mult (interval scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)

- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)
- bool is_cached (const node_data_type &__data)

10.186.1 Detailed Description

This class is a cached backward evaluator performing a first order slope evaluation with available partial first order slope information.

Definition at line 55 of file islp_evaluator.h.

10.186.2 Member Typedef Documentation

10.186.2.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.186.2.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.186.2.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.186.3 Constructor & Destructor Documentation

10.186.3.1 coco::islp_eval_INTERNAL::islp_eval_INTERNAL (std::vector< std::vector< interval > > & __slp_data, variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > * __d, std::vector< interval > & __islp) [inline]

Constructor: __slp_data contains the partial first order slope information, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies the DAG, and __d the cache (may be NULL). Eventually, __islp is a reference to the result vector. This vector must have the correct length and has to be initialized with zero, since all components are actually computed by adding to the components of __islp.

Definition at line 90 of file islp_evaluator.h.

10.186.3.2 coco::islp_eval_INTERNAL::islp_eval_INTERNAL (const islp_eval_INTERNAL & __d) [inline]

Standard Copy Constructor

Definition at line 114 of file islp_evaluator.h.

10.186.3.3 coco::islp_eval_INTERNAL::~islp_eval_INTERNAL ( ) [inline]

Standard Destructor

Definition at line 117 of file islp_evaluator.h.

10.186.3.4 coco::islp_eval_INTERNAL::islp_eval_INTERNAL (std::vector< std::vector< interval > > & __slp_data, variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > * __d, std::vector< interval > & __islp) [inline]

Constructor: __slp_data contains the partial first order slope information, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies the DAG, and __d the cache (may be NULL). Eventually, __islp is a reference to the result vector. This vector must have the correct length and has to be initialized with zero, since all components are actually computed by adding to the components of __islp.

Definition at line 90 of file islp_evaluator.h.

10.186.3.5 coco::islp_eval_INTERNAL::islp_eval_INTERNAL (const islp_eval_INTERNAL & __d) [inline]

Standard Copy Constructor

Definition at line 114 of file islp_evaluator.h.

10.186.3.6 coco::islp_eval_INTERNAL::~islp_eval_INTERNAL ( ) [inline]

Standard Destructor

Definition at line 117 of file islp_evaluator.h.

10.186.4 Member Function Documentation

10.186.4.1 int coco::islp_eval_INTERNAL::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 155 of file islp_evaluator.h.

10.186.4.2 int coco::islp_eval_INTERNAL::calculate (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 155 of file islp_evaluator.h.

10.186.4.3 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.186.4.4 bool coco::islp_eval_INTERNAL::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 267 of file islp_evaluator.h.

10.186.4.5 bool coco::islp_eval_INTERNAL::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 267 of file islp_evaluator.h.

10.186.4.6 void coco::islp_eval_INTERNAL::cleanup (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 218 of file islp_evaluator.h.

10.186.4.7 void coco::islp_eval_INTERNAL::cleanup (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 218 of file islp_evaluator.h.

10.186.4.8 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.186.4.9 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.186.4.10 void coco::islp_eval_INTERNAL::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 149 of file islp_evaluator.h.

10.186.4.11 void coco::islp_eval_INTERNAL::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 149 of file islp_evaluator.h.

10.186.4.12 bool coco::islp_eval_INTERNAL::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the first order slope for this node is already available.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 67 of file islp_evaluator.h.

10.186.4.13 bool coco::islp_eval_INTERNAL::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the first order slope for this node is already available.

Reimplemented from coco::coco::cached_backward_evaluator_base< islp_eval_type_INTERNAL, expression_node, bool, expression_const_walker >.

Definition at line 67 of file islp_evaluator.h.

10.186.4.14 void coco::islp_eval_INTERNAL::new_point (std::vector< std::vector< interval > > & __slp_data, const variable_indicator & __v) [inline]

This method changes the partial first order slope data to __slp_data. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 122 of file islp_evaluator.h.

10.186.4.15 void coco::islp_eval_INTERNAL::new_point (std::vector< std::vector< interval > > & __slp_data, const variable_indicator & __v) [inline]

This method changes the partial first order slope data to __slp_data. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 122 of file islp_evaluator.h.

10.186.4.16 void coco::islp_eval_INTERNAL::new_result (std::vector< interval > & __islp) [inline]

This method changes the result vector to __islp.

Definition at line 130 of file islp_evaluator.h.

10.186.4.17 void coco::islp_eval_INTERNAL::new_result (std::vector< interval > & __islp) [inline]

This method changes the result vector to __islp.

Definition at line 130 of file islp_evaluator.h.

10.186.4.18 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.186.4.19 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.186.4.20 void coco::islp_eval_INTERNAL::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 230 of file islp_evaluator.h.

10.186.4.21 void coco::islp_eval_INTERNAL::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 230 of file islp_evaluator.h.

10.186.4.22 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.186.4.23 void coco::islp_eval_INTERNAL::set_mult (interval *scal*) [inline]

This method causes the slope to be multiplied by `scal`.

Definition at line 136 of file islp_evaluator.h.

10.186.4.24 void coco::islp_eval_INTERNAL::set_mult (interval *scal*) [inline]

This method causes the slope to be multiplied by `scal`.

Definition at line 136 of file islp_evaluator.h.

10.186.4.25 expression_const_walker coco::islp_eval_INTERNAL::short_cut_to (const expression_node & *__data*) [inline]

NOP version, not needed

Definition at line 143 of file islp_evaluator.h.

10.186.4.26 expression_const_walker coco::islp_eval_INTERNAL::short_cut_to (const expression_node & *__data*) [inline]

NOP version, not needed

Definition at line 143 of file islp_evaluator.h.

10.186.4.27 int coco::islp_eval_INTERNAL::update (const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 238 of file islp_evaluator.h.

10.186.4.28 int coco::islp_eval_INTERNAL::update (const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 238 of file islp_evaluator.h.

10.186.4.29 int coco::islp_eval_INTERNAL::update (const expression_node & *__data,* const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 245 of file islp_evaluator.h.

10.186.4.30 int coco::islp_eval_INTERNAL::update (const expression_node & *__data,* const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 245 of file islp_evaluator.h.

10.186.4.31 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.186.4.32 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.186.4.33 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter false.

Definition at line 892 of file search_graph.cc.

10.186.4.34 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.186.4.35 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.186.4.36 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

The documentation for this class was generated from the following file:

- islp_evaluator_tmpl.h

10.187 islp_eval_INTERNAL Class Reference

Backward first order slope evaluation with prepared first order slope data.

`#include <islp_evaluator_tmpl.h>`

Public Member Functions

- islp_eval_INTERNAL (std::vector< std::vector< interval > > &__slp_data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__d, std::vector< interval > &__islp)
- islp_eval_INTERNAL (const islp_eval_INTERNAL &__d)
- ~islp_eval_INTERNAL ()
- void new_point (std::vector< std::vector< interval > > &__slp_data, const variable_indicator &__v)
- void new_result (std::vector< interval > &__islp)
- void set_mult (interval scal)
- expression_const_walker short_cut_to (const expression_node &__data)

- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.187.1 Detailed Description

This class is a cached backward evaluator performing a first order slope evaluation with available partial first order slope information.

10.187.2 Constructor & Destructor Documentation

10.187.2.1 islp_eval_INTERNAL::islp_eval_INTERNAL (std::vector< std::vector< interval > > & __slp_data, variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > * __d, std::vector< interval > & __islp) [inline]

Constructor: ___slp_data contains the partial first order slope information, ___v is a variable indicator specifying the variables that have changed value since the last evaluation, ___m specifies the DAG, and ___d the cache (may be NULL). Eventually, ___islp is a reference to the result vector. This vector must have the correct length and has to be initialized with zero, since all components are actually computed by adding to the components of ___islp.

Definition at line 89 of file islp_evaluator_tmpl.h.

10.187.2.2 islp_eval_INTERNAL::islp_eval_INTERNAL (const islp_eval_INTERNAL & __d) [inline]

Standard Copy Constructor

Definition at line 113 of file islp_evaluator_tmpl.h.

10.187.2.3 islp_eval_INTERNAL::~islp_eval_INTERNAL ( ) [inline]

Standard Destructor

Definition at line 116 of file islp_evaluator_tmpl.h.

10.187.3 Member Function Documentation

10.187.3.1 int islp_eval_INTERNAL::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 154 of file islp_evaluator_tmpl.h.

10.187.3.2 bool islp_eval_INTERNAL::calculate_value (bool eval_all) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 266 of file islp_evaluator_tmpl.h.

10.187.3.3 void islp_eval_INTERNAL::cleanup (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 217 of file islp_evaluator_tmpl.h.

10.187.3.4 void islp_eval_INTERNAL::initialize ( ) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 148 of file islp_evaluator_tmpl.h.

10.187.3.5 bool islp_eval_INTERNAL::is_cached (const node_data_type & *__data*) [inline, protected]

This function determines, whether the first order slope for this node is already available.

Definition at line 66 of file islp_evaluator_tmpl.h.

10.187.3.6 void islp_eval_INTERNAL::new_point (std::vector< std::vector< interval > > & *__slp_data,* const variable_indicator & *__v*) [inline]

This method changes the partial first order slope data to `___slp_data`. The parameter `___v` specifies which variables have changed value since the last evaluation.

Definition at line 121 of file islp_evaluator_tmpl.h.

10.187.3.7 void islp_eval_INTERNAL::new_result (std::vector< interval > & *__islp*) [inline]

This method changes the result vector to `___islp`.

Definition at line 129 of file islp_evaluator_tmpl.h.

10.187.3.8 void islp_eval_INTERNAL::retrieve_from_cache (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 229 of file islp_evaluator_tmpl.h.

10.187.3.9 void islp_eval_INTERNAL::set_mult (interval *scal*) [inline]

This method causes the slope to be multiplied by `scal`.

Definition at line 135 of file islp_evaluator_tmpl.h.

10.187.3.10 expression_const_walker islp_eval_INTERNAL::short_cut_to (const expression_node & *__data*) [inline]

NOP version, not needed

Definition at line 142 of file islp_evaluator_tmpl.h.

10.187.3.11 int islp_eval_INTERNAL::update (const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 237 of file islp_evaluator_tmpl.h.

10.187.3.12 int islp_eval_INTERNAL::update (const expression_node & *__data,* const bool & *__rval*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 244 of file islp_evaluator_tmpl.h.

The documentation for this class was generated from the following file:

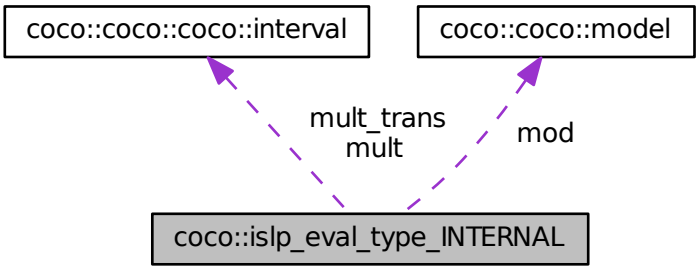
- islp_evaluator_tmpl.h

10.188 coco::islp_eval_type_INTERNAL Struct Reference

Visitor data for islp_eval.

`#include <islp_evaluator.h>`

Collaboration diagram for coco::islp_eval_type_INTERNAL:



Public Attributes

- std::vector< std::vector < interval > > * islp_data
- std::vector< std::vector < interval > > * islp_cache
- std::vector< interval > * islp_vec
- const model * mod
- interval mult
- interval mult_trans
- unsigned int child_n

10.188.1 Detailed Description

This class is the visitor data type for the islp_eval evaluator.

Definition at line 34 of file islp_evaluator.h.

10.188.2 Member Data Documentation

10.188.2.1 unsigned int islp_eval_type_INTERNAL::child_n

children counter

Definition at line 46 of file islp_evaluator.h.

10.188.2.2 std::vector< std::vector< interval > > * islp_eval_type_INTERNAL::islp_cache

slope cache

Definition at line 37 of file islp_evaluator.h.

10.188.2.3 std::vector< std::vector< interval > > * islp_eval_type_INTERNAL::islp_data

partial slope data

Definition at line 36 of file islp_evaluator.h.

10.188.2.4 std::vector< interval > * islp_eval_type_INTERNAL::islp_vec

result vector

Definition at line 42 of file islp_evaluator.h.

10.188.2.5 const model * islp_eval_type_INTERNAL::mod

the DAG

Definition at line 43 of file islp_evaluator.h.

10.188.2.6 interval islp_eval_type_INTERNAL::mult

the current value on the path

Definition at line 44 of file islp_evaluator.h.

10.188.2.7 interval islp_eval_type_INTERNAL::mult_trans

transfer variable for mult

Definition at line 45 of file islp_evaluator.h.

The documentation for this struct was generated from the following file:

- islp_evaluator_tmpl.h

10.189 islp_eval_type_INTERNAL Struct Reference

Visitor data for islp_eval.

`#include <islp_evaluator_tmpl.h>`

Public Attributes

- std::vector< std::vector < interval > > * islp_data
- std::vector< std::vector < interval > > * islp_cache
- std::vector< interval > * islp_vec
- const model * mod

- interval mult
- interval mult_trans
- unsigned int child_n

10.189.1 Detailed Description

This class is the visitor data type for the islp_eval evaluator.

10.189.2 Member Data Documentation

10.189.2.1 unsigned int islp_eval_type_INTERNAL::child_n

children counter

Definition at line 45 of file islp_evaluator_tmpl.h.

10.189.2.2 std::vector<std::vector<interval> >* islp_eval_type_INTERNAL::islp_cache

slope cache

Definition at line 36 of file islp_evaluator_tmpl.h.

10.189.2.3 std::vector<std::vector<interval> >* islp_eval_type_INTERNAL::islp_data

partial slope data

Definition at line 35 of file islp_evaluator_tmpl.h.

10.189.2.4 std::vector<interval>* islp_eval_type_INTERNAL::islp_vec

result vector

Definition at line 41 of file islp_evaluator_tmpl.h.

10.189.2.5 const model* islp_eval_type_INTERNAL::mod

the DAG

Definition at line 42 of file islp_evaluator_tmpl.h.

10.189.2.6 interval islp_eval_type_INTERNAL::mult

the current value on the path

Definition at line 43 of file islp_evaluator_tmpl.h.

10.189.2.7 interval islp_eval_type_INTERNAL::mult_trans

transfer variable for mult

Definition at line 44 of file islp_evaluator_tmpl.h.

The documentation for this struct was generated from the following file:

- islp_evaluator_tmpl.h

10.190 coco::ithirdderBackwardEvaluator Class Reference

`#include <ithirdder_evaluator.h>`

Inheritance diagram for coco::ithirdderBackwardEvaluator:

Collaboration diagram for coco::ithirdderBackwardEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- ithirdderBackwardEvaluator (std::vector< std::vector< interval > > *__der_data, std::vector< std::vector< interval > > *__hessv_data, std::vector< std::vector< interval > > *__hessw_data, std::vector< std::vector< interval > > *__ithirdder_data, variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > *__d, std::vector< std::vector< interval > > *__hv, std::vector< std::vector< interval > > *__hw, std::vector< std::vector< interval > > *__t, std::vector< interval > *__grad, std::vector< interval > *__hessv, std::vector< interval > *__hessw, std::vector< interval > *__ithirdder)
- ithirdderBackwardEvaluator (const ithirdderBackwardEvaluator &__he)
- ~ithirdderBackwardEvaluator ()
- void newPoint (std::vector< std::vector< interval > > *__der_data, std::vector< std::vector< interval > > *__hessv_data, std::vector< std::vector< interval > > *__hessw_data, std::vector< std::vector< interval > > *__ithirdder_data, const variable_indicator &__v)
- void newResult (std::vector< interval > *__grad, std::vector< interval > *__hessv, std::vector< interval > *__hessw, std::vector< interval > *__ithirdder)
- void set_mult (interval scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)

- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.190.1 Member Typedef Documentation

10.190.1.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.190.1.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.190.1.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.190.2 Constructor & Destructor Documentation

10.190.2.1 coco::ithirdderBackwardEvaluator::ithirdderBackwardEvaluator (std::vector< std::vector< interval > > * *__der_data*, std::vector< std::vector< interval > > * *__hessv_data*, std::vector< std::vector< interval > > * *__hessw_data*, std::vector< std::vector< interval > > * *__ithirdder_data*, variable_indicator & *__v*, const model & *__m*, std::vector< std::vector< interval > > * *__d*, std::vector< std::vector< interval > > * *__hv*, std::vector< std::vector< interval > > * *__hw*, std::vector< std::vector< interval > > * *__t*, std::vector< interval > * *__grad*, std::vector< interval > * *__hessv*, std::vector< interval > * *__hessw*, std::vector< interval > * *__ithirdder*) [inline]

Definition at line 1634 of file ithirdder_evaluator.h.

10.190.2.2 coco::ithirdderBackwardEvaluator::ithirdderBackwardEvaluator (const ithirdderBackwardEvaluator & *__he*) [inline]

Definition at line 1682 of file ithirdder_evaluator.h.

10.190.2.3 coco::ithirdderBackwardEvaluator::~ithirdderBackwardEvaluator ( ) [inline]

Definition at line 1684 of file ithirdder_evaluator.h.

10.190.3 Member Function Documentation

10.190.3.1 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The `__data` parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.190.3.2 int coco::ithirdderBackwardEvaluator::calculate (const expression_node & *__data*) [inline]

Definition at line 1735 of file ithirdder_evaluator.h.

10.190.3.3 bool coco::ithirdderBackwardEvaluator::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_backward_evaluator_base< ithirdderBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1976 of file ithirdder_evaluator.h.

10.190.3.4 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.190.3.5 void coco::ithirdderBackwardEvaluator::cleanup (const expression_node & *__data*) [inline]

Definition at line 1864 of file ithirdder_evaluator.h.

10.190.3.6 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.190.3.7 void coco::ithirdderBackwardEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_backward_evaluator_base< ithirdderBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1724 of file ithirdder_evaluator.h.

10.190.3.8 bool coco::ithirdderBackwardEvaluator::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_backward_evaluator_base< ithirdderBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1621 of file ithirdder_evaluator.h.

10.190.3.9 void coco::ithirdderBackwardEvaluator::newPoint (std::vector< std::vector< interval > > * __der_data, std::vector< std::vector< interval > > * __hessv_data, std::vector< std::vector< interval > > * __hessw_data, std::vector< std::vector< interval > > * __ithirdder_data, const variable_indicator & __v) [inline]

Definition at line 1686 of file ithirdder_evaluator.h.

10.190.3.10 void coco::ithirdderBackwardEvaluator::newResult (std::vector< interval > * __grad, std::vector< interval > * __hessv, std::vector< interval > * __hessw, std::vector< interval > * __ithirdder) [inline]

Definition at line 1702 of file ithirdder_evaluator.h.

10.190.3.11 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.190.3.12 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.190.3.13 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.190.3.14 void coco::ithirdderBackwardEvaluator::set_mult (interval scal) [inline]

Definition at line 1711 of file ithirdder_evaluator.h.

10.190.3.15 expression_const_walker coco::ithirdderBackwardEvaluator::short_cut_to (const expression_node & __data) [inline]

Definition at line 1720 of file ithirdder_evaluator.h.

10.190.3.16 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.190.3.17 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.190.3.18 int coco::ithirdderBackwardEvaluator::update (const bool & __rval) [inline]

Definition at line 1902 of file ithirdder_evaluator.h.

10.190.3.19 int coco::ithirdderBackwardEvaluator::update (const expression_node & *__data*, const bool & *__rval*) [inline]

Definition at line 1912 of file ithirdder_evaluator.h.

10.190.3.20 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 892 of file search_graph.cc.

10.190.3.21 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.190.3.22 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.190.3.23 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

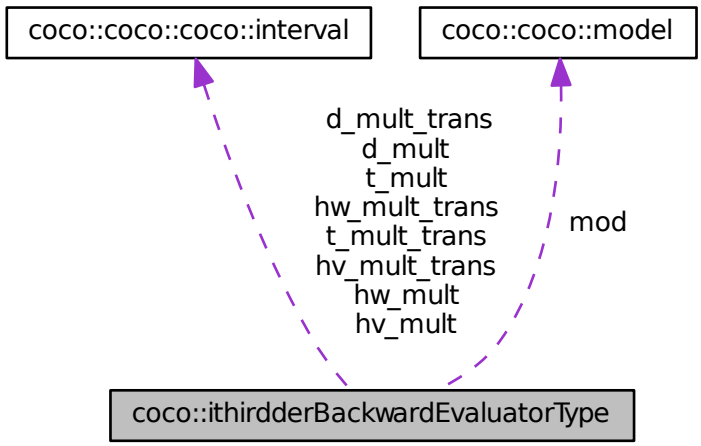
The documentation for this class was generated from the following file:

- ithirdder_evaluator.h

10.191 coco::ithirdderBackwardEvaluatorType Struct Reference

`#include <ithirdder_evaluator.h>`

Collaboration diagram for coco::ithirdderBackwardEvaluatorType:



Public Attributes

- std::vector< std::vector < interval > > * d_data
- std::vector< std::vector < interval > > * hv_data
- std::vector< std::vector < interval > > * hw_data
- std::vector< std::vector < interval > > * t_data
- std::vector< std::vector < interval > > * d_cache
- std::vector< std::vector < interval > > * hv_cache
- std::vector< std::vector < interval > > * hw_cache
- std::vector< std::vector < interval > > * t_cache
- std::vector< interval > * grad_vec
- std::vector< interval > * hessv_vec
- std::vector< interval > * hessw_vec
- std::vector< interval > * ithirdder_vec
- bool calc_grad
- bool calc_hessv
- bool calc_hessw
- const model * mod
- interval d_mult
- interval d_mult_trans
- interval hv_mult
- interval hv_mult_trans
- interval hw_mult
- interval hw_mult_trans
- interval t_mult
- interval t_mult_trans
- bool is_linear
- unsigned int child_n

10.191.1 Member Data Documentation

10.191.1.1 bool coco::ithirdderBackwardEvaluatorType::calc_grad

Definition at line 1596 of file ithirdder_evaluator.h.

10.191.1.2 bool coco::ithirdderBackwardEvaluatorType::calc_hessv

Definition at line 1597 of file ithirdder_evaluator.h.

10.191.1.3 bool coco::ithirdderBackwardEvaluatorType::calc_hessw

Definition at line 1598 of file ithirdder_evaluator.h.

10.191.1.4 unsigned int coco::ithirdderBackwardEvaluatorType::child_n

Definition at line 1609 of file ithirdder_evaluator.h.

10.191.1.5 std::vector<std::vector<interval> >* coco::ithirdderBackwardEvaluatorType::d_cache

Definition at line 1588 of file ithirdder_evaluator.h.

10.191.1.6 std::vector<std::vector<interval> >* coco::ithirdderBackwardEvaluatorType::d_data

Definition at line 1584 of file ithirdder_evaluator.h.

10.191.1.7 interval coco::ithirdderBackwardEvaluatorType::d_mult

Definition at line 1600 of file ithirdder_evaluator.h.

10.191.1.8 interval coco::ithirdderBackwardEvaluatorType::d_mult_trans

Definition at line 1601 of file ithirdder_evaluator.h.

10.191.1.9 std::vector<interval>* coco::ithirdderBackwardEvaluatorType::grad_vec

Definition at line 1592 of file ithirdder_evaluator.h.

10.191.1.10 std::vector<interval>* coco::ithirdderBackwardEvaluatorType::hessv_vec

Definition at line 1593 of file ithirdder_evaluator.h.

10.191.1.11 std::vector<interval>* coco::ithirdderBackwardEvaluatorType::hessw_vec

Definition at line 1594 of file ithirdder_evaluator.h.

10.191.1.12 std::vector<std::vector<interval> >* coco::ithirdderBackwardEvaluatorType::hv_cache

Definition at line 1589 of file ithirdder_evaluator.h.

10.191.1.13 std::vector<std::vector<interval> >* coco::ithirdderBackwardEvaluatorType::hv_data

Definition at line 1585 of file ithirdder_evaluator.h.

10.191.1.14 interval coco::ithirdderBackwardEvaluatorType::hv_mult

Definition at line 1602 of file ithirdder_evaluator.h.

10.191.1.15 interval coco::ithirdderBackwardEvaluatorType::hv_mult_trans

Definition at line 1603 of file ithirdder_evaluator.h.

10.191.1.16 std::vector<std::vector<interval> >* coco::ithirdderBackwardEvaluatorType::hw_cache

Definition at line 1590 of file ithirdder_evaluator.h.

10.191.1.17 std::vector<std::vector<interval> >* coco::ithirdderBackwardEvaluatorType::hw_data

Definition at line 1586 of file ithirdder_evaluator.h.

10.191.1.18 interval coco::ithirdderBackwardEvaluatorType::hw_mult

Definition at line 1604 of file ithirdder_evaluator.h.

10.191.1.19 interval coco::ithirdderBackwardEvaluatorType::hw_mult_trans

Definition at line 1605 of file ithirdder_evaluator.h.

10.191.1.20 bool coco::ithirdderBackwardEvaluatorType::is_linear

Definition at line 1608 of file ithirdder_evaluator.h.

10.191.1.21 std::vector<interval>* coco::ithirdderBackwardEvaluatorType::ithirdder_vec

Definition at line 1595 of file ithirdder_evaluator.h.

10.191.1.22 const model* coco::ithirdderBackwardEvaluatorType::mod

Definition at line 1599 of file ithirdder_evaluator.h.

10.191.1.23 std::vector<std::vector<interval> >* coco::ithirdderBackwardEvaluatorType::t_cache

Definition at line 1591 of file ithirdder_evaluator.h.

10.191.1.24 std::vector<std::vector<interval> >* coco::ithirdderBackwardEvaluatorType::t_data

Definition at line 1587 of file ithirdder_evaluator.h.

10.191.1.25 interval coco::ithirdderBackwardEvaluatorType::t_mult

Definition at line 1606 of file ithirdder_evaluator.h.

10.191.1.26 interval coco::ithirdderBackwardEvaluatorType::t_mult_trans

Definition at line 1607 of file ithirdder_evaluator.h.

The documentation for this struct was generated from the following file:

- ithirdder_evaluator.h

10.192 coco::ithirdderForwardEvaluator Class Reference

`#include <ithirdder_evaluator.h>`

Inheritance diagram for coco::ithirdderForwardEvaluator:

Collaboration diagram for coco::ithirdderForwardEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- ithirdderForwardEvaluator (const std::vector< ihessNumber > &__x, const std::vector< interval > &__rg, const variable_indicator &__v, const model &__m, std::vector< std::vector< interval > > &__d, std::vector< std::vector< interval > > &__hv, std::vector< std::vector< interval > > &__hw, std::vector< std::vector< interval > > &__t, std::vector< ihessNumber > *__c, bool inters=true, const std::vector< interval > *_center=NULL)
- ithirdderForwardEvaluator (const ithirdderForwardEvaluator &__x)
- ~ithirdderForwardEvaluator ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void newPoint (const std::vector< ihessNumber > &__x, const variable_indicator &__v)
- void newRange (const std::vector< interval > &__rg, const variable_indicator &__v)
- void newCenter (const std::vector< interval > *_center)
- void changeIntersectBehaviour (bool intersect)
- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)

- int update (const ihessNumber &__rval)
- int update (const expression_node &__data, const ithirdderForwardEvaluatorReturnValue &__rval)
- ithirdderForwardEvaluatorReturnValue calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.192.1 Member Typedef Documentation

10.192.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.192.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.192.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.192.2 Constructor & Destructor Documentation

10.192.2.1 coco::ithirdderForwardEvaluator::ithirdderForwardEvaluator (const std::vector< ihessNumber > & __x, const std::vector< interval > & __rg, const variable_indicator & __v, const model & __m, std::vector< std::vector< interval > > & __d, std::vector< std::vector< interval > > & __hv, std::vector< std::vector< interval > > & __hw, std::vector< std::vector< interval > > & __t, std::vector< ihessNumber > * __c, bool *inters* = true, const std::vector< interval > * *_center* = NULL) [inline]

Definition at line 268 of file ithirdder_evaluator.h.

10.192.2.2 coco::ithirdderForwardEvaluator::ithirdderForwardEvaluator (const ithirdderForwardEvaluator & __x) [inline]

Definition at line 300 of file ithirdder_evaluator.h.

10.192.2.3 coco::ithirdderForwardEvaluator::~ithirdderForwardEvaluator ( ) [inline]

Definition at line 304 of file ithirdder_evaluator.h.

10.192.3 Member Function Documentation

10.192.3.1 void coco::ithirdderForwardEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 429 of file ithirdder_evaluator.h.

10.192.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.192.3.3 ithirdderForwardEvaluatorReturnValue coco::ithirdderForwardEvaluator::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< ithirdderForwardEvaluatorType, expression_node, ithirdderForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 1573 of file ithirdder_evaluator.h.

10.192.3.4 void coco::ithirdderForwardEvaluator::changeIntersectBehaviour (bool *intersect*) [inline]

Definition at line 338 of file ithirdder_evaluator.h.

10.192.3.5 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.192.3.6 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.192.3.7 void coco::ithirdderForwardEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< ithirdderForwardEvaluatorType, expression_node, ithirdderForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 343 of file ithirdder_evaluator.h.

10.192.3.8 int coco::ithirdderForwardEvaluator::initialize (const expression_node & __data) [inline]

Definition at line 351 of file ithirdder_evaluator.h.

10.192.3.9 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.192.3.10 bool coco::ithirdderForwardEvaluator::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< ithirdderForwardEvaluatorType, expression_node, ithirdderForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 195 of file ithirdder_evaluator.h.

10.192.3.11 void coco::ithirdderForwardEvaluator::newCenter (const std::vector< interval > * _center) [inline]

This method changes the center for centered forms to _center.

Definition at line 333 of file ithirdder_evaluator.h.

10.192.3.12 void coco::ithirdderForwardEvaluator::newPoint (const std::vector< ihessNumber > & __x, const variable_indicator & __v) [inline]

Definition at line 309 of file ithirdder_evaluator.h.

10.192.3.13 void coco::ithirdderForwardEvaluator::newRange (const std::vector< interval > & __rg, const variable_indicator & __v) [inline]

This method changes the node ranges to ___rg. The parameter * ___v specifies which variables have changed value since the last * evaluation.

Definition at line 326 of file ithirdder_evaluator.h.

10.192.3.14 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.192.3.15 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.192.3.16 void coco::ithirdderForwardEvaluator::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 438 of file ithirdder_evaluator.h.

10.192.3.17 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.192.3.18 expression_const_walker coco::ithirdderForwardEvaluator::short_cut_to (const expression_node & __data) [inline]

Definition at line 306 of file ithirdder_evaluator.h.

10.192.3.19 int coco::ithirdderForwardEvaluator::update (const ihessNumber & __rval) [inline]

Definition at line 445 of file ithirdder_evaluator.h.

10.192.3.20 int coco::ithirdderForwardEvaluator::update (const expression_node & __data, const ithirdderForwardEvaluatorReturnValue & __rval) [inline]

Definition at line 455 of file ithirdder_evaluator.h.

10.192.3.21 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.192.3.22 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.192.3.23 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.192.3.24 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.192.3.25 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.192.3.26 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

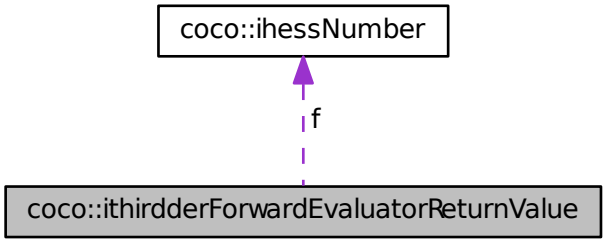
The documentation for this class was generated from the following file:

- ithirdder_evaluator.h

10.193 coco::ithirdderForwardEvaluatorReturnValue Struct Reference

`#include <ithirdder_evaluator.h>`

Collaboration diagram for coco::ithirdderForwardEvaluatorReturnValue:



Public Attributes

- ihessNumber f
- unsigned int nn
- tristate in_chn

10.193.1 Member Data Documentation

10.193.1.1 ihessNumber coco::ithirdderForwardEvaluatorReturnValue::f

Definition at line 160 of file ithirdder_evaluator.h.

10.193.1.2 tristate coco::ithirdderForwardEvaluatorReturnValue::in_chn

Definition at line 163 of file ithirdder_evaluator.h.

10.193.1.3 unsigned int coco::ithirdderForwardEvaluatorReturnValue::nn

Definition at line 162 of file ithirdder_evaluator.h.

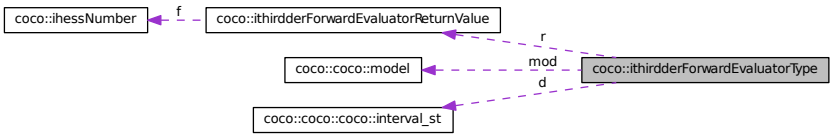
The documentation for this struct was generated from the following file:

- ithirdder_evaluator.h

10.194 coco::ithirdderForwardEvaluatorType Struct Reference

`#include <ithirdder_evaluator.h>`

Collaboration diagram for coco::ithirdderForwardEvaluatorType:



Public Attributes

- const std::vector< ihessNumber > * x
- const std::vector< interval > * range
- std::vector< ihessNumber > * f_cache
- std::vector< std::vector < interval > > * t_data
- std::vector< std::vector < interval > > * hv_data
- std::vector< std::vector < interval > > * hw_data
- std::vector< std::vector < interval > > * d_data
- std::vector< unsigned int > * t
- std::vector< bool > * b
- bool do_intersect
- ithirdderForwardEvaluatorReturnValue r
- const model * mod
- union {
  void * p
  interval_st d
  } u

- unsigned int n
- unsigned int info
- const std::vector< interval > * c

10.194.1 Member Data Documentation

10.194.1.1 std::vector<bool>* coco::ithirdderForwardEvaluatorType::b

Definition at line 176 of file ithirdder_evaluator.h.

10.194.1.2 const std::vector<interval>* coco::ithirdderForwardEvaluatorType::c

Definition at line 183 of file ithirdder_evaluator.h.

10.194.1.3 interval_st coco::ithirdderForwardEvaluatorType::d

Definition at line 180 of file ithirdder_evaluator.h.

10.194.1.4 std::vector<std::vector<interval> >* coco::ithirdderForwardEvaluatorType::d_data

Definition at line 174 of file ithirdder_evaluator.h.

10.194.1.5 bool coco::ithirdderForwardEvaluatorType::do_intersect

Definition at line 177 of file ithirdder_evaluator.h.

10.194.1.6 std::vector<ihessNumber>* coco::ithirdderForwardEvaluatorType::f_cache

Definition at line 170 of file ithirdder_evaluator.h.

10.194.1.7 std::vector<std::vector<interval> >* coco::ithirdderForwardEvaluatorType::hv_data

Definition at line 172 of file ithirdder_evaluator.h.

10.194.1.8 std::vector<std::vector<interval> >* coco::ithirdderForwardEvaluatorType::hw_data

Definition at line 173 of file ithirdder_evaluator.h.

10.194.1.9 unsigned int coco::ithirdderForwardEvaluatorType::info

Definition at line 181 of file ithirdder_evaluator.h.

10.194.1.10 const model* coco::ithirdderForwardEvaluatorType::mod

Definition at line 179 of file ithirdder_evaluator.h.

10.194.1.11 unsigned int coco::ithirdderForwardEvaluatorType::n

Definition at line 181 of file ithirdder_evaluator.h.

10.194.1.12 void* coco::ithirdderForwardEvaluatorType::p

Definition at line 180 of file ithirdder_evaluator.h.

10.194.1.13 ithirdderForwardEvaluatorReturnValue coco::ithirdderForwardEvaluatorType::r

Definition at line 178 of file ithirdder_evaluator.h.

10.194.1.14 const std::vector<interval>* coco::ithirdderForwardEvaluatorType::range

the ranges of all nodes

Definition at line 169 of file ithirdder_evaluator.h.

10.194.1.15 std::vector<unsigned int>* coco::ithirdderForwardEvaluatorType::t

Definition at line 175 of file ithirdder_evaluator.h.

10.194.1.16 std::vector<std::vector<interval> >* coco::ithirdderForwardEvaluatorType::t_data

Definition at line 171 of file ithirdder_evaluator.h.

10.194.1.17 union { ... } coco::ithirdderForwardEvaluatorType::u

10.194.1.18 const std::vector<ihessNumber>* coco::ithirdderForwardEvaluatorType::x

Definition at line 168 of file ithirdder_evaluator.h.

The documentation for this struct was generated from the following file:

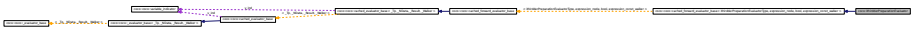
- ithirdder_evaluator.h

10.195 coco::ithirdderPreparationEvaluator Class Reference

`#include <ithirdder_evaluator.h>`

Inheritance diagram for coco::ithirdderPreparationEvaluator:

Collaboration diagram for coco::ithirdderPreparationEvaluator:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- ithirdderPreparationEvaluator (std::vector< std::vector< interval > > &__d, std::vector< std::vector< interval > > &__hv, std::vector< std::vector< interval > > &__hw, std::vector< std::vector< interval > > &__t, unsigned int _num_of_nodes)
- ithirdderPreparationEvaluator (const ithirdderPreparationEvaluator &__x)
- ~ithirdderPreparationEvaluator ()
- void initialize ()
- bool is_cached (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- int update (bool __rval)
- int update (const expression_node &__data, bool __rval)
- bool calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

10.195.1 Member Typedef Documentation

10.195.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.195.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.195.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.195.2 Constructor & Destructor Documentation

10.195.2.1 coco::ithirdderPreparationEvaluator::ithirdderPreparationEvaluator (std::vector< std::vector< interval > > & __d, std::vector< std::vector< interval > > & __hv, std::vector< std::vector< interval > > & __hw, std::vector< std::vector< interval > > & __t, unsigned int _num_of_nodes) [inline]

Definition at line 89 of file ithirdder_evaluator.h.

10.195.2.2 coco::ithirdderPreparationEvaluator::ithirdderPreparationEvaluator (const ithirdderPreparationEvaluator & __x) [inline]

Definition at line 109 of file ithirdder_evaluator.h.

10.195.2.3 coco::ithirdderPreparationEvaluator::~ithirdderPreparationEvaluator ( ) [inline]

Definition at line 115 of file ithirdder_evaluator.h.

10.195.3 Member Function Documentation

10.195.3.1 void coco::ithirdderPreparationEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 140 of file ithirdder_evaluator.h.

10.195.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.195.3.3 bool coco::ithirdderPreparationEvaluator::calculate_value (bool eval_all) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< ithirdderPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 152 of file ithirdder_evaluator.h.

10.195.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.195.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.195.3.6 void coco::ithirdderPreparationEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< ithirdderPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 117 of file ithirdder_evaluator.h.

10.195.3.7 int coco::ithirdderPreparationEvaluator::initialize (const expression_node & __data) [inline]

Definition at line 126 of file ithirdder_evaluator.h.

10.195.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.195.3.9 bool coco::ithirdderPreparationEvaluator::is_cached (const expression_node & __data) [inline]

Definition at line 119 of file ithirdder_evaluator.h.

10.195.3.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & __data) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.195.3.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.195.3.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.195.3.13 void coco::ithirdderPreparationEvaluator::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 124 of file ithirdder_evaluator.h.

10.195.3.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.195.3.15 int coco::ithirdderPreparationEvaluator::update (bool __rval) [inline]

Definition at line 142 of file ithirdder_evaluator.h.

10.195.3.16 int coco::ithirdderPreparationEvaluator::update (const expression_node & __data, bool __rval) [inline]

Definition at line 144 of file ithirdder_evaluator.h.

10.195.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.195.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.195.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter false.

Definition at line 763 of file search_graph.cc.

10.195.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.195.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.195.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter true.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- ithirdder_evaluator.h

10.196 coco::ithirdderPreparationEvaluatorType Struct Reference

#include <ithirdder_evaluator.h>

Public Attributes

- std::vector< std::vector < interval > > * d
- std::vector< std::vector < interval > > * hv
- std::vector< std::vector < interval > > * hw
- std::vector< std::vector < interval > > * t

10.196.1 Member Data Documentation

10.196.1.1 std::vector<std::vector<interval> >* coco::ithirdderPreparationEvaluatorType::d

Definition at line 75 of file ithirdder_evaluator.h.

10.196.1.2 std::vector<std::vector<interval> >* coco::ithirdderPreparationEvaluatorType::hv

Definition at line 76 of file ithirdder_evaluator.h.

10.196.1.3 std::vector<std::vector<interval> >* coco::ithirdderPreparationEvaluatorType::hw

Definition at line 77 of file ithirdder_evaluator.h.

10.196.1.4 std::vector<std::vector<interval> >* coco::ithirdderPreparationEvaluatorType::t

Definition at line 78 of file ithirdder_evaluator.h.

The documentation for this struct was generated from the following file:

- ithirdder_evaluator.h

10.197 coco::model::lincoeff_visitor Class Reference

`#include <model.hpp>`

Inheritance diagram for coco::model::lincoeff_visitor:

Collaboration diagram for coco::model::lincoeff_visitor:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- lincoeff_visitor (const std::vector< interval > *_rg, vmtl::sparse_vector< double > *_cf, double *_cs, const model *_m)
- lincoeff_visitor (const lincoeff_visitor &__x)
- ~lincoeff_visitor ()
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

- bool is_cached (const expression_node &__data)
- virtual const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- void initialize (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int calculate (const expression_node &__data)
- int update (const expression_node &__data, const lincoeff_visitor_ret &__rval)
- int update (const lincoeff_visitor_ret &__rval)
- lincoeff_visitor_ret calculate_value (bool eval_all)

10.197.1 Detailed Description

This cached backward evaluator is used to collect the coefficients of a linear node on a DAG.

10.197.2 Member Typedef Documentation

10.197.2.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.197.2.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.197.2.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.197.3 Constructor & Destructor Documentation

10.197.3.1 coco::model::lincoeff_visitor::lincoeff_visitor (const std::vector< interval > * _rg, vmtl::sparse_vector< double > * _cf, double * _cs, const model * _m) [inline]

Constructor, which initializes the node ranges `ranges` with `_rg`, the results `coeffs` and `constant` with `_cf` and `_cs`, respectively, and the model `mod` with `_m`.

Definition at line 2458 of file model.hpp.

10.197.3.2 coco::model::lincoeff_visitor::lincoeff_visitor (const lincoeff_visitor & __x) [inline]

Standard Copy Constructor

Definition at line 2478 of file model.hpp.

10.197.3.3 coco::model::lincoeff_visitor::~lincoeff_visitor ( ) [inline]

Standard Destructor

Definition at line 2481 of file model.hpp.

10.197.4 Member Function Documentation

10.197.4.1 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The `__data` parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.197.4.2 int coco::model::lincoeff_visitor::calculate (const expression_node & __data) [inline]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Definition at line 2540 of file model.hpp.

10.197.4.3 lincoeff_visitor_ret coco::model::lincoeff_visitor::calculate_value (bool *eval_all*) [inline, virtual]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Reimplemented from coco::coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 3196 of file model.hpp.

10.197.4.4 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.197.4.5 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.197.4.6 void coco::model::lincoeff_visitor::initialize ( ) [inline, virtual]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Reimplemented from coco::coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 2497 of file model.hpp.

10.197.4.7 void coco::model::lincoeff_visitor::initialize (const expression_node & *__data*) [inline]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Definition at line 2508 of file model.hpp.

10.197.4.8 virtual bool coco::coco::cached_backward_evaluator_base::is_cached (const node_data_type & __data) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 906 of file search_graph.cc.

10.197.4.9 bool coco::model::lincoeff_visitor::is_cached (const expression_node & __data) [inline]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Definition at line 2486 of file model.hpp.

10.197.4.10 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.197.4.11 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.197.4.12 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.197.4.13 void coco::model::lincoeff_visitor::retrieve_from_cache (const expression_node & __data) [inline]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Definition at line 2519 of file model.hpp.

10.197.4.14 virtual const_walker coco::model::lincoeff_visitor::short_cut_to (const expression_node & __data) [inline, virtual]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Definition at line 2492 of file model.hpp.

10.197.4.15 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.197.4.16 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.197.4.17 int coco::model::lincoeff_visitor::update (const expression_node & __data, const lincoeff_visitor_ret & __rval) [inline]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Definition at line 2651 of file model.hpp.

10.197.4.18 int coco::model::lincoeff_visitor::update (const lincoeff_visitor_ret & __rval) [inline]

This is a method as needed by a cached backward evaluator.

See also

cached_backward_evaluator_base.

Definition at line 3188 of file model.hpp.

10.197.4.19 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 892 of file search_graph.cc.

10.197.4.20 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.197.4.21 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.197.4.22 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

The documentation for this class was generated from the following file:

- model.hpp

10.198 coco::model::lincoeff_visitor_ret Struct Reference

`#include <model.hpp>`

Public Attributes

- bool was_lin
- bool was_const
- int node_num
- double const_trans
- double help

10.198.1 Detailed Description

This is the return type of the model::lincoeff_visitor visitor.

10.198.2 Member Data Documentation

10.198.2.1 double coco::model::lincoeff_visitor_ret::const_trans

Definition at line 2423 of file model.hpp.

10.198.2.2 double coco::model::lincoeff_visitor_ret::help

Definition at line 2424 of file model.hpp.

10.198.2.3 int coco::model::lincoeff_visitor_ret::node_num

Definition at line 2422 of file model.hpp.

10.198.2.4 bool coco::model::lincoeff_visitor_ret::was_const

Definition at line 2421 of file model.hpp.

10.198.2.5 bool coco::model::lincoeff_visitor_ret::was_lin

Definition at line 2420 of file model.hpp.

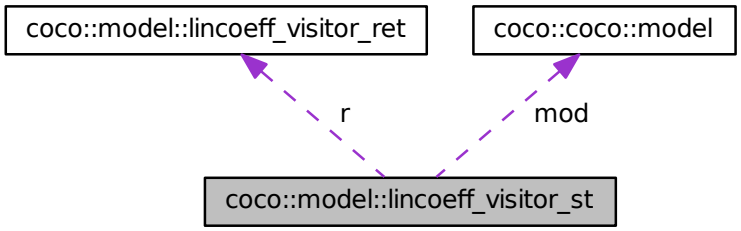
The documentation for this struct was generated from the following file:

- model.hpp

10.199 coco::model::lincoeff_visitor_st Struct Reference

`#include <model.hpp>`

Collaboration diagram for coco::model::lincoeff_visitor_st:



Public Attributes

- const model * mod
- const std::vector< interval > * ranges
- vmtl::sparse_vector< double > * coeffs
- double * constant
- double old_trans
- double trans
- double mm_help
- lincoeff_visitor_ret r
- int old_nlin
- int in_nlin
- unsigned int n

10.199.1 Detailed Description

This is the visitor data of the model::lincoeff_visitor visitor.

10.199.2 Member Data Documentation

10.199.2.1 vmtl::sparse_vector<double>* coco::model::lincoeff_visitor_st::coeffs

the coefficents of the linear function

Definition at line 2432 of file model.hpp.

10.199.2.2 double* coco::model::lincoeff_visitor_st::constant

the constant of the linear function

Definition at line 2433 of file model.hpp.

10.199.2.3 int coco::model::lincoeff_visitor_st::in_nlin

marker for non-linear parts

Definition at line 2439 of file model.hpp.

10.199.2.4 double coco::model::lincoeff_visitor_st::mm_help

helper for max, min

Definition at line 2436 of file model.hpp.

10.199.2.5 const model* coco::model::lincoeff_visitor_st::mod

the DAG

Definition at line 2430 of file model.hpp.

10.199.2.6 unsigned int coco::model::lincoeff_visitor_st::n

children counter

Definition at line 2440 of file model.hpp.

10.199.2.7 int coco::model::lincoeff_visitor_st::old_nlin

old marker for non-linear parts

Definition at line 2438 of file model.hpp.

10.199.2.8 double coco::model::lincoeff_visitor_st::old_trans

the old scaling factor

Definition at line 2434 of file model.hpp.

10.199.2.9 lincoeff_visitor_ret coco::model::lincoeff_visitor_st::r

the return value

Definition at line 2437 of file model.hpp.

10.199.2.10 const std::vector<interval>* coco::model::lincoeff_visitor_st::ranges

the ranges of all nodes

Definition at line 2431 of file model.hpp.

10.199.2.11 double coco::model::lincoeff_visitor_st::trans

the scaling factor

Definition at line 2435 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

10.200 coco::locopt_ret_record Class Reference

`#include <locopt_ret.h>`

Public Member Functions

- locopt_ret_record ()
- locopt_ret_record (const termination_reason &t)
- locopt_ret_record (const std::string &t_msg, int t_code)
- locopt_ret_record (const locopt_ret_record &r)
- locopt_ret_record & operator= (const locopt_ret_record &r)
- const termination_reason & get_termr () const

- const std::string & get_message () const
- int get_code () const
- double get_weight () const

10.200.1 Constructor & Destructor Documentation

10.200.1.1 coco::locopt_ret_record::locopt_ret_record ( ) [inline]

Constructors

Definition at line 123 of file locopt_ret.h.

10.200.1.2 coco::locopt_ret_record::locopt_ret_record (const termination_reason & *t*) [inline]

Definition at line 126 of file locopt_ret.h.

10.200.1.3 coco::locopt_ret_record::locopt_ret_record (const std::string & *t_msg*, int *t_code*) [inline]

Definition at line 129 of file locopt_ret.h.

10.200.1.4 coco::locopt_ret_record::locopt_ret_record (const locopt_ret_record & *r*) [inline]

Copy constructor

Definition at line 133 of file locopt_ret.h.

10.200.2 Member Function Documentation

10.200.2.1 int coco::locopt_ret_record::get_code ( ) const [inline]

This method return the IE return code.

Definition at line 154 of file locopt_ret.h.

10.200.2.2 const std::string& coco::locopt_ret_record::get_message ( ) const [inline]

This method returns the termination message.

Definition at line 151 of file locopt_ret.h.

10.200.2.3 const termination_reason& coco::locopt_ret_record::get_termr ( ) const [inline]

This method returns the termination reason.

Definition at line 148 of file locopt_ret.h.

10.200.2.4 double coco::locopt_ret_record::get_weight ( ) const [inline]

This method return the IE return weight.

Definition at line 157 of file locopt_ret.h.

10.200.2.5 locopt_ret_record& coco::locopt_ret_record::operator= (const locopt_ret_record & *r*) [inline]

Assignment operator

Definition at line 137 of file locopt_ret.h.

The documentation for this class was generated from the following file:

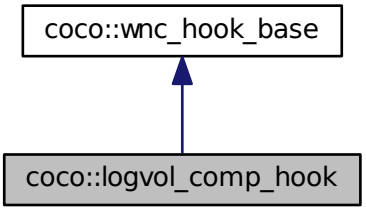
- locopt_ret.h

10.201 coco::logvol_comp_hook Class Reference

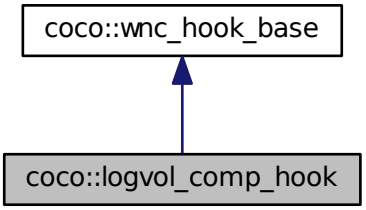
The log-volume computation hook (work node computation hook)

`#include <logvol_hook.h>`

Inheritance diagram for coco::logvol_comp_hook:



Collaboration diagram for coco::logvol_comp_hook:



Public Member Functions

- logvol_comp_hook ()
- virtual ~logvol_comp_hook ()
- logvol_comp_hook * new_copy () const
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *a=NULL, std::list< certificate > *b=NULL) const
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)
- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.201.1 Detailed Description

This class is a work node computation hook (see work_node_comp_hook), which calculates the log-volume of the work node (from the bound constraints) and keeps it in column "logvol".

10.201.2 Constructor & Destructor Documentation

10.201.2.1 coco::logvol_comp_hook::logvol_comp_hook ( ) [inline]

Standard Constructor

Definition at line 46 of file logvol_hook.h.

10.201.2.2 virtual coco::logvol_comp_hook::~logvol_comp_hook ( ) [inline, virtual]

Standard Destructor

Definition at line 49 of file logvol_hook.h.

10.201.3 Member Function Documentation

10.201.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected, inherited]

Remove all columns, whose identifiers are stored in __managed_cols from table stable.

Definition at line 34 of file comp_hook.cc.

10.201.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 238 of file comp_hook.h.

10.201.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 135 of file comp_hook.h.

10.201.3.4 bool coco::logvol_comp_hook::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon unregistering this hook, destroy the column "logvol" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 69 of file logvol_hook.h.

10.201.3.5 bool coco::logvol_comp_hook::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the column "logvol" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 64 of file logvol_hook.h.

10.201.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline, inherited]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.201.3.7 logvol_comp_hook* coco::logvol_comp_hook::new_copy ( ) const [inline, virtual]

Clone Operation

Implements coco::wnc_hook_base.

Definition at line 52 of file logvol_hook.h.

10.201.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_id of work_node wn.

Definition at line 43 of file comp_hook.cc.

10.201.3.9 void coco::logvol_comp_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *a* = NULL, std::list< certificate > * *b* = NULL) const [inline, virtual]

The evaluation operator of this computation hook. It stores the log-volume of the work_node `wn` in dbt_row `dbr`.

Implements coco::wnc_hook_base.

Definition at line 56 of file logvol_hook.h.

10.201.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_relation of work_node `wn`.

Definition at line 46 of file comp_hook.cc.

The documentation for this class was generated from the following file:

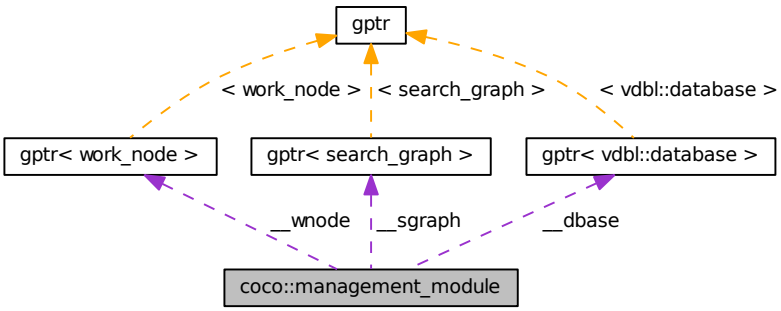
- logvol_hook.h

10.202 coco::management_module Class Reference

Management module base class.

`#include <mgmt_module.h>`

Collaboration diagram for coco::management_module:



Public Member Functions

- management_module (const std::string &__n, gptr< work_node > &wnode, search_focus &sfoc, const search_inspector *sinsp, gptr< search_graph > &sgraph, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, search_focus &sfoc, const search_inspector *sinsp, gptr< search_graph > &sgraph)

- management_module (const std::string &__n, gptr< work_node > &wnode, search_focus &sfoc, const search_inspector *sinsp, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, const search_inspector *sinsp, gptr< search_graph > &sgraph, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, search_focus &sfoc, const search_inspector *sinsp, gptr< search_graph > &sgraph, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, search_focus &sfoc, const search_inspector *sinsp)
- management_module (const std::string &__n, gptr< work_node > &wnode, const search_inspector *sinsp, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, const search_inspector *sinsp, gptr< search_graph > &sgraph)
- management_module (const std::string &__n, search_focus &sfoc, const search_inspector *sinsp, gptr< search_graph > &sgraph)
- management_module (const std::string &__n, search_focus &sfoc, const search_inspector *sinsp, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, const search_inspector *sinsp, gptr< search_graph > &sgraph, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, const search_inspector *sinsp)
- management_module (const std::string &__n, search_focus &sfoc, const search_inspector *sinsp)
- management_module (const std::string &__n, const search_inspector *sinsp, gptr< search_graph > &sgraph)
- management_module (const std::string &__n, const search_inspector *sinsp, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, search_focus &sfoc, gptr< search_graph > &sgraph, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, search_focus &sfoc, gptr< search_graph > &sgraph)
- management_module (const std::string &__n, gptr< work_node > &wnode, search_focus &sfoc, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, gptr< search_graph > &sgraph, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, search_focus &sfoc, gptr< search_graph > &sgraph, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, search_focus &sfoc)
- management_module (const std::string &__n, gptr< work_node > &wnode, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode, gptr< search_graph > &sgraph)
- management_module (const std::string &__n, search_focus &sfoc, gptr< search_graph > &sgraph)
- management_module (const std::string &__n, search_focus &sfoc, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< search_graph > &sgraph, gptr< vdbl::database > &dbase)
- management_module (const std::string &__n, gptr< work_node > &wnode)
- management_module (const std::string &__n, search_focus &sfoc)
- management_module (const std::string &__n, const search_inspector *sinsp)
- management_module (const std::string &__n, gptr< search_graph > &sgraph)
- management_module (const std::string &__n, gptr< vdbl::database > &dbase)
- virtual ~management_module ()
- const model * get_model () const
- const std::string & get_name () const
- virtual int manage (const control_data &_c)

Protected Attributes

- std::string __name
- gptr< work_node > * __wnode
- search_focus * __sfocus
- const search_inspector * __sinsp
- gptr< vdbl::database > * __dbase
- gptr< search_graph > * __sgraph

10.202.1 Detailed Description

This is the base class of all COCONUT management modules. A management module specializes in performing changes to the internal data structures.

10.202.2 Constructor & Destructor Documentation

10.202.2.1 coco::management_module::management_module (const std::string & *__n*, gptr< work_node > & *wnode*, search_focus & *sfoc*, const search_inspector * *sinsp*, gptr< search_graph > & *sgraph*, gptr< vdbl::database > & *dbase*) [inline]

This is the most general constructor for a management module containing parameters for setting all internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus, sinsp sets __sinsp, sgraph sets __sgraph, and dbase sets __dbase.

Definition at line 94 of file mgmt_module.h.

10.202.2.2 coco::management_module::management_module (const std::string & *__n*, gptr< work_node > & *wnode*, search_focus & *sfoc*, const search_inspector * *sinsp*, gptr< search_graph > & *sgraph*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus, sinsp sets __sinsp, and sgraph sets __sgraph.

Definition at line 108 of file mgmt_module.h.

10.202.2.3 coco::management_module::management_module (const std::string & *__n*, gptr< work_node > & *wnode*, search_focus & *sfoc*, const search_inspector * *sinsp*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus, sinsp sets __sinsp, and dbase sets __dbase.

Definition at line 121 of file mgmt_module.h.

10.202.2.4 coco::management_module::management_module (const std::string & *__n*, gptr< work_node > & *wnode*, const search_inspector * *sinsp*, gptr< search_graph > & *sgraph*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sinsp sets __sinsp, sgraph sets __sgraph, and dbase sets __dbase.

Definition at line 134 of file mgmt_module.h.

10.202.2.5 coco::management_module::management_module (const std::string & *__n*, search_focus & *sfoc*, const search_inspector * *sinsp*, gptr< search_graph > & *sgraph*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, sinsp sets __sinsp, sgraph sets __sgraph, and dbase sets __dbase.

Definition at line 147 of file mgmt_module.h.

10.202.2.6 coco::management_module::management_module (const std::string & *__n*, gptr< work_node > & *wnode*, search_focus & *sfoc*, const search_inspector * *sinsp*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus, and sinsp sets __sinsp.

Definition at line 160 of file mgmt_module.h.

10.202.2.7 coco::management_module::management_module (const std::string & *__n*, gptr< work_node > & *wnode*, const search_inspector * *sinsp*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sinsp sets __sinsp, and dbase sets __dbase.

Definition at line 172 of file mgmt_module.h.

10.202.2.8 coco::management_module::management_module (const std::string & *__n*, gptr< work_node > & *wnode*, const search_inspector * *sinsp*, gptr< search_graph > & *sgraph*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sinsp sets __sinsp, and sgraph sets __sgraph.

Definition at line 184 of file mgmt_module.h.

10.202.2.9 coco::management_module::management_module (const std::string & *__n*, search_focus & *sfoc*, const search_inspector * *sinsp*, gptr< search_graph > & *sgraph*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, sinsp sets __sinsp, and sgraph sets __sgraph.

Definition at line 196 of file mgmt_module.h.

10.202.2.10 coco::management_module::management_module (const std::string & *__n*, search_focus & *sfoc*, const search_inspector * *sinsp*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, sinsp sets __sinsp, and dbase sets __dbase.

Definition at line 208 of file mgmt_module.h.

10.202.2.11 coco::management_module::management_module (const std::string & __n, const search_inspector * sinsp, gptr< search_graph > & sgraph, gptr< vdbl::database > & dbase) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sinsp sets __sinsp, sgraph sets __sgraph, and dbase sets __dbase.

Definition at line 220 of file mgmt_module.h.

10.202.2.12 coco::management_module::management_module (const std::string & __n, gptr< work_node > & wnode, const search_inspector * sinsp) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and sinsp sets __sinsp.

Definition at line 231 of file mgmt_module.h.

10.202.2.13 coco::management_module::management_module (const std::string & __n, search_focus & sfoc, const search_inspector * sinsp) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, sinsp sets __sinsp.

Definition at line 240 of file mgmt_module.h.

10.202.2.14 coco::management_module::management_module (const std::string & __n, const search_inspector * sinsp, gptr< search_graph > & sgraph) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sinsp sets __sinsp, and sgraph sets __sgraph.

Definition at line 249 of file mgmt_module.h.

10.202.2.15 coco::management_module::management_module (const std::string & __n, const search_inspector * sinsp, gptr< vdbl::database > & dbase) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sinsp sets __sinsp, and dbase sets __dbase.

Definition at line 258 of file mgmt_module.h.

10.202.2.16 coco::management_module::management_module (const std::string & __n, gptr< work_node > & wnode, search_focus & sfoc, gptr< search_graph > & sgraph, gptr< vdbl::database > & dbase) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus, sgraph sets __sgraph, and dbase sets __dbase.

Definition at line 268 of file mgmt_module.h.

10.202.2.17 coco::management_module::management_module (const std::string & __n, gptr< work_node > & *wnode*, search_focus & *sfoc*, gptr< search_graph > & *sgraph*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus, and sgraph sets __sgraph.

Definition at line 281 of file mgmt_module.h.

10.202.2.18 coco::management_module::management_module (const std::string & __n, gptr< work_node > & *wnode*, search_focus & *sfoc*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sfoc sets __sfocus, and dbase sets __dbase.

Definition at line 293 of file mgmt_module.h.

10.202.2.19 coco::management_module::management_module (const std::string & __n, gptr< work_node > & *wnode*, gptr< search_graph > & *sgraph*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, sgraph sets __sgraph, and dbase sets __dbase.

Definition at line 305 of file mgmt_module.h.

10.202.2.20 coco::management_module::management_module (const std::string & __n, search_focus & *sfoc*, gptr< search_graph > & *sgraph*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, sgraph sets __sgraph, and dbase sets __dbase.

Definition at line 317 of file mgmt_module.h.

10.202.2.21 coco::management_module::management_module (const std::string & __n, gptr< work_node > & *wnode*, search_focus & *sfoc*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and sfoc sets __sfocus.

Definition at line 328 of file mgmt_module.h.

10.202.2.22 coco::management_module::management_module (const std::string & __n, gptr< work_node > & *wnode*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and dbase sets __dbase.

Definition at line 338 of file mgmt_module.h.

10.202.2.23 coco::management_module::management_module (const std::string & __n, gptr< work_node > & *wnode*, gptr< search_graph > & *sgraph*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and sgraph sets __sgraph.

Definition at line 348 of file mgmt_module.h.

10.202.2.24 coco::management_module::management_module (const std::string & __n, search_focus & sfoc, gptr< search_graph > & sgraph) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, and sgraph sets __sgraph.

Definition at line 358 of file mgmt_module.h.

10.202.2.25 coco::management_module::management_module (const std::string & __n, search_focus & sfoc, gptr< vdbl::database > & dbase) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sfoc sets __sfocus, and dbase sets __dbase.

Definition at line 368 of file mgmt_module.h.

10.202.2.26 coco::management_module::management_module (const std::string & __n, gptr< search_graph > & sgraph, gptr< vdbl::database > & dbase) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, sgraph sets __sgraph, and dbase sets __dbase.

Definition at line 378 of file mgmt_module.h.

10.202.2.27 coco::management_module::management_module (const std::string & __n, gptr< work_node > & wnode) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, and wnode sets __wnode.

Definition at line 388 of file mgmt_module.h.

10.202.2.28 coco::management_module::management_module (const std::string & __n, search_focus & sfoc) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name, and sfoc sets __sfocus.

Definition at line 396 of file mgmt_module.h.

10.202.2.29 coco::management_module::management_module (const std::string & __n, const search_inspector * sinsp) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name and sinsp sets __sinsp.

Definition at line 404 of file mgmt_module.h.

10.202.2.30 coco::management_module::management_module (const std::string & __n, gptr< search_graph > & sgraph) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter __n sets __name and sgraph sets __sgraph.

Definition at line 412 of file mgmt_module.h.

10.202.2.31 coco::management_module::management_module (const std::string & *__n*, gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a management module containing parameters for setting some internal variables. The parameter ___n sets __name and dbase sets __dbase.

Definition at line 420 of file mgmt_module.h.

10.202.2.32 virtual coco::management_module::~management_module ( ) [inline, virtual]

Standard Destructor

Definition at line 426 of file mgmt_module.h.

10.202.3 Member Function Documentation

10.202.3.1 const model* coco::management_module::get_model ( ) const [inline]

A call to this method returns a pointer to the model of the work node.

Definition at line 429 of file mgmt_module.h.

10.202.3.2 const std::string& coco::management_module::get_name ( ) const [inline]

The get_name method returns the identifier string of the management module.

Definition at line 434 of file mgmt_module.h.

10.202.3.3 virtual int coco::management_module::manage (const control_data & *_c*) [inline, virtual]

This method is the main method of a management module. It is supposed to perform changes to some internal data structure. It returns a status value. Here 0 denotes success. All other return values signal some error or warning. Service information and parameters are provided via the control_data structure _c.

Definition at line 441 of file mgmt_module.h.

10.202.4 Member Data Documentation

10.202.4.1 gptr<vdbl::database>* coco::management_module::__dbase [protected]

This variable contains a global pointer to the search database.

Definition at line 85 of file mgmt_module.h.

10.202.4.2 std::string coco::management_module::__name [protected]

This is the identifier string for a management module.

Definition at line 77 of file mgmt_module.h.

10.202.4.3 search_focus* coco::management_module::__sfocus [protected]

This variable contains a pointer to the search focus.

Definition at line 81 of file mgmt_module.h.

10.202.4.4 gptr<search_graph>* coco::management_module::__sgraph [protected]

This variable contains a global pointer to the search graph.

Definition at line 87 of file mgmt_module.h.

10.202.4.5 const search_inspector* coco::management_module::__sinsp [protected]

This variable contains a pointer to a search inspector

Definition at line 83 of file mgmt_module.h.

10.202.4.6 gptr<work_node>* coco::management_module::__wnode [protected]

This variable contains a global pointer to the work node.

Definition at line 79 of file mgmt_module.h.

The documentation for this class was generated from the following file:

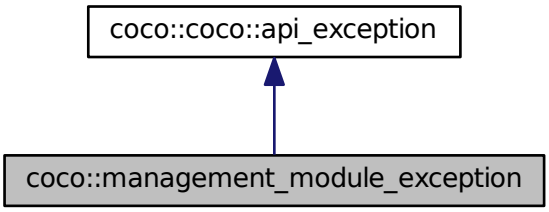
- mgmt_module.h

10.203 coco::management_module_exception Class Reference

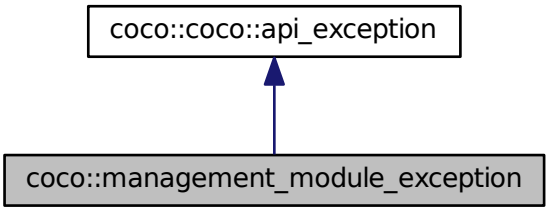
Management module exception class.

`#include <mgmt_module.h>`

Inheritance diagram for coco::management_module_exception:



Collaboration diagram for coco::management_module_exception:



Public Member Functions

- management_module_exception (const std::string &msg)
- management_module_exception (const char *msg)
- virtual ~management_module_exception () throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()

10.203.1 Detailed Description

This is the exceptions class thrown by management module subclasses. Every properly coded COCONUT management module should only throw exceptions of this type.

10.203.2 Constructor & Destructor Documentation

10.203.2.1 coco::management_module_exception::management_module_exception (const std::string & *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 57 of file mgmt_module.h.

10.203.2.2 coco::management_module_exception::management_module_exception (const char * *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 60 of file mgmt_module.h.

10.203.2.3 virtual coco::management_module_exception::~management_module_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 64 of file mgmt_module.h.

10.203.3 Member Function Documentation

10.203.3.1 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file expression.h.

10.203.3.2 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.203.3.3 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.203.3.4 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.203.3.5 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.203.3.6 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.203.3.7 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.203.3.8 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file expression.h.

10.203.3.9 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.203.3.10 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.203.3.11 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.203.3.12 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.203.3.13 const char * coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

Definition at line 57 of file api_exception.cc.

10.203.3.14 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.203.3.15 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.203.3.16 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.203.3.17 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.203.3.18 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file expression.h.

10.203.3.19 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.203.3.20 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following file:

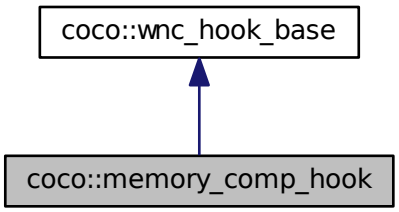
- mgmt_module.h

10.204 coco::memory_comp_hook Class Reference

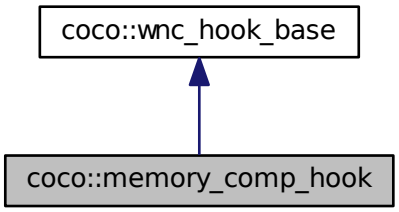
The memory computation hook (work node computation hook)

`#include <memory_hook.h>`

Inheritance diagram for coco::memory_comp_hook:



Collaboration diagram for coco::memory_comp_hook:



Public Member Functions

- memory_comp_hook ()
- virtual ~memory_comp_hook ()
- memory_comp_hook * new_copy () const
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *a=NULL, std::list< certificate > *b=NULL) const
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)
- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.204.1 Detailed Description

This class is a work node computation hook (see work_node_comp_hook), which calculates the memory of the work node (from the bound constraints) and keeps it in column "memory".

10.204.2 Constructor & Destructor Documentation

10.204.2.1 coco::memory_comp_hook::memory_comp_hook ( ) [inline]

Standard Constructor

Definition at line 46 of file memory_hook.h.

10.204.2.2 virtual coco::memory_comp_hook::~memory_comp_hook ( ) [inline, virtual]

Standard Destructor

Definition at line 49 of file memory_hook.h.

10.204.3 Member Function Documentation

10.204.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected, inherited]

Remove all columns, whose identifiers are stored in __managed_cols from table stable.

Definition at line 34 of file comp_hook.cc.

10.204.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in __managed_cols.

Definition at line 238 of file comp_hook.h.

10.204.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in __managed_cols.

Definition at line 135 of file comp_hook.h.

10.204.3.4 bool coco::memory_comp_hook::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon unregistering this hook, destroy the column "memory" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 79 of file memory_hook.h.

10.204.3.5 bool coco::memory_comp_hook::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the column "memory" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 74 of file memory_hook.h.

10.204.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline, inherited]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.204.3.7 memory_comp_hook* coco::memory_comp_hook::new_copy ( ) const [inline, virtual]

Clone Operation

Implements coco::wnc_hook_base.

Definition at line 52 of file memory_hook.h.

10.204.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_id of work_node `wn`.

Definition at line 43 of file comp_hook.cc.

10.204.3.9 void coco::memory_comp_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *a* = NULL, std::list< certificate > * *b* = NULL) const [inline, virtual]

The evaluation operator of this computation hook. It stores the newly required memory of the work_node `wn` in dbt_row `dbr`.

Implements coco::wnc_hook_base.

Definition at line 56 of file memory_hook.h.

10.204.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_relation of work_node `wn`.

Definition at line 46 of file comp_hook.cc.

The documentation for this class was generated from the following file:

- memory_hook.h

10.205 coco::coco::model Class Reference

The model class (an attributed DAG of expression nodes, lowest class in the model hierarchy)

Public Types

- typedef dag< expression_node > ::walker walker
- typedef dag< expression_node > ::const_walker const_walker
- typedef std::vector< walker > ::iterator ref_iterator
- typedef std::vector< walker > ::const_iterator const_ref_iterator

Public Member Functions

- model (model_gid *__id=NULL, bool clone=false)
- model (model_gid *__id, const erased_part &_ep, bool clone=false)
- model (int __num_of_vars)
- model (const model &__m)
- model (model_gid *__id, const model &__m)
- model (std::istream &inp, bool do_simplify=true, vdbl::database *_db=NULL, vdbl::userid _uid=vdbl-::userid(), std::vector< annotation > *_la=NULL, std::string mname=std::string(), bool continue_-on_errors=false)
- model (const char *name, bool do_simplify=true, vdbl::database *_db=NULL, vdbl::userid _uid=vdbl-::userid(), std::vector< annotation > *_la=NULL, bool continue_on_errors=false)
- ~model ()
- int next_num ()
- int next_variable_num ()
- int next_objective_num ()
- int next_constraint_num ()
- void detach_gid ()
- void compress_numbers ()
- void renumber_variables ()
- void renumber_constraints ()
- void prepare_kill ()
- bool micro_simplify ()
- bool basic_simplify (bool use_step2=true)
- bool d1_simplify (unsigned int preporder, const std::vector< interval > &ranges)
- void arrange_objectives ()
- void arrange_constraints ()
- void detect_0chain ()
- bool simplify_thin ()
- void set_counters ()
- void clr_sky_ground_link ()
- void write (std::ostream &__o=std::cout) const

- void check_counters (bool reset) const
- ref_iterator ghost_begin ()
- const_ref_iterator ghost_begin () const
- ref_iterator ghost_end ()
- const_ref_iterator ghost_end () const
- walker store_node (const walker &_w)
- walker store_variable (const walker &_w)
- walker store_ghost (const walker &_w)
- walker store_objective (const walker &_w)
- walker store_constraint (const walker &_w)
- void free_node_num (unsigned int _nnum)
- void free_var_num (int _vnum)
- void free_obj_num (const walker &_w, unsigned int _cnum)
- void free_const_num (const walker &_w, unsigned int _cnum)
- void free_all ()
- void free_nodes_on_destroy ()
- void keep_nodes_on_destroy ()
- void new_variables (int _new_num_of_vars)
- bool is_empty (const walker &_w) const
- walker empty_reference () const
- bool get_linear_coeffs (const const_walker &expr, vmtl::sparse_vector< double > &coeffs, double &constant, const std::vector< interval > &_ranges)
- bool get_linear_coeffs (const const_walker &expr, vmtl::sparse_vector< double > &coeffs, double &constant)

Methods for model access

These methods can be used for accessing important information of a model and the nodes, variables, constraints,...

- unsigned int number_of_variables () const
- unsigned int number_of_nodes () const
- unsigned int number_of_objectives () const
- unsigned int number_of_constraints () const
- unsigned int number_of_managed_nodes () const
- unsigned int number_of_managed_variables () const
- unsigned int number_of_managed_objectives () const
- unsigned int number_of_managed_constraints () const
- const walker & var (unsigned int i) const
- const walker & node (unsigned int i) const
- const walker & objective (unsigned int i) const
- const walker & constraint (unsigned int i) const
- model_gid * gid_data () const

- void remove_node (const walker &_w, unsigned int _nnum)
- void remove_node (const walker &_w)
- void remove_node (unsigned int __node_num)

Model construction methods

These methods can be used to build a model within a COCONUT module, if it is not read from a file in .dag format.

- walker ghost (unsigned int _nnum)
- walker constant (double _constant)
- walker constant (const std::vector< double > &_constant)
- walker variable (unsigned int _vnum)
- walker binary (const walker &_op1, const walker &_op2, int expr_type, double _coeff1=1.0, double _coeff2=1.0)
- walker binary (const walker &_op1, const walker &_op2, int expr_type, const basic_alltype &_params, double _coeff1=1.0, double _coeff2=1.0)
- walker unary (const walker &_op1, int expr_type, double _coeff=1.0)
- walker unary (const walker &_op1, int expr_type, const basic_alltype &_params, double _coeff=1.0)
- walker nary (const std::vector< walker > &_op, int expr_type, const std::vector< double > &_coeffs=std::vector< double >())
- walker nary (const std::vector< walker > &_op, int expr_type, const basic_alltype &_params, const std::vector< double > &_coeffs=std::vector< double >())
- walker vnary (int expr_type,...)
- walker vxnary (const basic_alltype &_params, int expr_type,...)

Methods for retrieving modeling information

These methods return names of variables and constants and other data which is irrelevant for the solution process but necessary for user interaction.

- const std::string model_name () const
- const std::string var_name (unsigned int n) const
- const std::string const_name (unsigned int n) const
- const std::string obj_name (unsigned int n) const
- double obj_adj (unsigned int n) const
- double obj_mult (unsigned int n) const
- size_t n_fixed_vars () const
- std::pair< const std::string, double > fixed_var (unsigned int n) const
- size_t n_unused_vars () const
- const std::string & unused_var (unsigned int n) const
- size_t n_unused_objs () const
- const std::string & unused_obj (unsigned int n) const
- size_t n_unused_constrs () const
- const std::string & unused_constr (unsigned int n) const
- const std::map< int, std::vector< double > > & trace_points () const
- const std::map< int, std::vector< double > > & optima () const
- bool get_obj_num (unsigned int node_num, unsigned int &obj_num) const
- bool get_const_num (unsigned int node_num, unsigned int &const_num) const

Public Attributes

- vmtl::sparse_matrix< double > lin
- std::vector < vmtl::sparse_matrix< double > > matd
- std::vector < vmtl::sparse_matrix < interval > > mati
- std::vector< int > ocoeff
- std::vector< walker > objectives
- std::vector< walker > constraints
- bool has_read_errors

Friends

- class dag_delta
- class dag_undelta

10.205.1 Detailed Description

This class is used for the internal representation of optimization problems. It stores one optimization problem, which is passed to the various inference engines. The class is a subclass of `dag<expression_node>`, so the optimization problem is stored in form of a DAG of nodes of type expression_node. Models are internally represented by a hierarchy of classes: model_iddata, model_gid, model.

10.205.2 Member Typedef Documentation

10.205.2.1 typedef std::vector<walker>::const_iterator coco::coco::model::const_ref_iterator

This is a const iterator to the various vectors of walkers

Definition at line 70 of file search_graph.cc.

10.205.2.2 typedef dag<expression_node>::const_walker coco::coco::model::const_walker

Const walker (generalized const pointer) to the DAG.

Definition at line 65 of file search_graph.cc.

10.205.2.3 typedef std::vector<walker>::iterator coco::coco::model::ref_iterator

This is an iterator to the various vectors of walkers

Definition at line 68 of file search_graph.cc.

10.205.2.4 typedef dag<expression_node>::walker coco::coco::model::walker

Walker (generalized pointer) to the DAG.

Definition at line 63 of file search_graph.cc.

10.205.3 Constructor & Destructor Documentation

10.205.3.1 coco::model::model (model_gid * *__id* = `NULL`, bool *clone* = `false`) `[inline]`

This constructor generates a new model. If `__id` is non-NULL and `clone` is `true`, the new model has model_gid `__id`. Otherwise, new a model_gid structure is generated, either empty (`__id` is NULL) or a copy of `__id`.

Definition at line 718 of file model.hpp.

10.205.3.2 coco::model::model (model_gid * __id, const erased_part & _ep, bool clone = false)

This constructor generates a new model from the erased_part structure _ep (produced by some erase subgraph method). If __id is non-NULL and clone is true, the new model has model_gid __id. Otherwise, new a model_gid structure is generated, either empty (__id is NULL) or a copy of __id.

Definition at line 238 of file model.cc.

10.205.3.3 coco::model::model (int __num_of_vars) [inline]

This constructor generates a new model prepared for __num_of_vars variables.

Definition at line 711 of file model.hpp.

10.205.3.4 coco::model::model (const model & __m) [inline]

Standard Copy Constructor, which generates a new model_gid.

Definition at line 740 of file model.hpp.

10.205.3.5 coco::model::model (model_gid * __id, const model & __m) [inline]

Copy Constructor, which transfers the model __m to the model_gid __id.

Definition at line 756 of file model.hpp.

10.205.3.6 coco::model::model (std::istream & inp, bool do_simplify = true, vdbl::database * _db = NULL, vdbl::userid _uid = vdbl::userid(), std::vector< annotation > * _la = NULL, std::string mname = std::string(), bool continue_on_errors = false)

This constructor reads a model in .dag format from istream inp and builds the internal structure. The parameter do_simplify determines, whether the simplifier is invoked after reading. The parameters _db, _uid, and _la can be used to initialize the search database and the model annotations. The parameter mname can be used to initialize the model name. If continue_on_errors is true, the constructor does not throw an I/O exception.

Definition at line 554 of file model.cc.

10.205.3.7 coco::model::model (const char * name, bool do_simplify = true, vdbl::database * _db = NULL, vdbl::userid _uid = vdbl::userid(), std::vector< annotation > * _la = NULL, bool continue_on_errors = false)

This constructor reads a model in .dag format from file name and builds the internal structure. The parameter do_simplify determines, whether the simplifier is invoked after reading. The parameters _db, _uid, and _la can be used to initialize the search database and the model annotations. If continue_on_errors is true, the constructor does not throw an I/O exception.

Definition at line 564 of file model.cc.

10.205.3.8 coco::model::~model ( ) [inline]

Standard Destructor

Definition at line 772 of file model.hpp.

10.205.4 Member Function Documentation

10.205.4.1 void coco::model::arrange_constraints ( ) [inline]

A call to arrange_constraints sorts the constraints according to their complexity.

Definition at line 1810 of file model.hpp.

10.205.4.2 void coco::model::arrange_objectives ( ) [inline]

A call to arrange_objectives sorts the objectives according to their complexity.

Definition at line 1805 of file model.hpp.

10.205.4.3 bool coco::model::basic_simplify (bool *use_step2* = `true`)

This method calls the full model simplifier.

Definition at line 4466 of file model.cc.

10.205.4.4 model::walker coco::model::binary (const walker & *_op1*, const walker & *_op2*, int *expr_type*, double *_coeff1* = `1.0`, double *_coeff2* = `1.0`) [inline]

This method constructs a new binary operation with left operand `_op1`, multiplicative coefficient `_coeff1`, right operand `_op2`, multiplicative coefficient `_coeff2`, and expression type `expr_type`.

Definition at line 2150 of file model.hpp.

10.205.4.5 model::walker coco::model::binary (const walker & *_op1*, const walker & *_op2*, int *expr_type*, const basic_alltype & *_params*, double *_coeff1* = `1.0`, double *_coeff2* = `1.0`) [inline]

This method constructs a new binary operation with left operand `_op1`, multiplicative coefficient `_coeff1`, right operand `_op2`, multiplicative coefficient `_coeff2`, expression type `expr_type`, and additional expression parameters `_params`.

Definition at line 2170 of file model.hpp.

10.205.4.6 void coco::model::check_counters (bool *reset*) const

Check consistency of the number of parents and children of all nodes.

Definition at line 4816 of file model.cc.

10.205.4.7 void coco::model::clr_sky_ground_link ( ) [inline]

This method removes the edge from sky to ground in the DAG after all other nodes have been inserted.

Definition at line 2271 of file model.hpp.

10.205.4.8 void coco::model::compress_numbers ( ) [inline]

If this method is called, the nodes, variables, and constraints are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed.

Definition at line 1826 of file model.hpp.

10.205.4.9 const std::string coco::model::const_name (unsigned int *n*) const [inline]

This method returns the name of the constraint n.

Definition at line 827 of file model.hpp.

10.205.4.10 model::walker coco::model::constant (double *_constant*) [inline]

This method constructs a new scalar constant with value _constant.

Definition at line 2058 of file model.hpp.

10.205.4.11 model::walker coco::model::constant (const std::vector< double > & *_constant*) [inline]

This method constructs a new vector constant with value _constant.

Definition at line 2069 of file model.hpp.

10.205.4.12 const model::walker & coco::model::constraint (unsigned int *i*) const [inline]

A call to this method returns a walker to constraint i.

Definition at line 873 of file model.hpp.

10.205.4.13 bool coco::model::d1_simplify (unsigned int *preporder*, const std::vector< interval > & *ranges*)

This method calls the 1D function simplifier.

Definition at line 538 of file model_d1simplify.cc.

10.205.4.14 void coco::model::detach_gid ( ) [inline]

A call to this method detaches the model from its model_gid. A new model_gid is generated within the same model_iddata.

Definition at line 709 of file model.hpp.

10.205.4.15 void coco::model::detect_0chain ( )

This method identifies all 0-chains (univariate functions) in the DAG.

Definition at line 4779 of file model.cc.

10.205.4.16 model::walker coco::model::empty_reference ( ) const [inline]

This method returns an empty node reference.

Definition at line 706 of file model.hpp.

10.205.4.17 std::pair< const std::string, double > coco::model::fixed_var (unsigned int *n*) const [inline]

This method returns the name and value of the fixed variable n.

Definition at line 834 of file model.hpp.

10.205.4.18 void coco::model::free_all ( ) [inline]

A call to this method removes all node, variable, and constraint references kept by this model.

Definition at line 1927 of file model.hpp.

10.205.4.19 void coco::model::free_const_num (const walker & _w, unsigned int _cnum) [inline]

This method removes the constraint _cnum from all model data structures.

Definition at line 2030 of file model.hpp.

10.205.4.20 void coco::model::free_node_num (unsigned int _nnum) [inline]

This method removes the node _nnum from all model data structures.

Definition at line 1993 of file model.hpp.

10.205.4.21 void coco::model::free_nodes_on_destroy ( ) [inline]

This method marks tells the model to call free_all during model destruction.

Definition at line 802 of file model.hpp.

10.205.4.22 void coco::model::free_obj_num (const walker & _w, unsigned int _cnum) [inline]

This method removes the objective _cnum from all model data structures.

Definition at line 2017 of file model.hpp.

10.205.4.23 void coco::model::free_var_num (int _vnum) [inline]

This method removes the variable _vnum from all model data structures.

Definition at line 2005 of file model.hpp.

10.205.4.24 bool coco::model::get_const_num (unsigned int *node_num*, unsigned int & *const_num*) const [inline]

The get_const_num method assigns to const_num the constraint number associated with the constraint, whose result node has node number node_num. The return value is true, if there is a constraint associated with node node_num.

Definition at line 798 of file model.hpp.

10.205.4.25 bool coco::model::get_linear_coeffs (const const_walker & *expr*, vmtl::sparse_vector< double > & *coeffs*, double & *constant*, const std::vector< interval > & *_ranges*) [inline]

A call to this method retrieves the coefficients (coeffs) and the constant (constant) of the linear function, whose result node is expr. It returns true, if the expression was indeed linear and false otherwise. The retrieval of the coefficients is to be performed in the feasible set determined by the node ranges _ranges.

Definition at line 3201 of file model.hpp.

10.205.4.26 bool coco::model::get_linear_coeffs (const const_walker & *expr,* vmtl::sparse_vector< double > & *coeffs,* double & *constant*) [inline]

A call to this method retrieves the coefficients (`coeffs`) and the constant (`constant`) of the linear function, whose result node is `expr`. It returns `true`, if the expression was indeed linear and `false` otherwise.

Definition at line 3212 of file model.hpp.

10.205.4.27 bool coco::model::get_obj_num (unsigned int *node_num,* unsigned int & *obj_num*) const [inline]

The get_obj_num method assigns to `obj_num` the objective number associated with the objective, whose result node has node number `node_num`. The return value is `true`, if there is an objective associated with node `node_num`.

Definition at line 794 of file model.hpp.

10.205.4.28 model::walker coco::model::ghost (unsigned int *_nnum*) [inline]

This method constructs a new ghost with node number `_nnum`.

Definition at line 2080 of file model.hpp.

10.205.4.29 ref_iterator coco::coco::model::ghost_begin ( ) [inline]

Get an iterator to the first ghost.

Definition at line 412 of file search_graph.cc.

10.205.4.30 const_ref_iterator coco::coco::model::ghost_begin ( ) const [inline]

Get a const iterator to the first ghost.

Definition at line 414 of file search_graph.cc.

10.205.4.31 ref_iterator coco::coco::model::ghost_end ( ) [inline]

Get an iterator beyond the last ghost.

Definition at line 416 of file search_graph.cc.

10.205.4.32 const_ref_iterator coco::coco::model::ghost_end ( ) const [inline]

Get a const iterator beyond the last ghost.

Definition at line 418 of file search_graph.cc.

10.205.4.33 model_gid* coco::coco::model::gid_data ( ) const [inline]

This method returns the model_gid structure this model belongs to.

Definition at line 212 of file search_graph.cc.

10.205.4.34 bool coco::model::is_empty (const walker & _w) const [inline]

Check whether this node is empty (unused).

Definition at line 705 of file model.hpp.

10.205.4.35 void coco::model::keep_nodes_on_destroy ( ) [inline]

This method marks tells the model not to call free_all during model destruction.

Definition at line 813 of file model.hpp.

10.205.4.36 bool coco::model::micro_simplify ( )

This method calls the micro simplifier which only initializes the most important data and semantics information.

Definition at line 3751 of file model.cc.

10.205.4.37 const std::string coco::model::model_name ( ) const [inline]

This method returns the name of the model.

Definition at line 824 of file model.hpp.

10.205.4.38 size_t coco::model::n_fixed_vars ( ) const [inline]

This method returns the number of variables fixed by the preprocessor.

Definition at line 833 of file model.hpp.

10.205.4.39 size_t coco::model::n_unused_constrs ( ) const [inline]

This method returns the number of unused constraints.

Definition at line 843 of file model.hpp.

10.205.4.40 size_t coco::model::n_unused_objs ( ) const [inline]

This method returns the number of unused objectives.

Definition at line 839 of file model.hpp.

10.205.4.41 size_t coco::model::n_unused_vars ( ) const [inline]

This method returns the number of unused variables.

Definition at line 836 of file model.hpp.

10.205.4.42 model::walker coco::model::nary (const std::vector< walker > & _op, int expr_type, const std::vector< double > & _coeffs = std::vector<double>()) [inline]

This method constructs a new n-ary operation with operand list _op, multiplicative coefficient list _coeffs, and expression type expr_type.

Definition at line 2212 of file model.hpp.

10.205.4.43 model::walker coco::model::nary (const std::vector< walker > & *_op,* int *expr_type,* const basic_alltype & *_params,* const std::vector< double > & *_coeffs =* std::vector<double>()) [inline]

This method constructs a new n-ary operation with operand list _op, multiplicative coefficient list _coeffs, expression type expr_type, and additional expression parameters _params.

Definition at line 2231 of file model.hpp.

10.205.4.44 void coco::model::new_variables (int *_new_num_of_vars*) [inline]

This method sets the number of variables in the model to _new_num_of_vars and communicates the change to all models below the same model_iddata.

Definition at line 2053 of file model.hpp.

10.205.4.45 int coco::model::next_constraint_num ( ) [inline]

This method returns the next free constraint number.

Definition at line 1818 of file model.hpp.

10.205.4.46 int coco::model::next_num ( ) [inline]

This method returns the next free node number.

Definition at line 1815 of file model.hpp.

10.205.4.47 int coco::model::next_objective_num ( ) [inline]

This method returns the next free objective number.

Definition at line 1817 of file model.hpp.

10.205.4.48 int coco::model::next_variable_num ( ) [inline]

This method returns the next free variable number.

Definition at line 1816 of file model.hpp.

10.205.4.49 const model::walker & coco::model::node (unsigned int *i*) const [inline]

A call to this method returns a walker to node i.

Definition at line 864 of file model.hpp.

10.205.4.50 unsigned int coco::model::number_of_constraints ( ) const [inline]

The number_of_constraints method returns the number of constraints across all models governed by the same model_iddata.

Definition at line 858 of file model.hpp.

10.205.4.51 unsigned int coco::coco::model::number_of_managed_constraints ( ) const [inline]

This method returns the number of constraints managed by this model class (the number of constraints in this DAG.

Definition at line 201 of file search_graph.cc.

10.205.4.52 unsigned int coco::coco::model::number_of_managed_nodes ( ) const [inline]

This method returns the number of nodes managed by this model class (the number of nodes in this DAG.

Definition at line 189 of file search_graph.cc.

10.205.4.53 unsigned int coco::coco::model::number_of_managed_objectives ( ) const [inline]

This method returns the number of objectives managed by this model class (the number of objectives in this DAG.

Definition at line 197 of file search_graph.cc.

10.205.4.54 unsigned int coco::coco::model::number_of_managed_variables ( ) const [inline]

This method returns the number of variables managed by this model class (the number of variables in this DAG.

Definition at line 193 of file search_graph.cc.

10.205.4.55 unsigned int coco::model::number_of_nodes ( ) const [inline]

The number_of_nodes method returns the number of nodes across all models governed by the same model_iddata.

Definition at line 861 of file model.hpp.

10.205.4.56 unsigned int coco::model::number_of_objectives ( ) const [inline]

The number_of_objectives method returns the number of objectives across all models governed by the same model_iddata.

Definition at line 855 of file model.hpp.

10.205.4.57 unsigned int coco::model::number_of_variables ( ) const [inline]

The number_of_variables method returns the number of variables across all models governed by the same model_iddata.

Definition at line 852 of file model.hpp.

10.205.4.58 double coco::model::obj_adj (unsigned int *n*) const [inline]

This method returns the additive constant in the objective function n.

Definition at line 831 of file model.hpp.

10.205.4.59 double coco::model::obj_mult (unsigned int *n*) const [inline]

This method returns the scalar multiplier of the objective function n.

Definition at line 832 of file model.hpp.

10.205.4.60 const std::string coco::model::obj_name (unsigned int *n*) const [inline]

This method returns the name of the objective function n.

Definition at line 829 of file model.hpp.

10.205.4.61 const model::walker & coco::model::objective (unsigned int *i*) const [inline]

A call to this method returns a walker to objective i.

Definition at line 870 of file model.hpp.

10.205.4.62 const std::map< int, std::vector< double > > & coco::model::optima ( ) const [inline]

This method returns the known optima defined. The return value is a map of known optima numbers to coordinate vectors.

Definition at line 849 of file model.hpp.

10.205.4.63 void coco::coco::model::prepare_kill ( ) [inline]

This method takes care that upon deletion everything goes away.

Definition at line 373 of file search_graph.cc.

10.205.4.64 void coco::model::remove_node (const walker & *_w,* unsigned int *_nnum*)

The remove_node methods remove the node, specified by its walker _w, its node number _nnum, or by both.

Definition at line 283 of file model.cc.

10.205.4.65 void coco::model::remove_node (const walker & *_w*) [inline]

The remove_node methods remove the node, specified by its walker _w, its node number _nnum, or by both.

Definition at line 2043 of file model.hpp.

10.205.4.66 void coco::model::remove_node (unsigned int *__node_num*) [inline]

The remove_node methods remove the node, specified by its walker _w, its node number _nnum, or by both.

Definition at line 2048 of file model.hpp.

10.205.4.67 void coco::coco::model::renumber_constraints ( )

If this method is called, the constraints are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed.

10.205.4.68 void coco::model::renumber_variables ( ) [inline]

If this method is called, the variables are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed.

Definition at line 1820 of file model.hpp.

10.205.4.69 void coco::model::set_counters ( ) [inline]

This method initializes the `n_parents` and `n_children` members of all nodes.

See also

expression_node.

Definition at line 2277 of file model.hpp.

10.205.4.70 bool coco::model::simplify_thin ( ) [inline]

This simplifier replaces all thin nodes by corresponding constants and simplifies the DAG accordingly.

Bug The simplify_thin method does not yet work properly.

Definition at line 1800 of file model.hpp.

10.205.4.71 model::walker coco::model::store_constraint (const walker & _w) [inline]

This method stores a new constraint in all model data structures

Definition at line 1900 of file model.hpp.

10.205.4.72 model::walker coco::model::store_ghost (const walker & _w) [inline]

This method stores a new ghost in all model data structures

Definition at line 1912 of file model.hpp.

10.205.4.73 model::walker coco::model::store_node (const walker & _w) [inline]

This method stores a new node in all model data structures

Definition at line 1856 of file model.hpp.

10.205.4.74 model::walker coco::model::store_objective (const walker & _w) [inline]

This method stores a new objective in all model data structures

Definition at line 1888 of file model.hpp.

10.205.4.75 model::walker coco::model::store_variable (const walker & _w) [inline]

This method stores a new variable in all model data structures

Definition at line 1872 of file model.hpp.

10.205.4.76 const std::map< int, std::vector< double > > & coco::model::trace_points ( ) const [inline]

This method returns the trace points defined. The return value is a map of trace point numbers to coordinate vectors.

Definition at line 847 of file model.hpp.

10.205.4.77 model::walker coco::model::unary (const walker & *_op1,* int *expr_type,* double *_coeff =* 1.0) [inline]

This method constructs a new unary operation with operand _op1, multiplicative coefficient _coeff, and expression type expr_type.

Definition at line 2187 of file model.hpp.

10.205.4.78 model::walker coco::model::unary (const walker & *_op1,* int *expr_type,* const basic_alltype & *_params,* double *_coeff =* 1.0) [inline]

This method constructs a new unary operation with operand _op1, multiplicative coefficient _coeff, expression type expr_type, and additional expression parameters _params.

Definition at line 2201 of file model.hpp.

10.205.4.79 const std::string & coco::model::unused_constr (unsigned int *n*) const [inline]

This method returns the name of the unused constraint n.

Definition at line 845 of file model.hpp.

10.205.4.80 const std::string & coco::model::unused_obj (unsigned int *n*) const [inline]

This method returns the name of the unused objectives n.

Definition at line 841 of file model.hpp.

10.205.4.81 const std::string & coco::model::unused_var (unsigned int *n*) const [inline]

This method returns the name of the unused variable n.

Definition at line 837 of file model.hpp.

10.205.4.82 const model::walker & coco::model::var (unsigned int *i*) const [inline]

A call to this method returns a walker to variable i.

Definition at line 867 of file model.hpp.

10.205.4.83 const std::string coco::model::var_name (unsigned int *n*) const [inline]

This method returns the name of the variable n.

Definition at line 825 of file model.hpp.

10.205.4.84 model::walker coco::model::variable (unsigned int *_vnum*)

This method constructs a new variable with variable number _vnum.

Definition at line 4680 of file model.cc.

10.205.4.85 model::walker coco::model::vnary (int *expr_type, ...*)

This method constructs a new n-ary operation with expression type expr_type. The operands are specified as the additional parameters and need to be of type walker.

Definition at line 4724 of file model.cc.

10.205.4.86 model::walker coco::model::vxnary (const basic_alltype & *_params,* int *expr_type, ...*)

This method constructs a new n-ary operation with expression type expr_type and additional expression parameters _params. The operands and multiplicative constants are specified as the additional parameters and need to be of type walker followed by a double, and so on. The final double (in case it is 1.0) can be left out.

Definition at line 4749 of file model.cc.

10.205.4.87 void coco::model::write (std::ostream & *__o* = std::cout) const

This method writes the DAG in .dag format to the ostream __o.

Definition at line 4553 of file model.cc.

10.205.5 Friends And Related Function Documentation

10.205.5.1 friend class dag_delta [friend]

Definition at line 604 of file search_graph.cc.

10.205.5.2 friend class dag_undelta [friend]

Definition at line 605 of file search_graph.cc.

10.205.6 Member Data Documentation

10.205.6.1 std::vector<walker> coco::coco::model::constraints

This is a vector of walkers pointing to the result nodes of the constraints.

Definition at line 114 of file search_graph.cc.

10.205.6.2 bool coco::coco::model::has_read_errors

This member is set to if an I/O exception was raised during model reading.

Definition at line 117 of file search_graph.cc.

10.205.6.3 vmtl::sparse_matrix<double> coco::coco::model::lin

The `lin` member is the matrix of linear constraints.

Definition at line 93 of file search_graph.cc.

10.205.6.4 std::vector<vmtl::sparse_matrix<double> > coco::coco::model::matd

The `matd` member is a vector of double matrices.

Definition at line 95 of file search_graph.cc.

10.205.6.5 std::vector<vmtl::sparse_matrix<interval> > coco::coco::model::mati

The `matd` member is a vector of interval matrices.

Definition at line 97 of file search_graph.cc.

10.205.6.6 std::vector<walker> coco::coco::model::objectives

This is a vector of walkers pointing to the result nodes of the objective functions.

Definition at line 111 of file search_graph.cc.

10.205.6.7 std::vector<int> coco::coco::model::ocoeff

These objective coefficients are used to transfer any constraint satisfaction problem or optimization problem to a vector valued minimization problem. The values are per objective

1	for a minimization problem
-1	for a maximization problem
0	for a constraint satisfaction problem

Definition at line 108 of file search_graph.cc.

The documentation for this class was generated from the following files:

- model.h
- model.cc
- model.hpp
- model_d1func.cc
- model_d1simplify.cc

10.206 coco::model Class Reference

The model class (an attributed DAG of expression nodes, lowest class in the model hierarchy)

`#include <model.h>`

Classes

- struct __docompare_nodes

- struct __docompare_variables
- class detect_0chain_visitor
- struct detect_0chain_visitor_st
- class lincoeff_visitor
- struct lincoeff_visitor_ret
- struct lincoeff_visitor_st
- class simplify_visitor_0
- class simplify_visitor_m
- class sort_constraints

Public Types

- typedef dag< expression_node > ::walker walker
- typedef dag< expression_node > ::const_walker const_walker
- typedef std::vector< walker > ::iterator ref_iterator
- typedef std::vector< walker > ::const_iterator const_ref_iterator

Public Member Functions

- model (model_gid *__id=NULL, bool clone=false)
- model (model_gid *__id, const erased_part &_ep, bool clone=false)
- model (int __num_of_vars)
- model (const model &__m)
- model (model_gid *__id, const model &__m)
- model (std::istream &inp, bool do_simplify=true, vdbl::database *_db=NULL, vdbl::userid _uid=vdbl::userid(), std::vector< annotation > *_la=NULL, std::string mname=std::string(), bool continue_on_errors=false)
- model (const char *name, bool do_simplify=true, vdbl::database *_db=NULL, vdbl::userid _uid=vdbl::userid(), std::vector< annotation > *_la=NULL, bool continue_on_errors=false)
- ~model ()
- int next_num ()
- int next_variable_num ()
- int next_objective_num ()
- int next_constraint_num ()
- void detach_gid ()
- void compress_numbers ()
- void renumber_variables ()
- void renumber_constraints ()
- void prepare_kill ()
- bool micro_simplify ()
- bool basic_simplify (bool use_step2=true)
- bool d1_simplify (unsigned int preporder, const std::vector< interval > &ranges)
- void arrange_objectives ()
- void arrange_constraints ()
- void detect_0chain ()
- bool simplify_thin ()
- void set_counters ()
- void clr_sky_ground_link ()
- void write (std::ostream &__o=std::cout) const

- void check_counters (bool reset) const
- ref_iterator ghost_begin ()
- const_ref_iterator ghost_begin () const
- ref_iterator ghost_end ()
- const_ref_iterator ghost_end () const
- walker store_node (const walker &_w)
- walker store_variable (const walker &_w)
- walker store_ghost (const walker &_w)
- walker store_objective (const walker &_w)
- walker store_constraint (const walker &_w)
- void free_node_num (unsigned int _nnum)
- void free_var_num (int _vnum)
- void free_obj_num (const walker &_w, unsigned int _cnum)
- void free_const_num (const walker &_w, unsigned int _cnum)
- void free_all ()
- void free_nodes_on_destroy ()
- void keep_nodes_on_destroy ()
- void new_variables (int _new_num_of_vars)
- bool is_empty (const walker &_w) const
- walker empty_reference () const
- bool get_linear_coeffs (const const_walker &expr, vmtl::sparse_vector< double > &coeffs, double &constant, const std::vector< interval > &_ranges)
- bool get_linear_coeffs (const const_walker &expr, vmtl::sparse_vector< double > &coeffs, double &constant)

Methods for model access

These methods can be used for accessing important information of a model and the nodes, variables, constraints,...

- unsigned int number_of_variables () const
- unsigned int number_of_nodes () const
- unsigned int number_of_objectives () const
- unsigned int number_of_constraints () const
- unsigned int number_of_managed_nodes () const
- unsigned int number_of_managed_variables () const
- unsigned int number_of_managed_objectives () const
- unsigned int number_of_managed_constraints () const
- const walker & var (unsigned int i) const
- const walker & node (unsigned int i) const
- const walker & objective (unsigned int i) const
- const walker & constraint (unsigned int i) const
- model_gid * gid_data () const

- void remove_node (const walker &_w, unsigned int _nnum)
- void remove_node (const walker &_w)
- void remove_node (unsigned int __node_num)

Model construction methods

These methods can be used to build a model within a COCONUT module, if it is not read from a file in .dag format.

- walker ghost (unsigned int _nnum)
- walker constant (double _constant)
- walker constant (const std::vector< double > &_constant)
- walker variable (unsigned int _vnum)
- walker binary (const walker &_op1, const walker &_op2, int expr_type, double _coeff1=1.0, double _coeff2=1.0)
- walker binary (const walker &_op1, const walker &_op2, int expr_type, const basic_alltype &_params, double _coeff1=1.0, double _coeff2=1.0)
- walker unary (const walker &_op1, int expr_type, double _coeff=1.0)
- walker unary (const walker &_op1, int expr_type, const basic_alltype &_params, double _coeff=1.0)
- walker nary (const std::vector< walker > &_op, int expr_type, const std::vector< double > &_coeffs=std::vector< double >())
- walker nary (const std::vector< walker > &_op, int expr_type, const basic_alltype &_params, const std::vector< double > &_coeffs=std::vector< double >())
- walker vnary (int expr_type,...)
- walker vxnary (const basic_alltype &_params, int expr_type,...)

Methods for retrieving modeling information

These methods return names of variables and constants and other data which is irrelevant for the solution process but necessary for user interaction.

- const std::string model_name () const
- const std::string var_name (unsigned int n) const
- const std::string const_name (unsigned int n) const
- const std::string obj_name (unsigned int n) const
- double obj_adj (unsigned int n) const
- double obj_mult (unsigned int n) const
- size_t n_fixed_vars () const
- std::pair< const std::string, double > fixed_var (unsigned int n) const
- size_t n_unused_vars () const
- const std::string & unused_var (unsigned int n) const
- size_t n_unused_objs () const
- const std::string & unused_obj (unsigned int n) const
- size_t n_unused_constrs () const
- const std::string & unused_constr (unsigned int n) const
- const std::map< int, std::vector< double > > & trace_points () const
- const std::map< int, std::vector< double > > & optima () const
- bool get_obj_num (unsigned int node_num, unsigned int &obj_num) const
- bool get_const_num (unsigned int node_num, unsigned int &const_num) const

Public Attributes

- vmtl::sparse_matrix< double > lin
- std::vector < vmtl::sparse_matrix< double > > matd
- std::vector < vmtl::sparse_matrix < interval > > mati
- std::vector< int > ocoeff
- std::vector< walker > objectives
- std::vector< walker > constraints
- bool has_read_errors

Friends

- class dag_delta
- class dag_undelta

10.206.1 Detailed Description

This class is used for the internal representation of optimization problems. It stores one optimization problem, which is passed to the various inference engines. The class is a subclass of `dag<expression_node>`, so the optimization problem is stored in form of a DAG of nodes of type expression_node. Models are internally represented by a hierarchy of classes: model_iddata, model_gid, model.

10.206.2 Member Typedef Documentation

10.206.2.1 typedef std::vector<walker>::const_iterator coco::model::const_ref_iterator

This is a const iterator to the various vectors of walkers

Definition at line 70 of file model.h.

10.206.2.2 typedef dag<expression_node>::const_walker coco::model::const_walker

Const walker (generalized const pointer) to the DAG.

Definition at line 65 of file model.h.

10.206.2.3 typedef std::vector<walker>::iterator coco::model::ref_iterator

This is an iterator to the various vectors of walkers

Definition at line 68 of file model.h.

10.206.2.4 typedef dag<expression_node>::walker coco::model::walker

Walker (generalized pointer) to the DAG.

Definition at line 63 of file model.h.

10.206.3 Constructor & Destructor Documentation

10.206.3.1 coco::model::model (model_gid * *__id* = `NULL`, bool *clone* = `false`)

This constructor generates a new model. If `__id` is non-NULL and `clone` is `true`, the new model has model_gid `__id`. Otherwise, new a model_gid structure is generated, either empty (`__id` is NULL) or a copy of `__id`.

10.206.3.2 coco::model::model (model_gid * *__id*, const erased_part & *_ep*, bool *clone* = `false`)

This constructor generates a new model from the erased_part structure `_ep` (produced by some erase subgraph method). If `__id` is non-NULL and `clone` is `true`, the new model has model_gid `__id`. Otherwise, new a model_gid structure is generated, either empty (`__id` is NULL) or a copy of `__id`.

10.206.3.3 coco::model::model (int *__num_of_vars*)

This constructor generates a new model prepared for ___num_of_vars variables.

10.206.3.4 coco::model::model (const model & *__m*)

Standard Copy Constructor, which generates a new model_gid.

10.206.3.5 coco::model::model (model_gid * *__id,* const model & *__m*)

Copy Constructor, which transfers the model ___m to the model_gid ___id.

10.206.3.6 coco::model::model (std::istream & *inp,* bool *do_simplify* = true, vdbl::database * *_db* = NULL, vdbl::userid *_uid* = vdbl::userid(), std::vector< annotation > * *_la* = NULL, std::string *mname* = std::string(), bool *continue_on_errors* = false)

This constructor reads a model in .dag format from istream inp and builds the internal structure. The parameter do_simplify determines, whether the simplifier is invoked after reading. The parameters _db, _uid, and _la can be used to initialize the search database and the model annotations. The parameter mname can be used to initialize the model name. If continue_on_errors is true, the constructor does not throw an I/O exception.

10.206.3.7 coco::model::model (const char * *name,* bool *do_simplify* = true, vdbl::database * *_db* = NULL, vdbl::userid *_uid* = vdbl::userid(), std::vector< annotation > * *_la* = NULL, bool *continue_on_errors* = false)

This constructor reads a model in .dag format from file name and builds the internal structure. The parameter do_simplify determines, whether the simplifier is invoked after reading. The parameters _db, _uid, and _la can be used to initialize the search database and the model annotations. If continue_on_errors is true, the constructor does not throw an I/O exception.

10.206.3.8 coco::model::~model ( )

Standard Destructor

10.206.4 Member Function Documentation

10.206.4.1 void coco::model::arrange_constraints ( )

A call to arrange_constraints sorts the constraints according to their complexity.

10.206.4.2 void coco::model::arrange_objectives ( )

A call to arrange_objectives sorts the objectives according to their complexity.

10.206.4.3 bool coco::model::basic_simplify (bool *use_step2* = true)

This method calls the full model simplifier.

10.206.4.4 walker coco::model::binary (const walker & *_op1,* const walker & *_op2,* int *expr_type,* double *_coeff1* = 1.0, double *_coeff2* = 1.0)

This method constructs a new binary operation with left operand _op1, multiplicative coefficient _coeff1, right operand _op2, multiplicative coefficient _coeff2, and expression type expr_type.

10.206.4.5 walker coco::model::binary (const walker & *_op1,* const walker & *_op2,* int *expr_type,* const basic_alltype & *_params,* double *_coeff1* = 1.0, double *_coeff2* = 1.0)

This method constructs a new binary operation with left operand _op1, multiplicative coefficient _coeff1, right operand _op2, multiplicative coefficient _coeff2, expression type expr_type, and additional expression parameters _params.

10.206.4.6 void coco::model::check_counters (bool *reset*) const

Check consistency of the number of parents and children of all nodes.

10.206.4.7 void coco::model::clr_sky_ground_link ( )

This method removes the edge from sky to ground in the DAG after all other nodes have been inserted.

10.206.4.8 void coco::model::compress_numbers ( )

If this method is called, the nodes, variables, and constraints are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed.

10.206.4.9 const std::string coco::model::const_name (unsigned int *n*) const

This method returns the name of the constraint n.

10.206.4.10 walker coco::model::constant (double *_constant*)

This method constructs a new scalar constant with value _constant.

10.206.4.11 walker coco::model::constant (const std::vector< double > & *_constant*)

This method constructs a new vector constant with value _constant.

10.206.4.12 const walker& coco::model::constraint (unsigned int *i*) const

A call to this method returns a walker to constraint i.

10.206.4.13 bool coco::model::d1_simplify (unsigned int *preporder,* const std::vector< interval > & *ranges*)

This method calls the 1D function simplifier.

10.206.4.14 void coco::model::detach_gid ( )

A call to this method detaches the model from its model_gid. A new model_gid is generated within the same model_iddata.

10.206.4.15 void coco::model::detect_0chain ( )

This method identifies all 0-chains (univariate functions) in the DAG.

10.206.4.16 walker coco::model::empty_reference ( ) const

This method returns an empty node reference.

10.206.4.17 std::pair<const std::string, double> coco::model::fixed_var (unsigned int *n*) const

This method returns the name and value of the fixed variable `n`.

10.206.4.18 void coco::model::free_all ( )

A call to this method removes all node, variable, and constraint references kept by this model.

10.206.4.19 void coco::model::free_const_num (const walker & *_w,* unsigned int *_cnum*)

This method removes the constraint `_cnum` from all model data structures.

10.206.4.20 void coco::model::free_node_num (unsigned int *_nnum*)

This method removes the node `_nnum` from all model data structures.

10.206.4.21 void coco::model::free_nodes_on_destroy ( )

This method marks tells the model to call free_all during model destruction.

10.206.4.22 void coco::model::free_obj_num (const walker & *_w,* unsigned int *_cnum*)

This method removes the objective `_cnum` from all model data structures.

10.206.4.23 void coco::model::free_var_num (int *_vnum*)

This method removes the variable `_vnum` from all model data structures.

10.206.4.24 bool coco::model::get_const_num (unsigned int *node_num,* unsigned int & *const_num*) const

The get_const_num method assigns to `const_num` the constraint number associated with the constraint, whose result node has node number `node_num`. The return value is `true`, if there is a constraint associated with node `node_num`.

10.206.4.25 bool coco::model::get_linear_coeffs (const const_walker & *expr,* vmtl::sparse_vector< double > & *coeffs,* double & *constant,* const std::vector< interval > & *_ranges*)

A call to this method retrieves the coefficients (`coeffs`) and the constant (`constant`) of the linear function, whose result node is `expr`. It returns `true`, if the expression was indeed linear and `false` otherwise. The retrieval of the coefficients is to be performed in the feasible set determined by the node ranges `_ranges`.

10.206.4.26 bool coco::model::get_linear_coeffs (const const_walker & *expr,* vmtl::sparse_vector< double > & *coeffs,* double & *constant*)

A call to this method retrieves the coefficients (`coeffs`) and the constant (`constant`) of the linear function, whose result node is `expr`. It returns `true`, if the expression was indeed linear and `false` otherwise.

10.206.4.27 bool coco::model::get_obj_num (unsigned int *node_num,* unsigned int & *obj_num*) const

The get_obj_num method assigns to `obj_num` the objective number associated with the objective, whose result node has node number `node_num`. The return value is `true`, if there is an objective associated with node `node_num`.

10.206.4.28 walker coco::model::ghost (unsigned int *_nnum*)

This method constructs a new ghost with node number _nnum.

10.206.4.29 ref_iterator coco::model::ghost_begin ( ) [inline]

Get an iterator to the first ghost.

Definition at line 412 of file model.h.

10.206.4.30 const_ref_iterator coco::model::ghost_begin ( ) const [inline]

Get a const iterator to the first ghost.

Definition at line 414 of file model.h.

10.206.4.31 ref_iterator coco::model::ghost_end ( ) [inline]

Get an iterator beyond the last ghost.

Definition at line 416 of file model.h.

10.206.4.32 const_ref_iterator coco::model::ghost_end ( ) const [inline]

Get a const iterator beyond the last ghost.

Definition at line 418 of file model.h.

10.206.4.33 model_gid* coco::model::gid_data ( ) const [inline]

This method returns the model_gid structure this model belongs to.

Definition at line 212 of file model.h.

10.206.4.34 bool coco::model::is_empty (const walker & *_w*) const

Check whether this node is empty (unused).

10.206.4.35 void coco::model::keep_nodes_on_destroy ( )

This method marks tells the model not to call free_all during model destruction.

10.206.4.36 bool coco::model::micro_simplify ( )

This method calls the micro simplifier which only initializes the most important data and semantics information.

10.206.4.37 const std::string coco::model::model_name ( ) const

This method returns the name of the model.

10.206.4.38 size_t coco::model::n_fixed_vars ( ) const

This method returns the number of variables fixed by the preprocessor.

10.206.4.39 size_t coco::model::n_unused_constrs ( ) const

This method returns the number of unused constraints.

10.206.4.40 size_t coco::model::n_unused_objs ( ) const

This method returns the number of unused objectives.

10.206.4.41 size_t coco::model::n_unused_vars ( ) const

This method returns the number of unused variables.

10.206.4.42 walker coco::model::nary (const std::vector< walker > & *_op*, int *expr_type*, const std::vector< double > & *_coeffs* = `std::vector< double >()`)

This method constructs a new n-ary operation with operand list `_op`, multiplicative coefficient list `_coeffs`, and expression type `expr_type`.

10.206.4.43 walker coco::model::nary (const std::vector< walker > & *_op*, int *expr_type*, const basic_alltype & *_params*, const std::vector< double > & *_coeffs* = `std::vector< double >()`)

This method constructs a new n-ary operation with operand list `_op`, multiplicative coefficient list `_coeffs`, expression type `expr_type`, and additional expression parameters `_params`.

10.206.4.44 void coco::model::new_variables (int *_new_num_of_vars*)

This method sets the number of variables in the model to `_new_num_of_vars` and communicates the change to all models below the same model_iddata.

10.206.4.45 int coco::model::next_constraint_num ( )

This method returns the next free constraint number.

10.206.4.46 int coco::model::next_num ( )

This method returns the next free node number.

10.206.4.47 int coco::model::next_objective_num ( )

This method returns the next free objective number.

10.206.4.48 int coco::model::next_variable_num ( )

This method returns the next free variable number.

10.206.4.49 const walker& coco::model::node (unsigned int *i*) const

A call to this method returns a walker to node `i`.

10.206.4.50 unsigned int coco::model::number_of_constraints ( ) const

The number_of_constraints method returns the number of constraints across all models governed by the same model_iddata.

10.206.4.51 unsigned int coco::model::number_of_managed_constraints ( ) const [inline]

This method returns the number of constraints managed by this model class (the number of constraints in this DAG.

Definition at line 201 of file model.h.

10.206.4.52 unsigned int coco::model::number_of_managed_nodes ( ) const [inline]

This method returns the number of nodes managed by this model class (the number of nodes in this DAG.

Definition at line 189 of file model.h.

10.206.4.53 unsigned int coco::model::number_of_managed_objectives ( ) const [inline]

This method returns the number of objectives managed by this model class (the number of objectives in this DAG.

Definition at line 197 of file model.h.

10.206.4.54 unsigned int coco::model::number_of_managed_variables ( ) const [inline]

This method returns the number of variables managed by this model class (the number of variables in this DAG.

Definition at line 193 of file model.h.

10.206.4.55 unsigned int coco::model::number_of_nodes ( ) const

The number_of_nodes method returns the number of nodes across all models governed by the same model_iddata.

10.206.4.56 unsigned int coco::model::number_of_objectives ( ) const

The number_of_objectives method returns the number of objectives across all models governed by the same model_iddata.

10.206.4.57 unsigned int coco::model::number_of_variables ( ) const

The number_of_variables method returns the number of variables across all models governed by the same model_iddata.

10.206.4.58 double coco::model::obj_adj (unsigned int *n*) const

This method returns the additive constant in the objective function n.

10.206.4.59 double coco::model::obj_mult (unsigned int *n*) const

This method returns the scalar multiplier of the objective function n.

10.206.4.60 const std::string coco::model::obj_name (unsigned int *n*) const

This method returns the name of the objective function n.

10.206.4.61 const walker& coco::model::objective (unsigned int *i*) const

A call to this method returns a walker to objective i.

10.206.4.62 const std::map<int, std::vector<double> >& coco::model::optima ( ) const

This method returns the known optima defined. The return value is a map of known optima numbers to coordinate vectors.

10.206.4.63 void coco::model::prepare_kill ( ) [inline]

This method takes care that upon deletion everything goes away.

Definition at line 373 of file model.h.

10.206.4.64 void coco::model::remove_node (const walker & *_w*, unsigned int *_nnum*)

The remove_node methods remove the node, specified by its walker _w, its node number _nnum, or by both.

10.206.4.65 void coco::model::remove_node (const walker & *_w*)

The remove_node methods remove the node, specified by its walker _w, its node number _nnum, or by both.

10.206.4.66 void coco::model::remove_node (unsigned int *__node_num*)

The remove_node methods remove the node, specified by its walker _w, its node number _nnum, or by both.

10.206.4.67 void coco::model::renumber_constraints ( )

If this method is called, the constraints are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed.

10.206.4.68 void coco::model::renumber_variables ( )

If this method is called, the variables are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed.

10.206.4.69 void coco::model::set_counters ( )

This method initializes the `n_parents` and `n_children` members of all nodes.

See also

expression_node.

10.206.4.70 bool coco::model::simplify_thin ( )

This simplifier replaces all thin nodes by corresponding constants and simplifies the DAG accordingly.

Bug The simplify_thin method does not yet work properly.

10.206.4.71 walker coco::model::store_constraint (const walker & *_w*)

This method stores a new constraint in all model data structures

10.206.4.72 walker coco::model::store_ghost (const walker & *_w*)

This method stores a new ghost in all model data structures

10.206.4.73 walker coco::model::store_node (const walker & *_w*)

This method stores a new node in all model data structures

10.206.4.74 walker coco::model::store_objective (const walker & *_w*)

This method stores a new objective in all model data structures

10.206.4.75 walker coco::model::store_variable (const walker & *_w*)

This method stores a new variable in all model data structures

10.206.4.76 const std::map<int, std::vector<double> >& coco::model::trace_points ( ) const

This method returns the trace points defined. The return value is a map of trace point numbers to coordinate vectors.

10.206.4.77 walker coco::model::unary (const walker & *_op1*, int *expr_type*, double *_coeff* = `1.0`)

This method constructs a new unary operation with operand `_op1`, multiplicative coefficient `_coeff`, and expression type `expr_type`.

10.206.4.78 walker coco::model::unary (const walker & *_op1,* int *expr_type,* const basic_alltype & *_params,* double *_coeff* = `1.0`)

This method constructs a new unary operation with operand `_op1`, multiplicative coefficient `_coeff`, expression type `expr_type`, and additional expression parameters `_params`.

10.206.4.79 const std::string& coco::model::unused_constr (unsigned int *n*) const

This method returns the name of the unused constraint `n`.

10.206.4.80 const std::string& coco::model::unused_obj (unsigned int *n*) const

This method returns the name of the unused objectives `n`.

10.206.4.81 const std::string& coco::model::unused_var (unsigned int *n*) const

This method returns the name of the unused variable `n`.

10.206.4.82 const walker& coco::model::var (unsigned int *i*) const

A call to this method returns a walker to variable `i`.

10.206.4.83 const std::string coco::model::var_name (unsigned int *n*) const

This method returns the name of the variable `n`.

10.206.4.84 walker coco::model::variable (unsigned int *_vnum*)

This method constructs a new variable with variable number `_vnum`.

10.206.4.85 walker coco::model::vnary (int *expr_type, ...*)

This method constructs a new n-ary operation with expression type `expr_type`. The operands are specified as the additional parameters and need to be of type `walker`.

10.206.4.86 walker coco::model::vxnary (const basic_alltype & *_params,* int *expr_type, ...*)

This method constructs a new n-ary operation with expression type `expr_type` and additional expression parameters `_params`. The operands and multiplicative constants are specified as the additional parameters and need to be of type `walker` followed by a double, and so on. The final double (in case it is 1.0) can be left out.

10.206.4.87 void coco::model::write (std::ostream & *__o* = `std::cout`) const

This method writes the DAG in .dag format to the ostream `__o`.

10.206.5 Friends And Related Function Documentation

10.206.5.1 friend class dag_delta [friend]

Definition at line 604 of file model.h.

10.206.5.2 friend class dag_undelta [friend]

Definition at line 605 of file model.h.

10.206.6 Member Data Documentation

10.206.6.1 std::vector<walker> coco::model::constraints

This is a vector of walkers pointing to the result nodes of the constraints.

Definition at line 114 of file model.h.

10.206.6.2 bool coco::model::has_read_errors

This member is set to if an I/O exception was raised during model reading.

Definition at line 117 of file model.h.

10.206.6.3 vmtl::sparse_matrix<double> coco::model::lin

The `lin` member is the matrix of linear constraints.

Definition at line 93 of file model.h.

10.206.6.4 std::vector<vmtl::sparse_matrix<double> > coco::model::matd

The `matd` member is a vector of double matrices.

Definition at line 95 of file model.h.

10.206.6.5 std::vector<vmtl::sparse_matrix<interval> > coco::model::mati

The `matd` member is a vector of interval matrices.

Definition at line 97 of file model.h.

10.206.6.6 std::vector<walker> coco::model::objectives

This is a vector of walkers pointing to the result nodes of the objective functions.

Definition at line 111 of file model.h.

10.206.6.7 std::vector<int> coco::model::ocoeff

These objective coefficients are used to transfer any constraint satisfaction problem or optimization problem to a vector valued minimization problem. The values are per objective

1	for a minimization problem
-1	for a maximization problem
0	for a constraint satisfaction problem

Definition at line 108 of file model.h.

The documentation for this class was generated from the following file:

- model.h

10.207 coco::model_gid Class Reference

Model Group Data Class (middle class in the model hierarchy)

`#include <model.h>`

Public Member Functions

- void remove_node_ref (unsigned int _n)
- void remove_var_ref (unsigned int _n)
- void remove_obj_ref (unsigned int _n)
- void remove_const_ref (unsigned int _n)
- void set_glob_ref (const std::vector< model::walker > &node_ref)
- void set_gvar_ref (const std::vector< model::walker > &var_ref)
- model_gid (model &__m, model_iddata *__i=NULL)
- model_gid (model &__m, unsigned int n, model_iddata *__i=NULL)
- model_gid (model &__mr, const model_gid &__m)
- ~model_gid ()

Methods for accessing model group information

These methods can be used for accessing important information about a model group and its nodes, variables, constraints,...

- unsigned int number_of_nodes () const
- unsigned int number_of_variables () const
- unsigned int number_of_objectives () const
- unsigned int number_of_constraints () const
- void number_of_nodes (unsigned int _n)
- void number_of_variables (unsigned int _n)
- void number_of_objectives (unsigned int _n)
- void number_of_constraints (unsigned int _n)
- void mk_globref (unsigned int n, const model::walker &__w)
- void mk_gvarref (unsigned int n, const model::walker &__w)
- void mk_gobjref (unsigned int n, const model::walker &__w)
- void mk_gconstref (unsigned int n, const model::walker &__w)
- void make_obj_back_ref (unsigned int node, unsigned int onum)
- void make_const_back_ref (unsigned int node, unsigned int cnum)
- unsigned int get_node_id ()
- void remove_node_id (unsigned int n)
- unsigned int get_var_id ()
- void remove_var_id (unsigned int n)
- unsigned int get_obj_id ()
- void remove_obj_id (unsigned int n)
- unsigned int get_const_id ()
- void remove_const_id (unsigned int n)
- void compress_numbers (bool renumber_vars=false, bool renumber_obj=false, bool renumber_-const=false)
- const model::walker & node (unsigned int i) const
- const model::walker & variable (unsigned int i) const
- const model::walker & objective (unsigned int i) const
- const model::walker & constraint (unsigned int i) const

- bool empty (const model::walker &__x) const
- bool its_me (const model &__m) const
- model::walker empty_reference () const
- bool have_glob_ref (unsigned int _nnum) const
- bool have_gvar_ref (unsigned int _vnum) const
- bool have_gobj_ref (unsigned int _onum) const
- bool have_gconst_ref (unsigned int _cnum) const

Methods for handling modeling information

These methods return names of variables and constants and other data which is irrelevant for the solution process but necessary for user interaction.

- const std::string model_name () const
- void model_name (const std::string &n)
- void model_name (const char *n)
- const std::string var_name (unsigned int n) const
- void var_name (unsigned int n, const std::string &vn)
- void var_name (unsigned int n, const char *vn)
- const std::string obj_name (unsigned int n) const
- void obj_name (unsigned int n, const std::string &vn)
- void obj_name (unsigned int n, const char *vn)
- bool get_obj_num (unsigned int node_num, unsigned int &obj_num)
- const std::string const_name (unsigned int n) const
- void const_name (unsigned int n, const std::string &vn)
- void const_name (unsigned int n, const char *vn)
- bool get_const_num (unsigned int node_num, unsigned int &const_num)
- double obj_adj (unsigned int n) const
- void obj_adj (unsigned int n, double adj)
- double obj_mult (unsigned int n) const
- void obj_mult (unsigned int n, double mult)
- size_t n_fixed_vars () const
- std::pair< const std::string, double > fixed_var (unsigned int n) const
- void fixed_var (const std::string &vn, double val)
- void fixed_var (const char *vn, double val)
- size_t n_unused_vars () const
- const std::string & unused_var (unsigned int n) const
- void unused_var (const std::string &vn)
- void unused_var (const char *vn)
- size_t n_unused_objs () const
- const std::string & unused_obj (unsigned int n) const
- void unused_obj (const std::string &vn)
- void unused_obj (const char *vn)
- size_t n_unused_constrs () const
- const std::string & unused_constr (unsigned int n) const
- void unused_constr (const std::string &vn)
- void unused_constr (const char *vn)
- const std::map< int, std::vector< double > > & trace_points () const
- const std::map< int, std::vector< double > > & optima () const
- void add_trace_point (int nr, const std::vector< double > &d)
- void add_optimum (int nr, const std::vector< double > &d)

Friends

- class model_iddata

10.207.1 Detailed Description

Models are grouped into sets, which can be directly combined with each other, like models and model deltas. In this class global references to nodes, variables, constants, and back references from constraint numbers to constraints are managed. Furthermore, the reference model, the main model to which all the deltas belong, is stored.

10.207.2 Constructor & Destructor Documentation

10.207.2.1 coco::model_gid::model_gid (model & __m, model_iddata * __i = NULL)

Constructor, which also builds a new model_iddata structure, unless __i is non-NULL. The parameter __m sets the reference model.

10.207.2.2 coco::model_gid::model_gid (model & __m, unsigned int n, model_iddata * __i = NULL)

Constructor, which also builds a new model_iddata structure, unless __i is non-NULL. The parameter __m sets the reference model, while n presets the number of variables.

10.207.2.3 coco::model_gid::model_gid (model & __mr, const model_gid & __m)

Copy Constructor, which copies the model_gid __m but changes the reference model to __mr.

10.207.2.4 coco::model_gid::~model_gid ( ) [inline]

Standard Destructor

Definition at line 969 of file model.h.

10.207.3 Member Function Documentation

10.207.3.1 void coco::model_gid::add_optimum (int nr, const std::vector< double > & d) [inline]

A call to this method adds the known optimum nr with coordinates d to the list of known optima.

Definition at line 1216 of file model.h.

10.207.3.2 void coco::model_gid::add_trace_point (int nr, const std::vector< double > & d) [inline]

A call to this method adds the trace point nr with coordinates d to the list of trace points.

Definition at line 1212 of file model.h.

10.207.3.3 void coco::model_gid::compress_numbers (bool renumber_vars = false, bool renumber_obj = false, bool renumber_const = false) [inline]

If this method is called, the nodes, variables, objectives, and constraints are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed. The parameters renumber_vars, renumber_obj, and renumber_const determine whether the variable and constraint numbers, respectively, are renumbered in addition to the node numbers.

Definition at line 1056 of file model.h.

10.207.3.4 const std::string coco::model_gid::const_name (unsigned int *n*) const [inline]

This method returns the name of the constraint n.

Definition at line 1136 of file model.h.

10.207.3.5 void coco::model_gid::const_name (unsigned int *n,* const std::string & *vn*) [inline]

A call to this method sets the name of the constraint n to cn.

Definition at line 1139 of file model.h.

10.207.3.6 void coco::model_gid::const_name (unsigned int *n,* const char * *vn*) [inline]

A call to this method sets the name of the constraint n to cn.

Definition at line 1142 of file model.h.

10.207.3.7 const model::walker& coco::model_gid::constraint (unsigned int *i*) const [inline]

A call to this method returns a walker to constraint i.

Definition at line 1068 of file model.h.

10.207.3.8 bool coco::model_gid::empty (const model::walker & *__x*) const [inline]

This method checks whether the walker ___x points to an empty reference, i.e. to a deleted node.

Definition at line 1073 of file model.h.

10.207.3.9 model::walker coco::model_gid::empty_reference ( ) const [inline]

This method returns an empty reference. This is, e.g., stored in global reference arrays whenever a node is deleted.

Definition at line 1080 of file model.h.

10.207.3.10 std::pair<const std::string, double> coco::model_gid::fixed_var (unsigned int *n*) const [inline]

This method returns the name and value of the fixed variable n.

Definition at line 1166 of file model.h.

10.207.3.11 void coco::model_gid::fixed_var (const std::string & *vn,* double *val*) [inline]

This method stores the fixed variable with name vn and value val.

Definition at line 1169 of file model.h.

10.207.3.12 void coco::model_gid::fixed_var (const char * *vn,* double *val*) [inline]

This method stores the fixed variable with name vn and value val.

Definition at line 1171 of file model.h.

10.207.3.13 unsigned int coco::model_gid::get_const_id ( ) [inline]

This method returns a free constraint number and removes it from the list of free constraint numbers.

Definition at line 1044 of file model.h.

10.207.3.14 bool coco::model_gid::get_const_num (unsigned int *node_num,* unsigned int & *const_num*)

The get_const_num method assigns to `const_num` the constraint number associated with the constraint, whose result node has node number `node_num`. The return value is `true`, if there is a constraint associated with node `node_num`.

10.207.3.15 unsigned int coco::model_gid::get_node_id ( ) [inline]

This method returns a free node number a removes it from the list of free node numbers.

Definition at line 1026 of file model.h.

10.207.3.16 unsigned int coco::model_gid::get_obj_id ( ) [inline]

This method returns a free objective number and removes it from the list of free objective numbers.

Definition at line 1038 of file model.h.

10.207.3.17 bool coco::model_gid::get_obj_num (unsigned int *node_num,* unsigned int & *obj_num*)

The get_obj_num method assigns to `obj_num` the objective number associated with the objective, whose result node has node number `node_num`. The return value is `true`, if there is a objective associated with node `node_num`.

10.207.3.18 unsigned int coco::model_gid::get_var_id ( ) [inline]

This method returns a free variable number a removes it from the list of free variable numbers.

Definition at line 1032 of file model.h.

10.207.3.19 bool coco::model_gid::have_gconst_ref (unsigned int *_cnum*) const [inline]

This method checks whether a global constraint reference exists for constraint number `_cnum`.

Definition at line 1096 of file model.h.

10.207.3.20 bool coco::model_gid::have_glob_ref (unsigned int *_nnum*) const [inline]

This method checks whether a global node reference exists for node number `_nnum`.

Definition at line 1084 of file model.h.

10.207.3.21 bool coco::model_gid::have_gobj_ref (unsigned int *_onum*) const [inline]

This method checks whether a global objective reference exists for objective number `_onum`.

Definition at line 1092 of file model.h.

10.207.3.22 bool coco::model_gid::have_gvar_ref (unsigned int _*vnum*) const [inline]

This method checks whether a global variable reference exists for variable number _nnum.

Definition at line 1088 of file model.h.

10.207.3.23 bool coco::model_gid::its_me (const model & __*m*) const [inline]

The its_me method tests if the model __m is the reference model for this model group.

Definition at line 1077 of file model.h.

10.207.3.24 void coco::model_gid::make_const_back_ref (unsigned int *node*, unsigned int *cnum*)

This method constructs a constraint back reference from node number node to constraint number cnum.

10.207.3.25 void coco::model_gid::make_obj_back_ref (unsigned int *node*, unsigned int *onum*)

This method constructs a objective back reference from node number node to objective number cnum.

10.207.3.26 void coco::model_gid::mk_gconstref (unsigned int *n*, const model::walker & __*w*)

This method constructs a global constraint reference to __w with number n.

10.207.3.27 void coco::model_gid::mk_globref (unsigned int *n*, const model::walker & __*w*)

This method constructs a global node reference to __w with number n.

10.207.3.28 void coco::model_gid::mk_gobjref (unsigned int *n*, const model::walker & __*w*)

This method constructs a global objective reference to __w with number n.

10.207.3.29 void coco::model_gid::mk_gvarref (unsigned int *n*, const model::walker & __*w*)

This method constructs a global variable reference to __w with number n.

10.207.3.30 const std::string coco::model_gid::model_name ( ) const [inline]

This method returns the name of the model.

Definition at line 1106 of file model.h.

10.207.3.31 void coco::model_gid::model_name (const std::string & *n*) [inline]

This method sets the name of the model to n.

Definition at line 1108 of file model.h.

10.207.3.32 void coco::model_gid::model_name (const char * *n*) [inline]

This method sets the name of the model to n.

Definition at line 1110 of file model.h.

10.207.3.33 size_t coco::model_gid::n_fixed_vars ( ) const [inline]

This method returns the number of variables fixed by the presolve.

Definition at line 1164 of file model.h.

10.207.3.34 size_t coco::model_gid::n_unused_constrs ( ) const [inline]

This method returns the number of unused constraints.

Definition at line 1195 of file model.h.

10.207.3.35 size_t coco::model_gid::n_unused_objs ( ) const [inline]

This method returns the number of unused objectives.

Definition at line 1185 of file model.h.

10.207.3.36 size_t coco::model_gid::n_unused_vars ( ) const [inline]

This method returns the number of unused variables.

Definition at line 1175 of file model.h.

10.207.3.37 const model::walker& coco::model_gid::node (unsigned int *i*) const [inline]

A call to this method returns a walker to node `i`.

Definition at line 1061 of file model.h.

10.207.3.38 unsigned int coco::model_gid::number_of_constraints ( ) const [inline]

The number_of_constraints method returns the number of constraints across all models governed by the same model_iddata.

Definition at line 989 of file model.h.

10.207.3.39 void coco::model_gid::number_of_constraints (unsigned int *_n*) [inline]

This method sets the number of constraints across all model groups in the same model_iddata.

Definition at line 1002 of file model.h.

10.207.3.40 unsigned int coco::model_gid::number_of_nodes ( ) const [inline]

The number_of_nodes method returns the number of nodes across all models governed by the same model_iddata.

Definition at line 978 of file model.h.

10.207.3.41 void coco::model_gid::number_of_nodes (unsigned int *_n*) [inline]

This method sets the number of nodes across all model groups in the same model_iddata.

Definition at line 993 of file model.h.

10.207.3.42 unsigned int coco::model_gid::number_of_objectives ( ) const [inline]

The number_of_objectives method returns the number of objectives across all models governed by the same model_iddata.

Definition at line 985 of file model.h.

10.207.3.43 void coco::model_gid::number_of_objectives (unsigned int _n) [inline]

This method sets the number of objectives across all model groups in the same model_iddata.

Definition at line 999 of file model.h.

10.207.3.44 unsigned int coco::model_gid::number_of_variables ( ) const [inline]

The number_of_variables method returns the number of variables across all models governed by the same model_iddata.

Definition at line 981 of file model.h.

10.207.3.45 void coco::model_gid::number_of_variables (unsigned int _n) [inline]

This method sets the number of variables across all model groups in the same model_iddata.

Definition at line 996 of file model.h.

10.207.3.46 double coco::model_gid::obj_adj (unsigned int n) const [inline]

This method returns the additive constant in the objective function n.

Definition at line 1152 of file model.h.

10.207.3.47 void coco::model_gid::obj_adj (unsigned int n, double adj) [inline]

With this method the additive constant in the objective function n is set to adj.

Definition at line 1155 of file model.h.

10.207.3.48 double coco::model_gid::obj_mult (unsigned int n) const [inline]

This method returns the scalar multiplier of the objective function n.

Definition at line 1158 of file model.h.

10.207.3.49 void coco::model_gid::obj_mult (unsigned int n, double mult) [inline]

With this method the scalar multiplier of the objective function n is set to mult.

Definition at line 1161 of file model.h.

10.207.3.50 const std::string coco::model_gid::obj_name (unsigned int n) const [inline]

This method returns the name of the objective n.

Definition at line 1121 of file model.h.

10.207.3.51 void coco::model_gid::obj_name (unsigned int *n*, const std::string & *vn*) [inline]

A call to this method sets the name of the objective n to cn.

Definition at line 1124 of file model.h.

10.207.3.52 void coco::model_gid::obj_name (unsigned int *n*, const char * *vn*) [inline]

A call to this method sets the name of the objective n to cn.

Definition at line 1127 of file model.h.

10.207.3.53 const model::walker& coco::model_gid::objective (unsigned int *i*) const [inline]

A call to this method returns a walker to objective i.

Definition at line 1065 of file model.h.

10.207.3.54 const std::map<int, std::vector<double> >& coco::model_gid::optima ( ) const

This method returns the known optima defined. The return value is a map of known optima numbers to coordinate vectors.

10.207.3.55 void coco::model_gid::remove_const_id (unsigned int *n*) [inline]

This method inserts the constraint number n in the list of free constraint numbers.

Definition at line 1047 of file model.h.

10.207.3.56 void coco::model_gid::remove_const_ref (unsigned int *_n*)

This method removes the global constraint reference of node _n.

10.207.3.57 void coco::model_gid::remove_node_id (unsigned int *n*) [inline]

This method inserts the node number n in the list of free node numbers.

Definition at line 1029 of file model.h.

10.207.3.58 void coco::model_gid::remove_node_ref (unsigned int *_n*) [inline]

This method removes the global node reference of node _n.

Definition at line 940 of file model.h.

10.207.3.59 void coco::model_gid::remove_obj_id (unsigned int *n*) [inline]

This method inserts the objective number n in the list of free objective numbers.

Definition at line 1041 of file model.h.

10.207.3.60 void coco::model_gid::remove_obj_ref (unsigned int *_n*)

This method removes the global objective reference of node _n.

10.207.3.61 void coco::model_gid::remove_var_id (unsigned int *n*) [inline]

This method inserts the variable number n in the list of free variable numbers.

Definition at line 1035 of file model.h.

10.207.3.62 void coco::model_gid::remove_var_ref (unsigned int *_n*) [inline]

This method removes the global variable reference of node _n.

Definition at line 944 of file model.h.

10.207.3.63 void coco::model_gid::set_glob_ref (const std::vector< model::walker > & *node_ref*)

This method initializes the glob_ref array and the maximal node number.

10.207.3.64 void coco::model_gid::set_gvar_ref (const std::vector< model::walker > & *var_ref*)

This method initializes the glob_ref array and the maximal node number.

10.207.3.65 const std::map<int, std::vector<double> >& coco::model_gid::trace_points ( ) const

This method returns the trace points defined. The return value is a map of trace point numbers to coordinate vectors.

10.207.3.66 const std::string& coco::model_gid::unused_constr (unsigned int *n*) const [inline]

This method returns the name of the unused constraint n.

Definition at line 1197 of file model.h.

10.207.3.67 void coco::model_gid::unused_constr (const std::string & *vn*) [inline]

This method stores the unused constraint with name vn.

Definition at line 1200 of file model.h.

10.207.3.68 void coco::model_gid::unused_constr (const char * *vn*) [inline]

This method stores the unused constraint with name vn.

Definition at line 1202 of file model.h.

10.207.3.69 const std::string& coco::model_gid::unused_obj (unsigned int *n*) const [inline]

This method returns the name of the unused objective n.

Definition at line 1187 of file model.h.

10.207.3.70 void coco::model_gid::unused_obj (const std::string & *vn*) [inline]

This method stores the unused objective with name vn.

Definition at line 1190 of file model.h.

10.207.3.71 void coco::model_gid::unused_obj (const char * *vn*) [inline]

This method stores the unused objective with name vn.

Definition at line 1192 of file model.h.

10.207.3.72 const std::string& coco::model_gid::unused_var (unsigned int *n*) const [inline]

This method returns the name of the unused variable n.

Definition at line 1177 of file model.h.

10.207.3.73 void coco::model_gid::unused_var (const std::string & *vn*) [inline]

This method stores the unused variable with name vn.

Definition at line 1180 of file model.h.

10.207.3.74 void coco::model_gid::unused_var (const char * *vn*) [inline]

This method stores the unused variable with name vn.

Definition at line 1182 of file model.h.

10.207.3.75 const std::string coco::model_gid::var_name (unsigned int *n*) const [inline]

This method returns the name of the variable n.

Definition at line 1113 of file model.h.

10.207.3.76 void coco::model_gid::var_name (unsigned int *n,* const std::string & *vn*) [inline]

A call to this method sets the name of the variable n to vn.

Definition at line 1115 of file model.h.

10.207.3.77 void coco::model_gid::var_name (unsigned int *n,* const char * *vn*) [inline]

A call to this method sets the name of the variable n to vn.

Definition at line 1117 of file model.h.

10.207.3.78 const model::walker& coco::model_gid::variable (unsigned int *i*) const [inline]

A call to this method returns a walker to variable i.

Definition at line 1063 of file model.h.

10.207.4 Friends And Related Function Documentation

10.207.4.1 friend class model_iddata [friend]

Definition at line 1220 of file model.h.

The documentation for this class was generated from the following file:

- model.h

10.208 coco::coco::model_gid Class Reference

Model Group Data Class (middle class in the model hierarchy)

Public Member Functions

- void remove_node_ref (unsigned int _n)
- void remove_var_ref (unsigned int _n)
- void remove_obj_ref (unsigned int _n)
- void remove_const_ref (unsigned int _n)
- void set_glob_ref (const std::vector< model::walker > &node_ref)
- void set_gvar_ref (const std::vector< model::walker > &var_ref)
- model_gid (model &__m, model_iddata *__i=NULL)
- model_gid (model &__m, unsigned int n, model_iddata *__i=NULL)
- model_gid (model &__mr, const model_gid &__m)
- ~model_gid ()

Methods for accessing model group information

These methods can be used for accessing important information about a model group and its nodes, variables, constraints,...

- unsigned int number_of_nodes () const
- unsigned int number_of_variables () const
- unsigned int number_of_objectives () const
- unsigned int number_of_constraints () const
- void number_of_nodes (unsigned int _n)
- void number_of_variables (unsigned int _n)
- void number_of_objectives (unsigned int _n)
- void number_of_constraints (unsigned int _n)
- void mk_globref (unsigned int n, const model::walker &__w)
- void mk_gvarref (unsigned int n, const model::walker &__w)
- void mk_gobjref (unsigned int n, const model::walker &__w)
- void mk_gconstref (unsigned int n, const model::walker &__w)
- void make_obj_back_ref (unsigned int node, unsigned int onum)
- void make_const_back_ref (unsigned int node, unsigned int cnum)
- unsigned int get_node_id ()
- void remove_node_id (unsigned int n)
- unsigned int get_var_id ()
- void remove_var_id (unsigned int n)
- unsigned int get_obj_id ()
- void remove_obj_id (unsigned int n)
- unsigned int get_const_id ()
- void remove_const_id (unsigned int n)
- void compress_numbers (bool renumber_vars=false, bool renumber_obj=false, bool renumber_-const=false)
- const model::walker & node (unsigned int i) const
- const model::walker & variable (unsigned int i) const
- const model::walker & objective (unsigned int i) const
- const model::walker & constraint (unsigned int i) const
- bool empty (const model::walker &__x) const
- bool its_me (const model &__m) const

- model::walker empty_reference () const
- bool have_glob_ref (unsigned int _nnum) const
- bool have_gvar_ref (unsigned int _vnum) const
- bool have_gobj_ref (unsigned int _onum) const
- bool have_gconst_ref (unsigned int _cnum) const

Methods for handling modeling information

These methods return names of variables and constants and other data which is irrelevant for the solution process but necessary for user interaction.

- const std::string model_name () const
- void model_name (const std::string &n)
- void model_name (const char *n)
- const std::string var_name (unsigned int n) const
- void var_name (unsigned int n, const std::string &vn)
- void var_name (unsigned int n, const char *vn)
- const std::string obj_name (unsigned int n) const
- void obj_name (unsigned int n, const std::string &vn)
- void obj_name (unsigned int n, const char *vn)
- bool get_obj_num (unsigned int node_num, unsigned int &obj_num)
- const std::string const_name (unsigned int n) const
- void const_name (unsigned int n, const std::string &vn)
- void const_name (unsigned int n, const char *vn)
- bool get_const_num (unsigned int node_num, unsigned int &const_num)
- double obj_adj (unsigned int n) const
- void obj_adj (unsigned int n, double adj)
- double obj_mult (unsigned int n) const
- void obj_mult (unsigned int n, double mult)
- size_t n_fixed_vars () const
- std::pair< const std::string, double > fixed_var (unsigned int n) const
- void fixed_var (const std::string &vn, double val)
- void fixed_var (const char *vn, double val)
- size_t n_unused_vars () const
- const std::string & unused_var (unsigned int n) const
- void unused_var (const std::string &vn)
- void unused_var (const char *vn)
- size_t n_unused_objs () const
- const std::string & unused_obj (unsigned int n) const
- void unused_obj (const std::string &vn)
- void unused_obj (const char *vn)
- size_t n_unused_constrs () const
- const std::string & unused_constr (unsigned int n) const
- void unused_constr (const std::string &vn)
- void unused_constr (const char *vn)
- const std::map< int, std::vector< double > > & trace_points () const
- const std::map< int, std::vector< double > > & optima () const
- void add_trace_point (int nr, const std::vector< double > &d)
- void add_optimum (int nr, const std::vector< double > &d)

Friends

- class model_iddata

10.208.1 Detailed Description

Models are grouped into sets, which can be directly combined with each other, like models and model deltas. In this class global references to nodes, variables, constants, and back references from constraint numbers to constraints are managed. Furthermore, the reference model, the main model to which all the deltas belong, is stored.

10.208.2 Constructor & Destructor Documentation

10.208.2.1 coco::model_gid::model_gid (model & __m, model_iddata * __i = NULL) [inline]

Constructor, which also builds a new model_iddata structure, unless __i is non-NULL. The parameter __m sets the reference model.

Definition at line 394 of file model.hpp.

10.208.2.2 coco::model_gid::model_gid (model & __m, unsigned int n, model_iddata * __i = NULL) [inline]

Constructor, which also builds a new model_iddata structure, unless __i is non-NULL. The parameter __m sets the reference model, while n presets the number of variables.

Definition at line 405 of file model.hpp.

10.208.2.3 coco::model_gid::model_gid (model & __mr, const model_gid & __m) [inline]

Copy Constructor, which copies the model_gid __m but changes the reference model to __mr.

Definition at line 416 of file model.hpp.

10.208.2.4 coco::coco::model_gid::~model_gid ( ) [inline]

Standard Destructor

Definition at line 969 of file search_graph.cc.

10.208.3 Member Function Documentation

10.208.3.1 void coco::coco::model_gid::add_optimum (int nr, const std::vector< double > & d) [inline]

A call to this method adds the known optimum nr with coordinates d to the list of known optima.

Definition at line 1216 of file search_graph.cc.

10.208.3.2 void coco::coco::model_gid::add_trace_point (int nr, const std::vector< double > & d) [inline]

A call to this method adds the trace point nr with coordinates d to the list of trace points.

Definition at line 1212 of file search_graph.cc.

10.208.3.3 void coco::coco::model_gid::compress_numbers (bool *renumber_vars* = false, bool *renumber_obj* = false, bool *renumber_const* = false) [inline]

If this method is called, the nodes, variables, objectives, and constraints are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed. The parameters renumber_vars, renumber_obj, and renumber_const determine whether the variable and constraint numbers, respectively, are renumbered in addition to the node numbers.

Definition at line 1056 of file search_graph.cc.

10.208.3.4 const std::string coco::coco::model_gid::const_name (unsigned int *n*) const [inline]

This method returns the name of the constraint n.

Definition at line 1136 of file search_graph.cc.

10.208.3.5 void coco::coco::model_gid::const_name (unsigned int *n*, const std::string & *vn*) [inline]

A call to this method sets the name of the constraint n to cn.

Definition at line 1139 of file search_graph.cc.

10.208.3.6 void coco::coco::model_gid::const_name (unsigned int *n*, const char * *vn*) [inline]

A call to this method sets the name of the constraint n to cn.

Definition at line 1142 of file search_graph.cc.

10.208.3.7 const model::walker& coco::coco::model_gid::constraint (unsigned int *i*) const [inline]

A call to this method returns a walker to constraint i.

Definition at line 1068 of file search_graph.cc.

10.208.3.8 bool coco::coco::model_gid::empty (const model::walker & *__x*) const [inline]

This method checks whether the walker ___x points to an empty reference, i.e. to a deleted node.

Definition at line 1073 of file search_graph.cc.

10.208.3.9 model::walker coco::coco::model_gid::empty_reference ( ) const [inline]

This method returns an empty reference. This is, e.g., stored in global reference arrays whenever a node is deleted.

Definition at line 1080 of file search_graph.cc.

10.208.3.10 std::pair<const std::string, double> coco::coco::model_gid::fixed_var (unsigned int *n*) const [inline]

This method returns the name and value of the fixed variable n.

Definition at line 1166 of file search_graph.cc.

10.208.3.11 void coco::coco::model_gid::fixed_var (const std::string & *vn,* double *val*) [inline]

This method stores the fixed variable with name vn and value val.

Definition at line 1169 of file search_graph.cc.

10.208.3.12 void coco::coco::model_gid::fixed_var (const char * *vn,* double *val*) [inline]

This method stores the fixed variable with name vn and value val.

Definition at line 1171 of file search_graph.cc.

10.208.3.13 unsigned int coco::coco::model_gid::get_const_id ( ) [inline]

This method returns a free constraint number and removes it from the list of free constraint numbers.

Definition at line 1044 of file search_graph.cc.

10.208.3.14 bool coco::model_gid::get_const_num (unsigned int *node_num,* unsigned int & *const_num*) [inline]

The get_const_num method assigns to const_num the constraint number associated with the constraint, whose result node has node number node_num. The return value is true, if there is a constraint associated with node node_num.

Definition at line 665 of file model.hpp.

10.208.3.15 unsigned int coco::coco::model_gid::get_node_id ( ) [inline]

This method returns a free node number a removes it from the list of free node numbers.

Definition at line 1026 of file search_graph.cc.

10.208.3.16 unsigned int coco::coco::model_gid::get_obj_id ( ) [inline]

This method returns a free objective number and removes it from the list of free objective numbers.

Definition at line 1038 of file search_graph.cc.

10.208.3.17 bool coco::model_gid::get_obj_num (unsigned int *node_num,* unsigned int & *obj_num*) [inline]

The get_obj_num method assigns to obj_num the objective number associated with the objective, whose result node has node number node_num. The return value is true, if there is a objective associated with node node_num.

Definition at line 623 of file model.hpp.

10.208.3.18 unsigned int coco::coco::model_gid::get_var_id ( ) [inline]

This method returns a free variable number a removes it from the list of free variable numbers.

Definition at line 1032 of file search_graph.cc.

10.208.3.19 bool coco::coco::model_gid::have_gconst_ref (unsigned int *_cnum*) const [inline]

This method checks whether a global constraint reference exists for constraint number _cnum.

Definition at line 1096 of file search_graph.cc.

10.208.3.20 bool coco::coco::model_gid::have_glob_ref (unsigned int *_nnum*) const [inline]

This method checks whether a global node reference exists for node number _nnum.

Definition at line 1084 of file search_graph.cc.

10.208.3.21 bool coco::coco::model_gid::have_gobj_ref (unsigned int *_onum*) const [inline]

This method checks whether a global objective reference exists for objective number _onum.

Definition at line 1092 of file search_graph.cc.

10.208.3.22 bool coco::coco::model_gid::have_gvar_ref (unsigned int *_vnum*) const [inline]

This method checks whether a global variable reference exists for variable number _nnum.

Definition at line 1088 of file search_graph.cc.

10.208.3.23 bool coco::coco::model_gid::its_me (const model & *__m*) const [inline]

The its_me method tests if the model ___m is the reference model for this model group.

Definition at line 1077 of file search_graph.cc.

10.208.3.24 void coco::model_gid::make_const_back_ref (unsigned int *node,* unsigned int *cnum*) [inline]

This method constructs a constraint back reference from node number node to constraint number cnum.

Definition at line 676 of file model.hpp.

10.208.3.25 void coco::model_gid::make_obj_back_ref (unsigned int *node,* unsigned int *onum*) [inline]

This method constructs a objective back reference from node number node to objective number cnum.

Definition at line 634 of file model.hpp.

10.208.3.26 void coco::model_gid::mk_gconstref (unsigned int *n,* const model::walker & *__w*) [inline]

This method constructs a global constraint reference to ___w with number n.

Definition at line 583 of file model.hpp.

10.208.3.27 void coco::model_gid::mk_globref (unsigned int *n,* const model::walker & *__w*) [inline]

This method constructs a global node reference to ___w with number n.

Definition at line 539 of file model.hpp.

10.208.3.28 void coco::model_gid::mk_gobjref (unsigned int *n*, const model::walker & *__w*) [inline]

This method constructs a global objective reference to ___w with number n.

Definition at line 565 of file model.hpp.

10.208.3.29 void coco::model_gid::mk_gvarref (unsigned int *n*, const model::walker & *__w*) [inline]

This method constructs a global variable reference to ___w with number n.

Definition at line 552 of file model.hpp.

10.208.3.30 const std::string coco::coco::model_gid::model_name ( ) const [inline]

This method returns the name of the model.

Definition at line 1106 of file search_graph.cc.

10.208.3.31 void coco::coco::model_gid::model_name (const std::string & *n*) [inline]

This method sets the name of the model to n.

Definition at line 1108 of file search_graph.cc.

10.208.3.32 void coco::coco::model_gid::model_name (const char * *n*) [inline]

This method sets the name of the model to n.

Definition at line 1110 of file search_graph.cc.

10.208.3.33 size_t coco::coco::model_gid::n_fixed_vars ( ) const [inline]

This method returns the number of variables fixed by the presolve.

Definition at line 1164 of file search_graph.cc.

10.208.3.34 size_t coco::coco::model_gid::n_unused_constrs ( ) const [inline]

This method returns the number of unused constraints.

Definition at line 1195 of file search_graph.cc.

10.208.3.35 size_t coco::coco::model_gid::n_unused_objs ( ) const [inline]

This method returns the number of unused objectives.

Definition at line 1185 of file search_graph.cc.

10.208.3.36 size_t coco::coco::model_gid::n_unused_vars ( ) const [inline]

This method returns the number of unused variables.

Definition at line 1175 of file search_graph.cc.

10.208.3.37 const model::walker& coco::coco::model_gid::node (unsigned int *i*) const [inline]

A call to this method returns a walker to node `i`.

Definition at line 1061 of file search_graph.cc.

10.208.3.38 unsigned int coco::coco::model_gid::number_of_constraints ( ) const [inline]

The number_of_constraints method returns the number of constraints across all models governed by the same model_iddata.

Definition at line 989 of file search_graph.cc.

10.208.3.39 void coco::coco::model_gid::number_of_constraints (unsigned int *_n*) [inline]

This method sets the number of constraints across all model groups in the same model_iddata.

Definition at line 1002 of file search_graph.cc.

10.208.3.40 unsigned int coco::coco::model_gid::number_of_nodes ( ) const [inline]

The number_of_nodes method returns the number of nodes across all models governed by the same model_iddata.

Definition at line 978 of file search_graph.cc.

10.208.3.41 void coco::coco::model_gid::number_of_nodes (unsigned int *_n*) [inline]

This method sets the number of nodes across all model groups in the same model_iddata.

Definition at line 993 of file search_graph.cc.

10.208.3.42 unsigned int coco::coco::model_gid::number_of_objectives ( ) const [inline]

The number_of_objectives method returns the number of objectives across all models governed by the same model_iddata.

Definition at line 985 of file search_graph.cc.

10.208.3.43 void coco::coco::model_gid::number_of_objectives (unsigned int *_n*) [inline]

This method sets the number of objectives across all model groups in the same model_iddata.

Definition at line 999 of file search_graph.cc.

10.208.3.44 unsigned int coco::coco::model_gid::number_of_variables ( ) const [inline]

The number_of_variables method returns the number of variables across all models governed by the same model_iddata.

Definition at line 981 of file search_graph.cc.

10.208.3.45 void coco::coco::model_gid::number_of_variables (unsigned int *_n*) [inline]

This method sets the number of variables across all model groups in the same model_iddata.

Definition at line 996 of file search_graph.cc.

10.208.3.46 double coco::coco::model_gid::obj_adj (unsigned int *n*) const [inline]

This method returns the additive constant in the objective function n.

Definition at line 1152 of file search_graph.cc.

10.208.3.47 void coco::coco::model_gid::obj_adj (unsigned int *n*, double *adj*) [inline]

With this method the additive constant in the objective function n is set to adj.

Definition at line 1155 of file search_graph.cc.

10.208.3.48 double coco::coco::model_gid::obj_mult (unsigned int *n*) const [inline]

This method returns the scalar multiplier of the objective function n.

Definition at line 1158 of file search_graph.cc.

10.208.3.49 void coco::coco::model_gid::obj_mult (unsigned int *n*, double *mult*) [inline]

With this method the scalar multiplier of the objective function n is set to mult.

Definition at line 1161 of file search_graph.cc.

10.208.3.50 const std::string coco::coco::model_gid::obj_name (unsigned int *n*) const [inline]

This method returns the name of the objective n.

Definition at line 1121 of file search_graph.cc.

10.208.3.51 void coco::coco::model_gid::obj_name (unsigned int *n*, const std::string & *vn*) [inline]

A call to this method sets the name of the objective n to cn.

Definition at line 1124 of file search_graph.cc.

10.208.3.52 void coco::coco::model_gid::obj_name (unsigned int *n*, const char * *vn*) [inline]

A call to this method sets the name of the objective n to cn.

Definition at line 1127 of file search_graph.cc.

10.208.3.53 const model::walker& coco::coco::model_gid::objective (unsigned int *i*) const [inline]

A call to this method returns a walker to objective i.

Definition at line 1065 of file search_graph.cc.

10.208.3.54 const std::map< int, std::vector< double > > & coco::model_gid::optima ( ) const [inline]

This method returns the known optima defined. The return value is a map of known optima numbers to coordinate vectors.

Definition at line 702 of file model.hpp.

10.208.3.55 void coco::coco::model_gid::remove_const_id (unsigned int *n*) [inline]

This method inserts the constraint number n in the list of free constraint numbers.

Definition at line 1047 of file search_graph.cc.

10.208.3.56 void coco::model_gid::remove_const_ref (unsigned int *_n*) [inline]

This method removes the global constraint reference of node _n.

Definition at line 643 of file model.hpp.

10.208.3.57 void coco::coco::model_gid::remove_node_id (unsigned int *n*) [inline]

This method inserts the node number n in the list of free node numbers.

Definition at line 1029 of file search_graph.cc.

10.208.3.58 void coco::coco::model_gid::remove_node_ref (unsigned int *_n*) [inline]

This method removes the global node reference of node _n.

Definition at line 940 of file search_graph.cc.

10.208.3.59 void coco::coco::model_gid::remove_obj_id (unsigned int *n*) [inline]

This method inserts the objective number n in the list of free objective numbers.

Definition at line 1041 of file search_graph.cc.

10.208.3.60 void coco::model_gid::remove_obj_ref (unsigned int *_n*) [inline]

This method removes the global objective reference of node _n.

Definition at line 601 of file model.hpp.

10.208.3.61 void coco::coco::model_gid::remove_var_id (unsigned int *n*) [inline]

This method inserts the variable number n in the list of free variable numbers.

Definition at line 1035 of file search_graph.cc.

10.208.3.62 void coco::coco::model_gid::remove_var_ref (unsigned int *_n*) [inline]

This method removes the global variable reference of node _n.

Definition at line 944 of file search_graph.cc.

10.208.3.63 void coco::coco::model_gid::set_glob_ref (const std::vector< model::walker > & *node_ref*) [inline]

This method initializes the glob_ref array and the maximal node number.

Definition at line 686 of file model.hpp.

10.208.3.64 void coco::coco::model_gid::set_gvar_ref (const std::vector< model::walker > & *var_ref*) [inline]

This method initializes the glob_ref array and the maximal node number.

Definition at line 693 of file model.hpp.

10.208.3.65 const std::map< int, std::vector< double > > & coco::model_gid::trace_points ( ) const [inline]

This method returns the trace points defined. The return value is a map of trace point numbers to coordinate vectors.

Definition at line 699 of file model.hpp.

10.208.3.66 const std::string& coco::coco::model_gid::unused_constr (unsigned int *n*) const [inline]

This method returns the name of the unused constraint n.

Definition at line 1197 of file search_graph.cc.

10.208.3.67 void coco::coco::model_gid::unused_constr (const std::string & *vn*) [inline]

This method stores the unused constraint with name vn.

Definition at line 1200 of file search_graph.cc.

10.208.3.68 void coco::coco::model_gid::unused_constr (const char * *vn*) [inline]

This method stores the unused constraint with name vn.

Definition at line 1202 of file search_graph.cc.

10.208.3.69 const std::string& coco::coco::model_gid::unused_obj (unsigned int *n*) const [inline]

This method returns the name of the unused objective n.

Definition at line 1187 of file search_graph.cc.

10.208.3.70 void coco::coco::model_gid::unused_obj (const std::string & *vn*) [inline]

This method stores the unused objective with name vn.

Definition at line 1190 of file search_graph.cc.

10.208.3.71 void coco::coco::model_gid::unused_obj (const char * *vn*) [inline]

This method stores the unused objective with name vn.

Definition at line 1192 of file search_graph.cc.

10.208.3.72 const std::string& coco::coco::model_gid::unused_var (unsigned int *n*) const [inline]

This method returns the name of the unused variable n.

Definition at line 1177 of file search_graph.cc.

10.208.3.73 void coco::coco::model_gid::unused_var (const std::string & *vn*) [inline]

This method stores the unused variable with name vn.

Definition at line 1180 of file search_graph.cc.

10.208.3.74 void coco::coco::model_gid::unused_var (const char * *vn*) [inline]

This method stores the unused variable with name vn.

Definition at line 1182 of file search_graph.cc.

10.208.3.75 const std::string coco::coco::model_gid::var_name (unsigned int *n*) const [inline]

This method returns the name of the variable n.

Definition at line 1113 of file search_graph.cc.

10.208.3.76 void coco::coco::model_gid::var_name (unsigned int *n*, const std::string & *vn*) [inline]

A call to this method sets the name of the variable n to vn.

Definition at line 1115 of file search_graph.cc.

10.208.3.77 void coco::coco::model_gid::var_name (unsigned int *n*, const char * *vn*) [inline]

A call to this method sets the name of the variable n to vn.

Definition at line 1117 of file search_graph.cc.

10.208.3.78 const model::walker& coco::coco::model_gid::variable (unsigned int *i*) const [inline]

A call to this method returns a walker to variable i.

Definition at line 1063 of file search_graph.cc.

10.208.4 Friends And Related Function Documentation

10.208.4.1 friend class model_iddata [friend]

Definition at line 1220 of file search_graph.cc.

The documentation for this class was generated from the following files:

- model.h
- model.cc
- model.hpp

10.209 coco::coco::model_iddata Class Reference

The model id-data class (the topmost in the model class hierarchy)

Public Member Functions

- model_iddata (unsigned int n=0)
- ~model_iddata ()
- void new_ref (model_gid &__m)
- bool delete_ref (model_gid &__m)
- void compress_numbers (bool renum_vars, bool renum_objs=false, bool renum_consts=false)

Methods for handling the global model data

These methods can be used for handling important information of a model about nodes, variables, constraints,...

- unsigned int number_of_nodes () const
- unsigned int number_of_variables () const
- unsigned int number_of_objectives () const
- unsigned int number_of_constraints () const
- void number_of_nodes (unsigned int _n)
- void number_of_variables (unsigned int _n)
- void number_of_objectives (unsigned int _n)
- void number_of_constraints (unsigned int _n)
- unsigned int get_node_id ()
- void remove_node_id (unsigned int n)
- unsigned int get_var_id ()
- void remove_var_id (unsigned int n)
- unsigned int get_obj_id ()
- void remove_obj_id (unsigned int n)
- unsigned int get_const_id ()
- void remove_const_id (unsigned int n)

Methods for handling modeling information

These methods return names of variables and constants and other data which is irrelevant for the solution process but necessary for user interaction.

- const std::string model_name () const
- void model_name (const std::string &n)
- const std::string var_name (unsigned int n) const
- void var_name (unsigned int n, const std::string &vn)
- const std::string const_name (unsigned int n) const
- void const_name (unsigned int n, const std::string &cn)
- const std::string obj_name (unsigned int n) const
- void obj_name (unsigned int n, const std::string &vn)
- double obj_adj (unsigned int n) const
- void obj_adj (unsigned int n, double adj)
- double obj_mult (unsigned int n) const
- void obj_mult (unsigned int n, double mult)
- size_t n_fixed_vars () const
- std::pair< const std::string, double > fixed_var (unsigned int n) const
- void fixed_var (const std::string &vn, double val)
- size_t n_unused_vars () const
- const std::string & unused_var (unsigned int n) const
- void unused_var (const std::string &vn)
- size_t n_unused_objs () const
- const std::string & unused_obj (unsigned int n) const
- void unused_obj (const std::string &vn)
- size_t n_unused_constrs () const
- const std::string & unused_constr (unsigned int n) const
- void unused_constr (const std::string &vn)

Friends

- class model_gid

10.209.1 Detailed Description

Âă The model_iddata class is the topmost in a hierarchy of model classes. It represents the important global information, manages the free and used numbers for nodes, variables, and constraints. There is usually no need to directly access members or methods of this class. The class is managed automatically by the various methods of the model_gid and model classes. All model groups (model_gid) in one model_iddata have the same variable, node, and constraint numbering, and share the same global information (free and used numbers, names, ...).

10.209.2 Constructor & Destructor Documentation

10.209.2.1 coco::coco::model_iddata::model_iddata (unsigned int *n* = `0`) [inline]

Standard Constructor, which prepares for `n` variables.

Definition at line 690 of file search_graph.cc.

10.209.2.2 coco::coco::model_iddata::~model_iddata ( ) [inline]

Standard Destructor

Definition at line 703 of file search_graph.cc.

10.209.3 Member Function Documentation

10.209.3.1 void coco::model_iddata::compress_numbers (bool *renum_vars*, bool *renum_objs* = `false`, bool *renum_consts* = `false`)

If this method is called, the nodes, variables, and constraints are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed.

Definition at line 81 of file model.cc.

10.209.3.2 const std::string coco::model_iddata::const_name (unsigned int *n*) const [inline]

This method returns the name of the constraint `n`.

Definition at line 304 of file model.hpp.

10.209.3.3 void coco::model_iddata::const_name (unsigned int *n*, const std::string & *cn*) [inline]

A call to this method sets the name of the constraint `n` to `cn`.

Definition at line 316 of file model.hpp.

10.209.3.4 bool coco::model_iddata::delete_ref (model_gid & *__m*) [inline]

This method deletes the back reference to the model_gid ___m.

Definition at line 52 of file model.hpp.

10.209.3.5 std::pair< const std::string, double > coco::model_iddata::fixed_var (unsigned int *n*) const [inline]

This method returns the name and value of the fixed variable n.

Definition at line 337 of file model.hpp.

10.209.3.6 void coco::model_iddata::fixed_var (const std::string & *vn*, double *val*) [inline]

This method stores the fixed variable with name vn and value val.

Definition at line 347 of file model.hpp.

10.209.3.7 unsigned int coco::model_iddata::get_const_id ( ) [inline]

This method returns a free constraint number a removes it from the list of free constraint numbers.

Definition at line 170 of file model.hpp.

10.209.3.8 unsigned int coco::model_iddata::get_node_id ( ) [inline]

This method returns a free node number a removes it from the list of free node numbers.

Definition at line 66 of file model.hpp.

10.209.3.9 unsigned int coco::model_iddata::get_obj_id ( ) [inline]

This method returns a free objective number a removes it from the list of free objective numbers.

Definition at line 134 of file model.hpp.

10.209.3.10 unsigned int coco::model_iddata::get_var_id ( ) [inline]

This method returns a free variable number a removes it from the list of free variable numbers.

Definition at line 99 of file model.hpp.

10.209.3.11 const std::string coco::model_iddata::model_name ( ) const [inline]

This method returns the name of the model.

Definition at line 243 of file model.hpp.

10.209.3.12 void coco::model_iddata::model_name (const std::string & *n*) [inline]

This method sets the name of the model to n.

Definition at line 251 of file model.hpp.

10.209.3.13 size_t coco::coco::model_iddata::n_fixed_vars ( ) const [inline]

This method returns the number of variables fixed by the preprocessor.

Definition at line 813 of file search_graph.cc.

10.209.3.14 size_t coco::coco::model_iddata::n_unused_constrs ( ) const [inline]

This method returns the number of unused constraints.

Definition at line 834 of file search_graph.cc.

10.209.3.15 size_t coco::coco::model_iddata::n_unused_objs ( ) const [inline]

This method returns the number of unused objectives.

Definition at line 827 of file search_graph.cc.

10.209.3.16 size_t coco::coco::model_iddata::n_unused_vars ( ) const [inline]

This method returns the number of unused variables.

Definition at line 820 of file search_graph.cc.

10.209.3.17 void coco::coco::model_iddata::new_ref (model_gid & __m) [inline]

This method adds a new back reference to the model_gid ___m.

Definition at line 706 of file search_graph.cc.

10.209.3.18 unsigned int coco::coco::model_iddata::number_of_constraints ( ) const [inline]

A call to this method retrieves the start of the contiguous part of the free constraints. This is the allocated size of the constraint access vectors.

Definition at line 730 of file search_graph.cc.

10.209.3.19 void coco::model_iddata::number_of_constraints (unsigned int _n) [inline]

This method sets the number of constraints to _n.

Definition at line 234 of file model.hpp.

10.209.3.20 unsigned int coco::coco::model_iddata::number_of_nodes ( ) const [inline]

A call to this method retrieves the start of the contiguous part of the free nodes. This is the allocated size of the node access vectors.

Definition at line 718 of file search_graph.cc.

10.209.3.21 void coco::model_iddata::number_of_nodes (unsigned int _n) [inline]

This method sets the number of nodes to _n.

Definition at line 206 of file model.hpp.

10.209.3.22 unsigned int coco::coco::model_iddata::number_of_objectives ( ) const [inline]

A call to this method retrieves the start of the contiguous part of the free objectives. This is the allocated size of the objective access vectors.

Definition at line 726 of file search_graph.cc.

10.209.3.23 void coco::model_iddata::number_of_objectives (unsigned int _n) [inline]

This method sets the number of objectives to _n.

Definition at line 223 of file model.hpp.

10.209.3.24 unsigned int coco::coco::model_iddata::number_of_variables ( ) const [inline]

A call to this method retrieves the start of the contiguous part of the free variables. This is the allocated size of the variable access vectors.

Definition at line 722 of file search_graph.cc.

10.209.3.25 void coco::model_iddata::number_of_variables (unsigned int _n) [inline]

This method sets the number of variables to _n.

Definition at line 214 of file model.hpp.

10.209.3.26 double coco::model_iddata::obj_adj (unsigned int n) const [inline]

This method returns the additive constant in the objective function n.

Definition at line 328 of file model.hpp.

10.209.3.27 void coco::model_iddata::obj_adj (unsigned int n, double adj) [inline]

With this method the additive constant in the objective function n is set to adj.

Definition at line 330 of file model.hpp.

10.209.3.28 double coco::model_iddata::obj_mult (unsigned int n) const [inline]

This method returns the scalar multiplier of the objective function n.

Definition at line 332 of file model.hpp.

10.209.3.29 void coco::model_iddata::obj_mult (unsigned int n, double mult) [inline]

With this method the scalar multiplier of the objective function n is set to mult.

Definition at line 334 of file model.hpp.

10.209.3.30 const std::string coco::model_iddata::obj_name (unsigned int n) const [inline]

This method returns the name of the objective function.

Definition at line 280 of file model.hpp.

10.209.3.31 void coco::model_iddata::obj_name (unsigned int *n*, const std::string & *vn*) [inline]

A call to this method sets the name of the objective function n to vn.

Definition at line 292 of file model.hpp.

10.209.3.32 void coco::model_iddata::remove_const_id (unsigned int *n*) [inline]

This method inserts the constraint number n in the list of free constraint numbers.

Definition at line 183 of file model.hpp.

10.209.3.33 void coco::model_iddata::remove_node_id (unsigned int *n*) [inline]

This method inserts the node number n in the list of free node numbers.

Definition at line 79 of file model.hpp.

10.209.3.34 void coco::model_iddata::remove_obj_id (unsigned int *n*) [inline]

This method inserts the objective number n in the list of free objective numbers.

Definition at line 147 of file model.hpp.

10.209.3.35 void coco::model_iddata::remove_var_id (unsigned int *n*) [inline]

This method inserts the variable number n in the list of free variable numbers.

Definition at line 112 of file model.hpp.

10.209.3.36 const std::string & coco::model_iddata::unused_constr (unsigned int *n*) const [inline]

This method returns the name of the unused constraint n.

Definition at line 380 of file model.hpp.

10.209.3.37 void coco::model_iddata::unused_constr (const std::string & *vn*) [inline]

This method stores the unused constraint with name vn.

Definition at line 389 of file model.hpp.

10.209.3.38 const std::string & coco::model_iddata::unused_obj (unsigned int *n*) const [inline]

This method returns the name of the unused objective n.

Definition at line 366 of file model.hpp.

10.209.3.39 void coco::model_iddata::unused_obj (const std::string & *vn*) [inline]

This method stores the unused objective with name vn.

Definition at line 375 of file model.hpp.

10.209.3.40 const std::string & coco::model_iddata::unused_var (unsigned int *n*) const [inline]

This method returns the name of the unused variable n.

Definition at line 352 of file model.hpp.

10.209.3.41 void coco::model_iddata::unused_var (const std::string & *vn*) [inline]

This method stores the unused variable with name vn.

Definition at line 361 of file model.hpp.

10.209.3.42 const std::string coco::model_iddata::var_name (unsigned int *n*) const [inline]

This method returns the name of the variable n.

Definition at line 256 of file model.hpp.

10.209.3.43 void coco::model_iddata::var_name (unsigned int *n*, const std::string & *vn*) [inline]

A call to this method sets the name of the variable n to vn.

Definition at line 268 of file model.hpp.

10.209.4 Friends And Related Function Documentation

10.209.4.1 friend class model_gid [friend]

Definition at line 841 of file search_graph.cc.

The documentation for this class was generated from the following files:

- model.h
- model.cc
- model.hpp

10.210 coco::model_iddata Class Reference

The model id-data class (the topmost in the model class hierarchy)

```
#include <model.h>
```

Public Member Functions

- model_iddata (unsigned int n=0)
- ~model_iddata ()
- void new_ref (model_gid &__m)
- bool delete_ref (model_gid &__m)
- void compress_numbers (bool renum_vars, bool renum_objs=false, bool renum_consts=false)

Methods for handling the global model data

These methods can be used for handling important information of a model about nodes, variables, constraints,...

- unsigned int number_of_nodes () const
- unsigned int number_of_variables () const
- unsigned int number_of_objectives () const
- unsigned int number_of_constraints () const
- void number_of_nodes (unsigned int _n)
- void number_of_variables (unsigned int _n)
- void number_of_objectives (unsigned int _n)
- void number_of_constraints (unsigned int _n)
- unsigned int get_node_id ()
- void remove_node_id (unsigned int n)
- unsigned int get_var_id ()
- void remove_var_id (unsigned int n)
- unsigned int get_obj_id ()
- void remove_obj_id (unsigned int n)
- unsigned int get_const_id ()
- void remove_const_id (unsigned int n)

Methods for handling modeling information

These methods return names of variables and constants and other data which is irrelevant for the solution process but necessary for user interaction.

- const std::string model_name () const
- void model_name (const std::string &n)
- const std::string var_name (unsigned int n) const
- void var_name (unsigned int n, const std::string &vn)
- const std::string const_name (unsigned int n) const
- void const_name (unsigned int n, const std::string &cn)
- const std::string obj_name (unsigned int n) const
- void obj_name (unsigned int n, const std::string &vn)
- double obj_adj (unsigned int n) const
- void obj_adj (unsigned int n, double adj)
- double obj_mult (unsigned int n) const
- void obj_mult (unsigned int n, double mult)
- size_t n_fixed_vars () const
- std::pair< const std::string, double > fixed_var (unsigned int n) const
- void fixed_var (const std::string &vn, double val)
- size_t n_unused_vars () const
- const std::string & unused_var (unsigned int n) const
- void unused_var (const std::string &vn)
- size_t n_unused_objs () const
- const std::string & unused_obj (unsigned int n) const
- void unused_obj (const std::string &vn)
- size_t n_unused_constrs () const
- const std::string & unused_constr (unsigned int n) const
- void unused_constr (const std::string &vn)

Friends

- class model_gid

10.210.1 Detailed Description

Âă The model_iddata class is the topmost in a hierarchy of model classes. It represents the important global information, manages the free and used numbers for nodes, variables, and constraints. There is usually no need to directly access members or methods of this class. The class is managed automatically by the various methods of the model_gid and model classes. All model groups (model_gid) in one model_iddata have the same variable, node, and constraint numbering, and share the same global information (free and used numbers, names, ...).

10.210.2 Constructor & Destructor Documentation

10.210.2.1 coco::model_iddata::model_iddata (unsigned int *n* = `0`) `[inline]`

Standard Constructor, which prepares for `n` variables.

Definition at line 690 of file model.h.

10.210.2.2 coco::model_iddata::~model_iddata ( ) `[inline]`

Standard Destructor

Definition at line 703 of file model.h.

10.210.3 Member Function Documentation

10.210.3.1 void coco::model_iddata::compress_numbers (bool *renum_vars,* bool *renum_objs* = `false`, bool *renum_consts* = `false`)

If this method is called, the nodes, variables, and constraints are renumbered to fill gaps (unused numbers). To keep the model hierarchy consistent, all numbers in all models in the same model_iddata are changed.

10.210.3.2 const std::string coco::model_iddata::const_name (unsigned int *n*) const

This method returns the name of the constraint `n`.

10.210.3.3 void coco::model_iddata::const_name (unsigned int *n,* const std::string & *cn*)

A call to this method sets the name of the constraint `n` to `cn`.

10.210.3.4 bool coco::model_iddata::delete_ref (model_gid & *__m*)

This method deletes the back reference to the model_gid `__m`.

10.210.3.5 std::pair<const std::string, double> coco::model_iddata::fixed_var (unsigned int *n*) const

This method returns the name and value of the fixed variable `n`.

10.210.3.6 void coco::model_iddata::fixed_var (const std::string & *vn,* double *val*)

This method stores the fixed variable with name `vn` and value `val`.

10.210.3.7 unsigned int coco::model_iddata::get_const_id ( )

This method returns a free constraint number a removes it from the list of free constraint numbers.

10.210.3.8 unsigned int coco::model_iddata::get_node_id ( )

This method returns a free node number a removes it from the list of free node numbers.

10.210.3.9 unsigned int coco::model_iddata::get_obj_id ( )

This method returns a free objective number a removes it from the list of free objective numbers.

10.210.3.10 unsigned int coco::model_iddata::get_var_id ( )

This method returns a free variable number a removes it from the list of free variable numbers.

10.210.3.11 const std::string coco::model_iddata::model_name ( ) const

This method returns the name of the model.

10.210.3.12 void coco::model_iddata::model_name (const std::string & *n*)

This method sets the name of the model to `n`.

10.210.3.13 size_t coco::model_iddata::n_fixed_vars ( ) const [inline]

This method returns the number of variables fixed by the preprocessor.

Definition at line 813 of file model.h.

10.210.3.14 size_t coco::model_iddata::n_unused_constrs ( ) const [inline]

This method returns the number of unused constraints.

Definition at line 834 of file model.h.

10.210.3.15 size_t coco::model_iddata::n_unused_objs ( ) const [inline]

This method returns the number of unused objectives.

Definition at line 827 of file model.h.

10.210.3.16 size_t coco::model_iddata::n_unused_vars ( ) const [inline]

This method returns the number of unused variables.

Definition at line 820 of file model.h.

10.210.3.17 void coco::model_iddata::new_ref (model_gid & *__m*) [inline]

This method adds a new back reference to the model_gid `__m`.

Definition at line 706 of file model.h.

10.210.3.18 unsigned int coco::model_iddata::number_of_constraints ( ) const [inline]

A call to this method retrieves the start of the contiguous part of the free constraints. This is the allocated size of the constraint access vectors.

Definition at line 730 of file model.h.

10.210.3.19 void coco::model_iddata::number_of_constraints (unsigned int _n)

This method sets the number of constraints to _n.

10.210.3.20 unsigned int coco::model_iddata::number_of_nodes ( ) const [inline]

A call to this method retrieves the start of the contiguous part of the free nodes. This is the allocated size of the node access vectors.

Definition at line 718 of file model.h.

10.210.3.21 void coco::model_iddata::number_of_nodes (unsigned int _n)

This method sets the number of nodes to _n.

10.210.3.22 unsigned int coco::model_iddata::number_of_objectives ( ) const [inline]

A call to this method retrieves the start of the contiguous part of the free objectives. This is the allocated size of the objective access vectors.

Definition at line 726 of file model.h.

10.210.3.23 void coco::model_iddata::number_of_objectives (unsigned int _n)

This method sets the number of objectives to _n.

10.210.3.24 unsigned int coco::model_iddata::number_of_variables ( ) const [inline]

A call to this method retrieves the start of the contiguous part of the free variables. This is the allocated size of the variable access vectors.

Definition at line 722 of file model.h.

10.210.3.25 void coco::model_iddata::number_of_variables (unsigned int _n)

This method sets the number of variables to _n.

10.210.3.26 double coco::model_iddata::obj_adj (unsigned int n) const

This method returns the additive constant in the objective function n.

10.210.3.27 void coco::model_iddata::obj_adj (unsigned int n, double adj)

With this method the additive constant in the objective function n is set to adj.

10.210.3.28 double coco::model_iddata::obj_mult (unsigned int n) const

This method returns the scalar multiplier of the objective function n.

10.210.3.29 void coco::model_iddata::obj_mult (unsigned int *n*, double *mult*)

With this method the scalar multiplier of the objective function `n` is set to `mult`.

10.210.3.30 const std::string coco::model_iddata::obj_name (unsigned int *n*) const

This method returns the name of the objective function.

10.210.3.31 void coco::model_iddata::obj_name (unsigned int *n*, const std::string & *vn*)

A call to this method sets the name of the objective function `n` to `vn`.

10.210.3.32 void coco::model_iddata::remove_const_id (unsigned int *n*)

This method inserts the constraint number `n` in the list of free constraint numbers.

10.210.3.33 void coco::model_iddata::remove_node_id (unsigned int *n*)

This method inserts the node number `n` in the list of free node numbers.

10.210.3.34 void coco::model_iddata::remove_obj_id (unsigned int *n*)

This method inserts the objective number `n` in the list of free objective numbers.

10.210.3.35 void coco::model_iddata::remove_var_id (unsigned int *n*)

This method inserts the variable number `n` in the list of free variable numbers.

10.210.3.36 const std::string& coco::model_iddata::unused_constr (unsigned int *n*) const

This method returns the name of the unused constraint `n`.

10.210.3.37 void coco::model_iddata::unused_constr (const std::string & *vn*)

This method stores the unused constraint with name `vn`.

10.210.3.38 const std::string& coco::model_iddata::unused_obj (unsigned int *n*) const

This method returns the name of the unused objective `n`.

10.210.3.39 void coco::model_iddata::unused_obj (const std::string & *vn*)

This method stores the unused objective with name `vn`.

10.210.3.40 const std::string& coco::model_iddata::unused_var (unsigned int *n*) const

This method returns the name of the unused variable `n`.

10.210.3.41 void coco::model_iddata::unused_var (const std::string & *vn*)

This method stores the unused variable with name `vn`.

10.210.3.42 const std::string coco::model_iddata::var_name (unsigned int *n*) const

This method returns the name of the variable n.

10.210.3.43 void coco::model_iddata::var_name (unsigned int *n*, const std::string & *vn*)

A call to this method sets the name of the variable n to vn.

10.210.4 Friends And Related Function Documentation

10.210.4.1 friend class model_gid [friend]

Definition at line 841 of file model.h.

The documentation for this class was generated from the following file:

- model.h

10.211 coco::coco::coco::my_rounded_math< T > Struct Template Reference

10.211.1 Detailed Description

template<class T>struct coco::coco::coco::my_rounded_math< T >

Definition at line 91 of file search_graph.cc.

The documentation for this struct was generated from the following file:

- interval_boost.h

10.212 coco::coco::my_rounded_math< T > Struct Template Reference

```
#include <expression.h>
```

10.212.1 Detailed Description

template<class T>struct coco::coco::my_rounded_math< T >

Definition at line 91 of file expression.h.

The documentation for this struct was generated from the following file:

- interval_boost.h

10.213 coco::my_rounded_math Struct Reference

`#include <interval_boost.h>`

The documentation for this struct was generated from the following file:

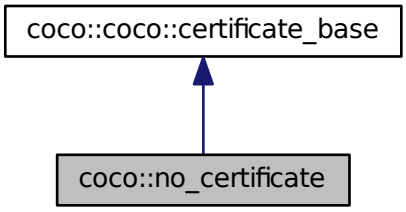
- interval_boost.h

10.214 coco::no_certificate Class Reference

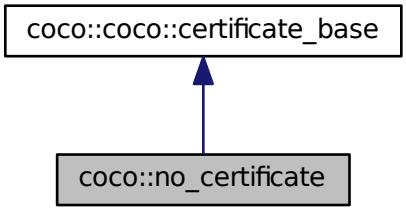
The not-certified certificate.

`#include <api_cert.h>`

Inheritance diagram for coco::no_certificate:



Collaboration diagram for coco::no_certificate:



Public Member Functions

- no_certificate ()
- no_certificate (const no_certificate &__c)
- virtual ~no_certificate ()
- no_certificate * new_copy () const
- void destroy_copy (certificate_base *__c) const
- bool verify (const work_node &_x) const
- bool operator== (const certificate_base &_c) const
- bool operator!= (const certificate_base &_c) const
- bool operator== (const no_certificate &_c) const
- bool operator!= (const no_certificate &_c) const
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- virtual std::string get_contents () const
- virtual std::string get_contents () const
- virtual std::string get_contents () const
- virtual std::string get_contents () const

Protected Attributes

- std::string _contents

10.214.1 Detailed Description

The empty certificate for inference engines, which do not produce certificates!

10.214.2 Constructor & Destructor Documentation

10.214.2.1 coco::no_certificate::no_certificate ( ) [inline]

Standard Constructor

Definition at line 88 of file api_cert.h.

10.214.2.2 coco::no_certificate::no_certificate (const no_certificate & __c) [inline]

Standard Copy Constructor

Definition at line 90 of file api_cert.h.

10.214.2.3 virtual coco::no_certificate::~no_certificate ( ) [inline, virtual]

Standard Destructor

Definition at line 98 of file api_cert.h.

10.214.3 Member Function Documentation

10.214.3.1 void coco::no_certificate::destroy_copy (certificate_base * __c) const [inline]

Clone Destructor

Definition at line 109 of file api_cert.h.

10.214.3.2 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.214.3.3 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.214.3.4 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.214.3.5 virtual std::string coco::coco::certificate_base::get_contents ( ) const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::rigorous_module_certificate, and coco::split_certificate.

Definition at line 165 of file search_graph.cc.

10.214.3.6 certificate coco::coco::certificate_base::make_certificate (const std::string & c) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.214.3.7 certificate coco::coco::certificate_base::make_certificate (const std::string & c) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.214.3.8 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents `c`.

Definition at line 156 of file search_graph.cc.

10.214.3.9 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents `c`.

Definition at line 156 of file search_graph.cc.

10.214.3.10 no_certificate* coco::no_certificate::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::certificate_base.

Definition at line 101 of file api_cert.h.

10.214.3.11 bool coco::no_certificate::operator!= (const certificate_base & *_c*) const [inline]

Definition at line 120 of file api_cert.h.

10.214.3.12 bool coco::no_certificate::operator!= (const no_certificate & *_c*) const [inline]

Definition at line 126 of file api_cert.h.

10.214.3.13 bool coco::no_certificate::operator== (const certificate_base & *_c*) const [inline]

Comparison operators

Definition at line 117 of file api_cert.h.

10.214.3.14 bool coco::no_certificate::operator== (const no_certificate & *_c*) const [inline]

Comparison operators

Definition at line 125 of file api_cert.h.

10.214.3.15 bool coco::no_certificate::verify (const work_node & *_x*) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. In this class: `false`.

Reimplemented from coco::coco::certificate_base.

Definition at line 114 of file api_cert.h.

10.214.4 Member Data Documentation

10.214.4.1 std::string coco::coco::certificate_base::_contents [protected, inherited]

The contents (descriptive string, type) of this certificate

Definition at line 128 of file search_graph.cc.

The documentation for this class was generated from the following file:

- api_cert.h

10.215 coco::num::Number Class Reference

`#include <expression.h>`

Public Member Functions

- Number ()
- Number (int n)
- Number (unsigned n)
- Number (double d)
- Number (cprat q)
- Number (const mpq_class &q)
- Number (const intv &i)
- Number (const str &s)
- Number (const Number &n)
- ~Number ()
- const Number & operator= (int n)
- const Number & operator= (unsigned n)
- const Number & operator= (double d)
- const Number & operator= (cprat q)
- const Number & operator= (const mpq_class &q)
- const Number & operator= (const intv &i)
- const Number & operator= (const str &s)
- const Number & operator= (const Number &n)
- cprat get_q () const
- mpq_class get_q_class () const
- str get_s () const
- intv get_i () const
- double get_r () const
- numtypes get_numtype ()
- const Number & approx ()
- str tostr () const
- str tooutputstr () const
- void fromstr (str s1)
- Number operator+ (const Number &n) const
- Number operator- (const Number &n) const
- Number operator* (const Number &n) const
- Number operator/ (const Number &n) const
- const Number & operator+= (const Number &n)

- const Number & operator-= (const Number &n)
- const Number & operator*= (const Number &n)
- const Number & operator/= (const Number &n)
- Number operator^ (const Number &n) const
- const Number & operator^= (const Number &n)
- bool seq (const Number &n) const
- bool sne (const Number &n) const
- bool sge (const Number &n) const
- bool sgt (const Number &n) const
- bool sle (const Number &n) const
- bool slt (const Number &n) const
- bool ceq (const Number &n) const
- bool cne (const Number &n) const
- bool cge (const Number &n) const
- bool cgt (const Number &n) const
- bool cle (const Number &n) const
- bool clt (const Number &n) const
- bool peq (const Number &n) const
- bool pne (const Number &n) const
- bool pge (const Number &n) const
- bool pgt (const Number &n) const
- bool ple (const Number &n) const
- bool plt (const Number &n) const
- bool operator== (const Number &n) const
- bool operator!= (const Number &n) const
- bool operator>= (const Number &n) const
- bool operator> (const Number &n) const
- bool operator<= (const Number &n) const
- bool operator< (const Number &n) const
- Number ()
- Number (int n)
- Number (unsigned n)
- Number (double d)
- Number (cprat q)
- Number (const mpq_class &q)
- Number (const intv &i)
- Number (const str &s)
- Number (const Number &n)
- ~Number ()
- const Number & operator= (int n)
- const Number & operator= (unsigned n)
- const Number & operator= (double d)
- const Number & operator= (cprat q)
- const Number & operator= (const mpq_class &q)
- const Number & operator= (const intv &i)
- const Number & operator= (const str &s)
- const Number & operator= (const Number &n)
- cprat get_q () const
- mpq_class get_q_class () const
- str get_s () const

- intv get_i () const
- double get_r () const
- numtypes get_numtype ()
- const Number & approx ()
- str tostr () const
- str tooutputstr () const
- void fromstr (str s1)
- Number operator+ (const Number &n) const
- Number operator- (const Number &n) const
- Number operator∗ (const Number &n) const
- Number operator/ (const Number &n) const
- const Number & operator+= (const Number &n)
- const Number & operator-= (const Number &n)
- const Number & operator∗= (const Number &n)
- const Number & operator/= (const Number &n)
- Number operator^ (const Number &n) const
- const Number & operator^= (const Number &n)
- bool seq (const Number &n) const
- bool sne (const Number &n) const
- bool sge (const Number &n) const
- bool sgt (const Number &n) const
- bool sle (const Number &n) const
- bool slt (const Number &n) const
- bool ceq (const Number &n) const
- bool cne (const Number &n) const
- bool cge (const Number &n) const
- bool cgt (const Number &n) const
- bool cle (const Number &n) const
- bool clt (const Number &n) const
- bool peq (const Number &n) const
- bool pne (const Number &n) const
- bool pge (const Number &n) const
- bool pgt (const Number &n) const
- bool ple (const Number &n) const
- bool plt (const Number &n) const
- bool operator== (const Number &n) const
- bool operator!= (const Number &n) const
- bool operator>= (const Number &n) const
- bool operator> (const Number &n) const
- bool operator<= (const Number &n) const
- bool operator< (const Number &n) const
- Number ()
- Number (int n)
- Number (unsigned n)
- Number (double d)
- Number (cprat q)
- Number (const mpq_class &q)
- Number (const intv &i)
- Number (const str &s)
- Number (const Number &n)

- ~Number ()
- const Number & operator= (int n)
- const Number & operator= (unsigned n)
- const Number & operator= (double d)
- const Number & operator= (cprat q)
- const Number & operator= (const mpq_class &q)
- const Number & operator= (const intv &i)
- const Number & operator= (const str &s)
- const Number & operator= (const Number &n)
- cprat get_q () const
- mpq_class get_q_class () const
- str get_s () const
- intv get_i () const
- double get_r () const
- numtypes get_numtype ()
- const Number & approx ()
- str tostr () const
- str tooutputstr () const
- void fromstr (str s1)
- Number operator+ (const Number &n) const
- Number operator- (const Number &n) const
- Number operator* (const Number &n) const
- Number operator/ (const Number &n) const
- const Number & operator+= (const Number &n)
- const Number & operator-= (const Number &n)
- const Number & operator*= (const Number &n)
- const Number & operator/= (const Number &n)
- Number operator^ (const Number &n) const
- const Number & operator^= (const Number &n)
- bool seq (const Number &n) const
- bool sne (const Number &n) const
- bool sge (const Number &n) const
- bool sgt (const Number &n) const
- bool sle (const Number &n) const
- bool slt (const Number &n) const
- bool ceq (const Number &n) const
- bool cne (const Number &n) const
- bool cge (const Number &n) const
- bool cgt (const Number &n) const
- bool cle (const Number &n) const
- bool clt (const Number &n) const
- bool peq (const Number &n) const
- bool pne (const Number &n) const
- bool pge (const Number &n) const
- bool pgt (const Number &n) const
- bool ple (const Number &n) const
- bool plt (const Number &n) const
- bool operator== (const Number &n) const
- bool operator!= (const Number &n) const
- bool operator>= (const Number &n) const

- bool operator> (const Number &n) const
- bool operator<= (const Number &n) const
- bool operator< (const Number &n) const
- Number ()
- Number (int n)
- Number (unsigned n)
- Number (double d)
- Number (cprat q)
- Number (const mpq_class &q)
- Number (const intv &i)
- Number (const str &s)
- Number (const Number &n)
- ~Number ()
- const Number & operator= (int n)
- const Number & operator= (unsigned n)
- const Number & operator= (double d)
- const Number & operator= (cprat q)
- const Number & operator= (const mpq_class &q)
- const Number & operator= (const intv &i)
- const Number & operator= (const str &s)
- const Number & operator= (const Number &n)
- cprat get_q () const
- mpq_class get_q_class () const
- str get_s () const
- intv get_i () const
- double get_r () const
- numtypes get_numtype ()
- const Number & approx ()
- str tostr () const
- str tooutputstr () const
- void fromstr (str s1)
- Number operator+ (const Number &n) const
- Number operator- (const Number &n) const
- Number operator* (const Number &n) const
- Number operator/ (const Number &n) const
- const Number & operator+= (const Number &n)
- const Number & operator-= (const Number &n)
- const Number & operator*= (const Number &n)
- const Number & operator/= (const Number &n)
- Number operator^ (const Number &n) const
- const Number & operator^= (const Number &n)
- bool seq (const Number &n) const
- bool sne (const Number &n) const
- bool sge (const Number &n) const
- bool sgt (const Number &n) const
- bool sle (const Number &n) const
- bool slt (const Number &n) const
- bool ceq (const Number &n) const
- bool cne (const Number &n) const
- bool cge (const Number &n) const

- bool cgt (const Number &n) const
- bool cle (const Number &n) const
- bool clt (const Number &n) const
- bool peq (const Number &n) const
- bool pne (const Number &n) const
- bool pge (const Number &n) const
- bool pgt (const Number &n) const
- bool ple (const Number &n) const
- bool plt (const Number &n) const
- bool operator== (const Number &n) const
- bool operator!= (const Number &n) const
- bool operator>= (const Number &n) const
- bool operator> (const Number &n) const
- bool operator<= (const Number &n) const
- bool operator< (const Number &n) const

Friends

- std::ostream & operator<< (std::ostream &os, const Number &n)
- std::istream & operator>> (std::istream &is, Number &n)
- Number acos (const Number &x)
- Number abs (const Number &x)
- Number acosh (const Number &x)
- Number acot (const Number &x)
- Number acoth (const Number &x)
- Number asin (const Number &x)
- Number asinh (const Number &x)
- Number atan (const Number &x)
- Number atanh (const Number &x)
- Number cos (const Number &x)
- Number cosh (const Number &x)
- Number cot (const Number &x)
- Number coth (const Number &x)
- Number exp (const Number &x)
- Number exp10 (const Number &x)
- Number exp2 (const Number &x)
- Number expm1 (const Number &x)
- Number log (const Number &x)
- Number log10 (const Number &x)
- Number log1p (const Number &x)
- Number log2 (const Number &x)
- Number power (const Number &x, int n)
- Number pow (const Number &x, const Number &y)
- Number sin (const Number &x)
- Number sinh (const Number &x)
- Number sqr (const Number &x)
- Number sqrt (const Number &x)
- Number tan (const Number &x)
- Number tanh (const Number &x)
- std::ostream & operator<< (std::ostream &os, const Number &n)

- std::istream & operator>> (std::istream &is, Number &n)
- Number acos (const Number &x)
- Number abs (const Number &x)
- Number acosh (const Number &x)
- Number acot (const Number &x)
- Number acoth (const Number &x)
- Number asin (const Number &x)
- Number asinh (const Number &x)
- Number atan (const Number &x)
- Number atanh (const Number &x)
- Number cos (const Number &x)
- Number cosh (const Number &x)
- Number cot (const Number &x)
- Number coth (const Number &x)
- Number exp (const Number &x)
- Number exp10 (const Number &x)
- Number exp2 (const Number &x)
- Number expm1 (const Number &x)
- Number log (const Number &x)
- Number log10 (const Number &x)
- Number log1p (const Number &x)
- Number log2 (const Number &x)
- Number power (const Number &x, int n)
- Number pow (const Number &x, const Number &y)
- Number sin (const Number &x)
- Number sinh (const Number &x)
- Number sqr (const Number &x)
- Number sqrt (const Number &x)
- Number tan (const Number &x)
- Number tanh (const Number &x)
- std::ostream & operator<< (std::ostream &os, const Number &n)
- std::istream & operator>> (std::istream &is, Number &n)
- Number acos (const Number &x)
- Number abs (const Number &x)
- Number acosh (const Number &x)
- Number acot (const Number &x)
- Number acoth (const Number &x)
- Number asin (const Number &x)
- Number asinh (const Number &x)
- Number atan (const Number &x)
- Number atanh (const Number &x)
- Number cos (const Number &x)
- Number cosh (const Number &x)
- Number cot (const Number &x)
- Number coth (const Number &x)
- Number exp (const Number &x)
- Number exp10 (const Number &x)
- Number exp2 (const Number &x)
- Number expm1 (const Number &x)
- Number log (const Number &x)

- Number log10 (const Number &x)
- Number log1p (const Number &x)
- Number log2 (const Number &x)
- Number power (const Number &x, int n)
- Number pow (const Number &x, const Number &y)
- Number sin (const Number &x)
- Number sinh (const Number &x)
- Number sqr (const Number &x)
- Number sqrt (const Number &x)
- Number tan (const Number &x)
- Number tanh (const Number &x)
- std::ostream & operator<< (std::ostream &os, const Number &n)
- std::istream & operator>> (std::istream &is, Number &n)
- Number acos (const Number &x)
- Number abs (const Number &x)
- Number acosh (const Number &x)
- Number acot (const Number &x)
- Number acoth (const Number &x)
- Number asin (const Number &x)
- Number asinh (const Number &x)
- Number atan (const Number &x)
- Number atanh (const Number &x)
- Number cos (const Number &x)
- Number cosh (const Number &x)
- Number cot (const Number &x)
- Number coth (const Number &x)
- Number exp (const Number &x)
- Number exp10 (const Number &x)
- Number exp2 (const Number &x)
- Number expm1 (const Number &x)
- Number log (const Number &x)
- Number log10 (const Number &x)
- Number log1p (const Number &x)
- Number log2 (const Number &x)
- Number power (const Number &x, int n)
- Number pow (const Number &x, const Number &y)
- Number sin (const Number &x)
- Number sinh (const Number &x)
- Number sqr (const Number &x)
- Number sqrt (const Number &x)
- Number tan (const Number &x)
- Number tanh (const Number &x)

10.215.1 Detailed Description

Definition at line 49 of file search_graph.cc.

10.215.2 Constructor & Destructor Documentation

10.215.2.1 coco::num::Number::Number ( )

10.215.2.2 coco::num::Number::Number (int *n*)

10.215.2.3 coco::num::Number::Number (unsigned *n*)

10.215.2.4 coco::num::Number::Number (double *d*)

10.215.2.5 coco::num::Number::Number (cprat *q*)

10.215.2.6 coco::num::Number::Number (const mpq_class & *q*)

10.215.2.7 coco::num::Number::Number (const intv & *i*)

10.215.2.8 coco::num::Number::Number (const str & *s*)

10.215.2.9 coco::num::Number::Number (const Number & *n*)

10.215.2.10 coco::num::Number::~Number ( )

10.215.2.11 coco::num::Number::Number ( )

10.215.2.12 coco::num::Number::Number (int *n*)

10.215.2.13 coco::num::Number::Number (unsigned *n*)

10.215.2.14 coco::num::Number::Number (double *d*)

10.215.2.15 coco::num::Number::Number (cprat *q*)

10.215.2.16 coco::num::Number::Number (const mpq_class & *q*)

10.215.2.17 coco::num::Number::Number (const intv & *i*)

10.215.2.18 coco::num::Number::Number (const str & *s*)

10.215.2.19 coco::num::Number::Number (const Number & *n*)

10.215.2.20 coco::num::Number::~Number ( )

10.215.2.21 coco::num::Number::Number ( )

10.215.2.22 coco::num::Number::Number (int *n*)

10.215.2.23 coco::num::Number::Number (unsigned *n*)

10.215.2.24 coco::num::Number::Number (double *d*)

10.215.2.25 coco::num::Number::Number (cprat *q*)

10.215.2.26 coco::num::Number::Number (const mpq_class & *q*)

10.215.2.27 coco::num::Number::Number (const intv & *i*)

10.215.2.28 coco::num::Number::Number (const str & *s*)

10.215.2.29 coco::num::Number::Number (const Number & *n*)

10.215.2.30 coco::num::Number::∼Number ( )

10.215.2.31 coco::num::Number::Number ( )

10.215.2.32 coco::num::Number::Number (int *n*)

10.215.2.33 coco::num::Number::Number (unsigned *n*)

10.215.2.34 coco::num::Number::Number (double *d*)

10.215.2.35 coco::num::Number::Number (cprat *q*)

10.215.2.36 coco::num::Number::Number (const mpq_class & *q*)

10.215.2.37 coco::num::Number::Number (const intv & *i*)

10.215.2.38 coco::num::Number::Number (const str & *s*)

10.215.2.39 coco::num::Number::Number (const Number & *n*)

10.215.2.40 coco::num::Number::∼Number ( )

10.215.3 Member Function Documentation

10.215.3.1 const Number& coco::num::Number::approx ( ) [inline]

Definition at line 108 of file expression.h.

10.215.3.2 const Number& coco::num::Number::approx ( ) [inline]

Definition at line 108 of file search_graph.cc.

10.215.3.3 const Number& coco::num::Number::approx ( ) [inline]

Definition at line 108 of file search_graph.cc.

10.215.3.4 const Number& coco::num::Number::approx ( ) [inline]

Definition at line 108 of file search_graph.cc.

10.215.3.5 bool coco::num::Number::ceq (const Number & *n*) const

10.215.3.6 bool coco::num::Number::ceq (const Number & *n*) const

10.215.3.7 bool coco::num::Number::ceq (const Number & *n*) const

10.215.3.8 bool coco::num::Number::ceq (const Number & *n*) const

10.215.3.9 bool coco::num::Number::cge (const Number & *n*) const

10.215.3.10 bool coco::num::Number::cge (const Number & *n*) const

10.215.3.11 bool coco::num::Number::cge (const Number & *n*) const

10.215.3.12 bool coco::num::Number::cge (const Number & *n*) const

10.215.3.13 bool coco::num::Number::cgt (const Number & *n*) const

10.215.3.14 bool coco::num::Number::cgt (const Number & *n*) const

10.215.3.15 bool coco::num::Number::cgt (const Number & *n*) const

10.215.3.16 bool coco::num::Number::cgt (const Number & *n*) const

10.215.3.17 bool coco::num::Number::cle (const Number & *n*) const

10.215.3.18 bool coco::num::Number::cle (const Number & *n*) const

10.215.3.19 bool coco::num::Number::cle (const Number & *n*) const

10.215.3.20 bool coco::num::Number::cle (const Number & *n*) const

10.215.3.21 bool coco::num::Number::clt (const Number & *n*) const

10.215.3.22 bool coco::num::Number::clt (const Number & *n*) const

10.215.3.23 bool coco::num::Number::clt (const Number & *n*) const

10.215.3.24 bool coco::num::Number::clt (const Number & *n*) const

10.215.3.25 bool coco::num::Number::cne (const Number & *n*) const

10.215.3.26 bool coco::num::Number::cne (const Number & *n*) const

10.215.3.27 bool coco::num::Number::cne (const Number & *n*) const

10.215.3.28 bool coco::num::Number::cne (const Number & *n*) const

10.215.3.29 void coco::num::Number::fromstr (str *s1*)

10.215.3.30 void coco::num::Number::fromstr (str *s1*)

10.215.3.31 void coco::num::Number::fromstr (str *s1*)

10.215.3.32 void coco::num::Number::fromstr (str *s1*)

10.215.3.33 intv coco::num::Number::get_i ( ) const [inline]

Definition at line 105 of file search_graph.cc.

10.215.3.34 intv coco::num::Number::get_i ( ) const [inline]

Definition at line 105 of file search_graph.cc.

10.215.3.35 intv coco::num::Number::get_i ( ) const [inline]

Definition at line 105 of file search_graph.cc.

10.215.3.36 intv coco::num::Number::get_i ( ) const [inline]

Definition at line 105 of file expression.h.

10.215.3.37 numtypes coco::num::Number::get_numtype ( ) [inline]

Definition at line 107 of file search_graph.cc.

10.215.3.38 numtypes coco::num::Number::get_numtype ( ) [inline]

Definition at line 107 of file search_graph.cc.

10.215.3.39 numtypes coco::num::Number::get_numtype ( ) [inline]

Definition at line 107 of file expression.h.

10.215.3.40 numtypes coco::num::Number::get_numtype ( ) [inline]

Definition at line 107 of file search_graph.cc.

10.215.3.41 cprat coco::num::Number::get_q ( ) const

10.215.3.42 cprat coco::num::Number::get_q ( ) const

10.215.3.43 cprat coco::num::Number::get_q ( ) const

10.215.3.44 cprat coco::num::Number::get_q ( ) const

10.215.3.45 mpq_class coco::num::Number::get_q_class ( ) const [inline]

Definition at line 102 of file expression.h.

10.215.3.46 mpq_class coco::num::Number::get_q_class ( ) const [inline]

Definition at line 102 of file search_graph.cc.

10.215.3.47 mpq_class coco::num::Number::get_q_class ( ) const [inline]

Definition at line 102 of file search_graph.cc.

10.215.3.48 mpq_class coco::num::Number::get_q_class ( ) const [inline]

Definition at line 102 of file search_graph.cc.

10.215.3.49 double coco::num::Number::get_r ( ) const [inline]

Definition at line 106 of file search_graph.cc.

10.215.3.50 double coco::num::Number::get_r ( ) const [inline]

Definition at line 106 of file search_graph.cc.

10.215.3.51 double coco::num::Number::get_r ( ) const [inline]

Definition at line 106 of file expression.h.

10.215.3.52 double coco::num::Number::get_r ( ) const [inline]

Definition at line 106 of file search_graph.cc.

10.215.3.53 str coco::num::Number::get_s ( ) const

10.215.3.54 str coco::num::Number::get_s ( ) const

10.215.3.55 str coco::num::Number::get_s ( ) const

10.215.3.56 str coco::num::Number::get_s ( ) const

10.215.3.57 bool coco::num::Number::operator!= (const Number & *n*) const [inline]

Definition at line 236 of file search_graph.cc.

10.215.3.58 bool coco::num::Number::operator!= (const Number & *n*) const [inline]

Definition at line 236 of file expression.h.

10.215.3.59 bool coco::num::Number::operator!= (const Number & *n*) const [inline]

Definition at line 236 of file search_graph.cc.

10.215.3.60 bool coco::num::Number::operator!= (const Number & *n*) const [inline]

Definition at line 236 of file search_graph.cc.

10.215.3.61 Number coco::num::Number::operator∗ (const Number & *n*) const

10.215.3.62 Number coco::num::Number::operator∗ (const Number & *n*) const

10.215.3.63 Number coco::num::Number::operator∗ (const Number & *n*) const

10.215.3.64 Number coco::num::Number::operator∗ (const Number & *n*) const

10.215.3.65 const Number& coco::num::Number::operator*= (const Number & *n*)

10.215.3.66 const Number& coco::num::Number::operator*= (const Number & *n*)

10.215.3.67 const Number& coco::num::Number::operator*= (const Number & *n*)

10.215.3.68 const Number& coco::num::Number::operator*= (const Number & *n*)

10.215.3.69 Number coco::num::Number::operator+ (const Number & *n*) const

10.215.3.70 Number coco::num::Number::operator+ (const Number & *n*) const

10.215.3.71 Number coco::num::Number::operator+ (const Number & *n*) const

10.215.3.72 Number coco::num::Number::operator+ (const Number & *n*) const

10.215.3.73 const Number& coco::num::Number::operator+= (const Number & *n*)

10.215.3.74 const Number& coco::num::Number::operator+= (const Number & *n*)

10.215.3.75 const Number& coco::num::Number::operator+= (const Number & *n*)

10.215.3.76 const Number& coco::num::Number::operator+= (const Number & *n*)

10.215.3.77 Number coco::num::Number::operator- (const Number & *n*) const

10.215.3.78 Number coco::num::Number::operator- (const Number & *n*) const

10.215.3.79 Number coco::num::Number::operator- (const Number & *n*) const

10.215.3.80 Number coco::num::Number::operator- (const Number & *n*) const

10.215.3.81 const Number& coco::num::Number::operator-= (const Number & *n*)

10.215.3.82 const Number& coco::num::Number::operator-= (const Number & *n*)

10.215.3.83 const Number& coco::num::Number::operator-= (const Number & *n*)

10.215.3.84 const Number& coco::num::Number::operator-= (const Number & *n*)

10.215.3.85 Number coco::num::Number::operator/ (const Number & *n*) const

10.215.3.86 Number coco::num::Number::operator/ (const Number & *n*) const

10.215.3.87 Number coco::num::Number::operator/ (const Number & *n*) const

10.215.3.88 Number coco::num::Number::operator/ (const Number & *n*) const

10.215.3.89 const Number& coco::num::Number::operator/= (const Number & *n*)

10.215.3.90 const Number& coco::num::Number::operator/= (const Number & *n*)

10.215.3.91 const Number& coco::num::Number::operator/= (const Number & *n*)

10.215.3.92 const Number& coco::num::Number::operator/= (const Number & *n*)

10.215.3.93 bool coco::num::Number::operator< (const Number & *n*) const [inline]

Definition at line 244 of file search_graph.cc.

10.215.3.94 bool coco::num::Number::operator< (const Number & *n*) const [inline]

Definition at line 244 of file expression.h.

10.215.3.95 bool coco::num::Number::operator< (const Number & *n*) const [inline]

Definition at line 244 of file search_graph.cc.

10.215.3.96 bool coco::num::Number::operator< (const Number & *n*) const [inline]

Definition at line 244 of file search_graph.cc.

10.215.3.97 bool coco::num::Number::operator<= (const Number & *n*) const [inline]

Definition at line 242 of file search_graph.cc.

10.215.3.98 bool coco::num::Number::operator<= (const Number & *n*) const [inline]

Definition at line 242 of file search_graph.cc.

10.215.3.99 bool coco::num::Number::operator<= (const Number & *n*) const [inline]

Definition at line 242 of file expression.h.

10.215.3.100 bool coco::num::Number::operator<= (const Number & *n*) const [inline]

Definition at line 242 of file search_graph.cc.

10.215.3.101 const Number& coco::num::Number::operator= (int *n*)

10.215.3.102 const Number& coco::num::Number::operator= (int *n*)

10.215.3.103 const Number& coco::num::Number::operator= (int *n*)

10.215.3.104 const Number& coco::num::Number::operator= (int *n*)

10.215.3.105 const Number& coco::num::Number::operator= (unsigned *n*)

10.215.3.106 const Number& coco::num::Number::operator= (unsigned *n*)

10.215.3.107 const Number& coco::num::Number::operator= (unsigned *n*)

10.215.3.108 const Number& coco::num::Number::operator= (unsigned *n*)

10.215.3.109 const Number& coco::num::Number::operator= (double *d*)

10.215.3.110 const Number& coco::num::Number::operator= (double *d*)

10.215.3.111 const Number& coco::num::Number::operator= (double *d*)

10.215.3.112 const Number& coco::num::Number::operator= (double *d*)

10.215.3.113 const Number& coco::num::Number::operator= (cprat *q*)

10.215.3.114 const Number& coco::num::Number::operator= (cprat *q*)

10.215.3.115 const Number& coco::num::Number::operator= (cprat *q*)

10.215.3.116 const Number& coco::num::Number::operator= (cprat *q*)

10.215.3.117 const Number& coco::num::Number::operator= (const mpq_class & *q*)

10.215.3.118 const Number& coco::num::Number::operator= (const mpq_class & *q*)

10.215.3.119 const Number& coco::num::Number::operator= (const mpq_class & *q*)

10.215.3.120 const Number& coco::num::Number::operator= (const mpq_class & *q*)

10.215.3.121 const Number& coco::num::Number::operator= (const intv & *i*)

10.215.3.122 const Number& coco::num::Number::operator= (const intv & *i*)

10.215.3.123 const Number& coco::num::Number::operator= (const intv & *i*)

10.215.3.124 const Number& coco::num::Number::operator= (const intv & *i*)

10.215.3.125 const Number& coco::num::Number::operator= (const str & *s*)

10.215.3.126 const Number& coco::num::Number::operator= (const str & *s*)

10.215.3.127 const Number& coco::num::Number::operator= (const str & *s*)

10.215.3.128 const Number& coco::num::Number::operator= (const str & *s*)

10.215.3.129 const Number& coco::num::Number::operator= (const Number & *n*)

10.215.3.130 const Number& coco::num::Number::operator= (const Number & *n*)

10.215.3.131 const Number& coco::num::Number::operator= (const Number & *n*)

10.215.3.132 const Number& coco::num::Number::operator= (const Number & *n*)

10.215.3.133 bool coco::num::Number::operator== (const Number & *n*) const [inline]

Definition at line 234 of file search_graph.cc.

10.215.3.134 bool coco::num::Number::operator== (const Number & *n*) const [inline]

Definition at line 234 of file expression.h.

10.215.3.135 bool coco::num::Number::operator== (const Number & *n*) const [inline]

Definition at line 234 of file search_graph.cc.

10.215.3.136 bool coco::num::Number::operator== (const Number & *n*) const [inline]

Definition at line 234 of file search_graph.cc.

10.215.3.137 bool coco::num::Number::operator> (const Number & *n*) const [inline]

Definition at line 240 of file search_graph.cc.

10.215.3.138 bool coco::num::Number::operator> (const Number & *n*) const [inline]

Definition at line 240 of file search_graph.cc.

10.215.3.139 bool coco::num::Number::operator> (const Number & *n*) const [inline]

Definition at line 240 of file search_graph.cc.

10.215.3.140 bool coco::num::Number::operator> (const Number & *n*) const [inline]

Definition at line 240 of file expression.h.

10.215.3.141 bool coco::num::Number::operator>= (const Number & *n*) const [inline]

Definition at line 238 of file search_graph.cc.

10.215.3.142 bool coco::num::Number::operator>= (const Number & *n*) const [inline]

Definition at line 238 of file search_graph.cc.

10.215.3.143 bool coco::num::Number::operator>= (const Number & *n*) const [inline]

Definition at line 238 of file search_graph.cc.

10.215.3.144 bool coco::num::Number::operator>= (const Number & *n*) const [inline]

Definition at line 238 of file expression.h.

10.215.3.145 Number coco::num::Number::operator^ (const Number & *n*) const [inline]

Definition at line 194 of file search_graph.cc.

10.215.3.146 Number coco::num::Number::operator^ (const Number & *n*) const [inline]

Definition at line 194 of file expression.h.

10.215.3.147 Number coco::num::Number::operator^ (const Number & *n*) const [inline]

Definition at line 194 of file search_graph.cc.

10.215.3.148 Number coco::num::Number::operator^ (const Number & *n*) const [inline]

Definition at line 194 of file search_graph.cc.

10.215.3.149 const Number& coco::num::Number::operator^= (const Number & *n*) [inline]

Definition at line 196 of file search_graph.cc.

10.215.3.150 const Number& coco::num::Number::operator^= (const Number & *n*) [inline]

Definition at line 196 of file expression.h.

10.215.3.151 const Number& coco::num::Number::operator^= (const Number & *n*) [inline]

Definition at line 196 of file search_graph.cc.

10.215.3.152 const Number& coco::num::Number::operator^= (const Number & *n*) [inline]

Definition at line 196 of file search_graph.cc.

10.215.3.153 bool coco::num::Number::peq (const Number & *n*) const

10.215.3.154 bool coco::num::Number::peq (const Number & *n*) const

10.215.3.155 bool coco::num::Number::peq (const Number & *n*) const

10.215.3.156 bool coco::num::Number::peq (const Number & *n*) const

10.215.3.157 bool coco::num::Number::pge (const Number & *n*) const

10.215.3.158 bool coco::num::Number::pge (const Number & *n*) const

10.215.3.159 bool coco::num::Number::pge (const Number & *n*) const

10.215.3.160 bool coco::num::Number::pge (const Number & *n*) const

10.215.3.161 bool coco::num::Number::pgt (const Number & *n*) const

10.215.3.162 bool coco::num::Number::pgt (const Number & *n*) const

10.215.3.163 bool coco::num::Number::pgt (const Number & *n*) const

10.215.3.164 bool coco::num::Number::pgt (const Number & *n*) const

10.215.3.165 bool coco::num::Number::ple (const Number & *n*) const

10.215.3.166 bool coco::num::Number::ple (const Number & *n*) const

10.215.3.167 bool coco::num::Number::ple (const Number & *n*) const

10.215.3.168 bool coco::num::Number::ple (const Number & *n*) const

10.215.3.169 bool coco::num::Number::plt (const Number & *n*) const

10.215.3.170 bool coco::num::Number::plt (const Number & *n*) const

10.215.3.171 bool coco::num::Number::plt (const Number & *n*) const

10.215.3.172 bool coco::num::Number::plt (const Number & *n*) const

10.215.3.173 bool coco::num::Number::pne (const Number & *n*) const

10.215.3.174 bool coco::num::Number::pne (const Number & *n*) const

10.215.3.175 bool coco::num::Number::pne (const Number & *n*) const

10.215.3.176 bool coco::num::Number::pne (const Number & *n*) const

10.215.3.177 bool coco::num::Number::seq (const Number & *n*) const

10.215.3.178 bool coco::num::Number::seq (const Number & *n*) const

10.215.3.179 bool coco::num::Number::seq (const Number & *n*) const

10.215.3.180 bool coco::num::Number::seq (const Number & *n*) const

10.215.3.181 bool coco::num::Number::sge (const Number & *n*) const

10.215.3.182 bool coco::num::Number::sge (const Number & *n*) const

10.215.3.183 bool coco::num::Number::sge (const Number & *n*) const

10.215.3.184 bool coco::num::Number::sge (const Number & *n*) const

10.215.3.185 bool coco::num::Number::sgt (const Number & *n*) const

10.215.3.186 bool coco::num::Number::sgt (const Number & *n*) const

10.215.3.187 bool coco::num::Number::sgt (const Number & *n*) const

10.215.3.188 bool coco::num::Number::sgt (const Number & *n*) const

10.215.3.189 bool coco::num::Number::sle (const Number & *n*) const

10.215.3.190 bool coco::num::Number::sle (const Number & *n*) const

10.215.3.191 bool coco::num::Number::sle (const Number & *n*) const

10.215.3.192 bool coco::num::Number::sle (const Number & *n*) const

10.215.3.193 bool coco::num::Number::slt (const Number & *n*) const

10.215.3.194 bool coco::num::Number::slt (const Number & *n*) const

10.215.3.195 bool coco::num::Number::slt (const Number & *n*) const

10.215.3.196 bool coco::num::Number::slt (const Number & *n*) const

10.215.3.197 bool coco::num::Number::sne (const Number & *n*) const

10.215.3.198 bool coco::num::Number::sne (const Number & *n*) const

10.215.3.199 bool coco::num::Number::sne (const Number & *n*) const

10.215.3.200 bool coco::num::Number::sne (const Number & *n*) const

10.215.3.201 str coco::num::Number::tooutputstr ( ) const

10.215.3.202 str coco::num::Number::tooutputstr ( ) const

10.215.3.203 str coco::num::Number::tooutputstr ( ) const

10.215.3.204 str coco::num::Number::tooutputstr ( ) const

10.215.3.205 str coco::num::Number::tostr ( ) const

10.215.3.206 str coco::num::Number::tostr ( ) const

10.215.3.207 str coco::num::Number::tostr ( ) const

10.215.3.208 str coco::num::Number::tostr ( ) const

10.215.4 Friends And Related Function Documentation

10.215.4.1 Number abs (const Number & *x*) [friend]

10.215.4.2 Number abs (const Number & *x*) [friend]

10.215.4.3 Number abs (const Number & *x*) [friend]

10.215.4.4 Number abs (const Number & *x*) [friend]

10.215.4.5 Number acos (const Number & *x*) [friend]

10.215.4.6 Number acos (const Number & *x*) [friend]

10.215.4.7 Number acos (const Number & *x*) [friend]

10.215.4.8 Number acos (const Number & *x*) [friend]

10.215.4.9 Number acosh (const Number & *x*) [friend]

10.215.4.10 **Number acosh (const Number & *x*)** [friend]

10.215.4.11 **Number acosh (const Number & *x*)** [friend]

10.215.4.12 **Number acosh (const Number & *x*)** [friend]

10.215.4.13 **Number acot (const Number & *x*)** [friend]

10.215.4.14 **Number acot (const Number & *x*)** [friend]

10.215.4.15 **Number acot (const Number & *x*)** [friend]

10.215.4.16 **Number acot (const Number & *x*)** [friend]

10.215.4.17 **Number acoth (const Number & *x*)** [friend]

10.215.4.18 **Number acoth (const Number & *x*)** [friend]

10.215.4.19 **Number acoth (const Number & *x*)** [friend]

10.215.4.20 **Number acoth (const Number & *x*)** [friend]

10.215.4.21 **Number asin (const Number & *x*)** [friend]

10.215.4.22 **Number asin (const Number & *x*)** [friend]

10.215.4.23 **Number asin (const Number & *x*)** [friend]

10.215.4.24 **Number asin (const Number & *x*)** [friend]

10.215.4.25 **Number asinh (const Number & *x*)** [friend]

10.215.4.26 **Number asinh (const Number & *x*)** [friend]

10.215.4.27 **Number asinh (const Number & *x*)** [friend]

10.215.4.28 **Number asinh (const Number & *x*)** [friend]

10.215.4.29 **Number atan (const Number & *x*)** [friend]

10.215.4.30 **Number atan (const Number & *x*)** [friend]

10.215.4.31 **Number atan (const Number & *x*)** [friend]

10.215.4.32 **Number atan (const Number & *x*)** [friend]

10.215.4.33 **Number atanh (const Number & *x*)** [friend]

10.215.4.34 **Number atanh (const Number & *x*)** [friend]

10.215.4.35 **Number atanh (const Number & *x*)** [friend]

10.215.4.36 **Number atanh (const Number & *x*)** [friend]

10.215.4.37 **Number cos (const Number & *x*)** [friend]

10.215.4.38 **Number cos (const Number & *x*)** [friend]

10.215.4.39 **Number cos (const Number & *x*)** [friend]

10.215.4.40 **Number cos (const Number & *x*)** [friend]

10.215.4.41 **Number cosh (const Number & *x*)** [friend]

10.215.4.42 **Number cosh (const Number & *x*)** [friend]

10.215.4.43 **Number cosh (const Number & *x*)** [friend]

10.215.4.44 **Number cosh (const Number & *x*)** [friend]

10.215.4.45 **Number cot (const Number & *x*)** [friend]

10.215.4.46 **Number cot (const Number & *x*)** [friend]

10.215.4.47 **Number cot (const Number & *x*)** [friend]

10.215.4.48 **Number cot (const Number & *x*)** [friend]

10.215.4.49 **Number coth (const Number & *x*)** [friend]

10.215.4.50 **Number coth (const Number & *x*)** [friend]

10.215.4.51 **Number coth (const Number & *x*)** [friend]

10.215.4.52 **Number coth (const Number & *x*)** [friend]

10.215.4.53 **Number exp (const Number & *x*)** [friend]

10.215.4.54 **Number exp (const Number & *x*)** [friend]

10.215.4.55 **Number exp (const Number & *x*)** [friend]

10.215.4.56 **Number exp (const Number & *x*)** [friend]

10.215.4.57 **Number exp10 (const Number & *x*)** [friend]

10.215.4.58 **Number exp10 (const Number & *x*)** [friend]

10.215.4.59 **Number exp10 (const Number & *x*)** [friend]

10.215.4.60 **Number exp10 (const Number & *x*)** [friend]

10.215.4.61 **Number exp2 (const Number & *x*)** [friend]

10.215.4.62 **Number exp2 (const Number & *x*)** [friend]

10.215.4.63 **Number exp2 (const Number & *x*)** [friend]

10.215.4.64 **Number exp2 (const Number & *x*)** [friend]

10.215.4.65 **Number expm1 (const Number & *x*)** [friend]

10.215.4.66 **Number expm1 (const Number & *x*)** [friend]

10.215.4.67 **Number expm1 (const Number & *x*)** [friend]

10.215.4.68 **Number expm1 (const Number & *x*)** [friend]

10.215.4.69 **Number log (const Number & *x*)** [friend]

10.215.4.70 **Number log (const Number & *x*)** [friend]

10.215.4.71 **Number log (const Number & *x*)** [friend]

10.215.4.72 **Number log (const Number & *x*)** [friend]

10.215.4.73 **Number log10 (const Number & *x*)** [friend]

10.215.4.74 **Number log10 (const Number & *x*)** [friend]

10.215.4.75 **Number log10 (const Number & *x*)** [friend]

10.215.4.76 **Number log10 (const Number & *x*)** [friend]

10.215.4.77 **Number log1p (const Number & *x*)** [friend]

10.215.4.78 **Number log1p (const Number & *x*)** [friend]

10.215.4.79 **Number log1p (const Number & *x*)** [friend]

10.215.4.80 **Number log1p (const Number & *x*)** [friend]

10.215.4.81 **Number log2 (const Number & *x*)** [friend]

10.215.4.82 **Number log2 (const Number & *x*)** [friend]

10.215.4.83 **Number log2 (const Number & *x*)** [friend]

10.215.4.84 **Number log2 (const Number & *x*)** [friend]

10.215.4.85 **std::ostream& operator<< (std::ostream & *os*, const Number & *n*)** [friend]

10.215.4.86 **std::ostream& operator<< (std::ostream & *os*, const Number & *n*)** [friend]

10.215.4.87 **std::ostream& operator<< (std::ostream & *os*, const Number & *n*)** [friend]

10.215.4.88 std::ostream& operator<< (std::ostream & *os,* const Number & *n*) [friend]

10.215.4.89 std::istream& operator>> (std::istream & *is,* Number & *n*) [friend]

10.215.4.90 std::istream& operator>> (std::istream & *is,* Number & *n*) [friend]

10.215.4.91 std::istream& operator>> (std::istream & *is,* Number & *n*) [friend]

10.215.4.92 std::istream& operator>> (std::istream & *is,* Number & *n*) [friend]

10.215.4.93 Number pow (const Number & *x,* const Number & *y*) [friend]

10.215.4.94 Number pow (const Number & *x,* const Number & *y*) [friend]

10.215.4.95 Number pow (const Number & *x,* const Number & *y*) [friend]

10.215.4.96 Number pow (const Number & *x,* const Number & *y*) [friend]

10.215.4.97 Number power (const Number & *x,* int *n*) [friend]

10.215.4.98 Number power (const Number & *x,* int *n*) [friend]

10.215.4.99 Number power (const Number & *x,* int *n*) [friend]

10.215.4.100 Number power (const Number & *x,* int *n*) [friend]

10.215.4.101 Number sin (const Number & *x*) [friend]

10.215.4.102 Number sin (const Number & *x*) [friend]

10.215.4.103 Number sin (const Number & *x*) [friend]

10.215.4.104 Number sin (const Number & *x*) [friend]

10.215.4.105 Number sinh (const Number & *x*) [friend]

10.215.4.106 Number sinh (const Number & *x*) [friend]

10.215.4.107 Number sinh (const Number & *x*) [friend]

10.215.4.108 Number sinh (const Number & *x*) [friend]

10.215.4.109 Number sqr (const Number & *x*) [friend]

10.215.4.110 Number sqr (const Number & *x*) [friend]

10.215.4.111 Number sqr (const Number & *x*) [friend]

10.215.4.112 Number sqr (const Number & *x*) [friend]

10.215.4.113 Number sqrt (const Number & *x*) [friend]

10.215.4.114 Number sqrt (const Number & x) [friend]

10.215.4.115 Number sqrt (const Number & x) [friend]

10.215.4.116 Number sqrt (const Number & x) [friend]

10.215.4.117 Number tan (const Number & x) [friend]

10.215.4.118 Number tan (const Number & x) [friend]

10.215.4.119 Number tan (const Number & x) [friend]

10.215.4.120 Number tan (const Number & x) [friend]

10.215.4.121 Number tanh (const Number & x) [friend]

10.215.4.122 Number tanh (const Number & x) [friend]

10.215.4.123 Number tanh (const Number & x) [friend]

10.215.4.124 Number tanh (const Number & x) [friend]

The documentation for this class was generated from the following file:

- Number.h

10.216 coco::coco::num::Number Class Reference

Public Member Functions

- Number ()
- Number (int n)
- Number (unsigned n)
- Number (double d)
- Number (cprat q)
- Number (const mpq_class &q)
- Number (const intv &i)
- Number (const str &s)
- Number (const Number &n)
- ~Number ()
- const Number & operator= (int n)
- const Number & operator= (unsigned n)
- const Number & operator= (double d)
- const Number & operator= (cprat q)
- const Number & operator= (const mpq_class &q)
- const Number & operator= (const intv &i)
- const Number & operator= (const str &s)
- const Number & operator= (const Number &n)
- cprat get_q () const
- mpq_class get_q_class () const

- str get_s () const
- intv get_i () const
- double get_r () const
- numtypes get_numtype ()
- const Number & approx ()
- str tostr () const
- str tooutputstr () const
- void fromstr (str s1)
- Number operator+ (const Number &n) const
- Number operator- (const Number &n) const
- Number operator∗ (const Number &n) const
- Number operator/ (const Number &n) const
- const Number & operator+= (const Number &n)
- const Number & operator-= (const Number &n)
- const Number & operator∗= (const Number &n)
- const Number & operator/= (const Number &n)
- Number operator^ (const Number &n) const
- const Number & operator^= (const Number &n)
- bool seq (const Number &n) const
- bool sne (const Number &n) const
- bool sge (const Number &n) const
- bool sgt (const Number &n) const
- bool sle (const Number &n) const
- bool slt (const Number &n) const
- bool ceq (const Number &n) const
- bool cne (const Number &n) const
- bool cge (const Number &n) const
- bool cgt (const Number &n) const
- bool cle (const Number &n) const
- bool clt (const Number &n) const
- bool peq (const Number &n) const
- bool pne (const Number &n) const
- bool pge (const Number &n) const
- bool pgt (const Number &n) const
- bool ple (const Number &n) const
- bool plt (const Number &n) const
- bool operator== (const Number &n) const
- bool operator!= (const Number &n) const
- bool operator>= (const Number &n) const
- bool operator> (const Number &n) const
- bool operator<= (const Number &n) const
- bool operator< (const Number &n) const

Friends

- std::ostream & operator<< (std::ostream &os, const Number &n)
- std::istream & operator>> (std::istream &is, Number &n)
- Number acos (const Number &x)
- Number abs (const Number &x)
- Number acosh (const Number &x)
- Number acot (const Number &x)
- Number acoth (const Number &x)
- Number asin (const Number &x)
- Number asinh (const Number &x)
- Number atan (const Number &x)
- Number atanh (const Number &x)
- Number cos (const Number &x)
- Number cosh (const Number &x)
- Number cot (const Number &x)
- Number coth (const Number &x)
- Number exp (const Number &x)
- Number exp10 (const Number &x)
- Number exp2 (const Number &x)
- Number expm1 (const Number &x)
- Number log (const Number &x)
- Number log10 (const Number &x)
- Number log1p (const Number &x)
- Number log2 (const Number &x)
- Number power (const Number &x, int n)
- Number pow (const Number &x, const Number &y)
- Number sin (const Number &x)
- Number sinh (const Number &x)
- Number sqr (const Number &x)
- Number sqrt (const Number &x)
- Number tan (const Number &x)
- Number tanh (const Number &x)

10.216.1 Constructor & Destructor Documentation

10.216.1.1 coco::coco::num::Number::Number ( )

10.216.1.2 coco::coco::num::Number::Number (int *n*)

10.216.1.3 coco::coco::num::Number::Number (unsigned *n*)

10.216.1.4 coco::coco::num::Number::Number (double *d*)

10.216.1.5 coco::coco::num::Number::Number (cprat *q*)

10.216.1.6 coco::coco::num::Number::Number (const mpq_class & *q*)

10.216.1.7 coco::coco::num::Number::Number (const intv & *i*)

10.216.1.8 coco::coco::num::Number::Number (const str & *s*)

10.216.1.9 coco::coco::num::Number::Number (const Number & *n*)

10.216.1.10 coco::coco::num::Number::∼Number ( )

10.216.2 Member Function Documentation

10.216.2.1 const Number& coco::coco::num::Number::approx ( ) [inline]

Definition at line 108 of file search_graph.cc.

10.216.2.2 bool coco::coco::num::Number::ceq (const Number & *n*) const

10.216.2.3 bool coco::coco::num::Number::cge (const Number & *n*) const

10.216.2.4 bool coco::coco::num::Number::cgt (const Number & *n*) const

10.216.2.5 bool coco::coco::num::Number::cle (const Number & *n*) const

10.216.2.6 bool coco::coco::num::Number::clt (const Number & *n*) const

10.216.2.7 bool coco::coco::num::Number::cne (const Number & *n*) const

10.216.2.8 void coco::coco::num::Number::fromstr (str *s1*)

10.216.2.9 intv coco::coco::num::Number::get_i ( ) const [inline]

Definition at line 105 of file search_graph.cc.

10.216.2.10 numtypes coco::coco::num::Number::get_numtype ( ) [inline]

Definition at line 107 of file search_graph.cc.

10.216.2.11 cprat coco::coco::num::Number::get_q ( ) const

10.216.2.12 mpq_class coco::coco::num::Number::get_q_class ( ) const [inline]

Definition at line 102 of file search_graph.cc.

10.216.2.13 double coco::coco::num::Number::get_r ( ) const [inline]

Definition at line 106 of file search_graph.cc.

10.216.2.14 str coco::coco::num::Number::get_s ( ) const

10.216.2.15 bool coco::coco::num::Number::operator!= (const Number & *n*) const [inline]

Definition at line 236 of file search_graph.cc.

10.216.2.16 Number coco::coco::num::Number::operator∗ (const Number & *n*) const

10.216.2.17 const Number& coco::coco::num::Number::operator∗= (const Number & *n*)

10.216.2.18 Number coco::coco::num::Number::operator+ (const Number & *n*) const

10.216.2.19 const Number& coco::coco::num::Number::operator+= (const Number & *n*)

10.216.2.20 Number coco::coco::num::Number::operator- (const Number & *n*) const

10.216.2.21 const Number& coco::coco::num::Number::operator-= (const Number & *n*)

10.216.2.22 Number coco::coco::num::Number::operator/ (const Number & *n*) const

10.216.2.23 const Number& coco::coco::num::Number::operator/= (const Number & *n*)

10.216.2.24 bool coco::coco::num::Number::operator< (const Number & *n*) const [inline]

Definition at line 244 of file search_graph.cc.

10.216.2.25 bool coco::coco::num::Number::operator<= (const Number & *n*) const [inline]

Definition at line 242 of file search_graph.cc.

10.216.2.26 const Number& coco::coco::num::Number::operator= (int *n*)

10.216.2.27 const Number& coco::coco::num::Number::operator= (unsigned *n*)

10.216.2.28 const Number& coco::coco::num::Number::operator= (double *d*)

10.216.2.29 const Number& coco::coco::num::Number::operator= (cprat *q*)

10.216.2.30 const Number& coco::coco::num::Number::operator= (const mpq_class & *q*)

10.216.2.31 const Number& coco::coco::num::Number::operator= (const intv & *i*)

10.216.2.32 const Number& coco::coco::num::Number::operator= (const str & *s*)

10.216.2.33 const Number& coco::coco::num::Number::operator= (const Number & *n*)

10.216.2.34 bool coco::coco::num::Number::operator== (const Number & *n*) const [inline]

Definition at line 234 of file search_graph.cc.

10.216.2.35 bool coco::coco::num::Number::operator> (const Number & *n*) const [inline]

Definition at line 240 of file search_graph.cc.

10.216.2.36 bool coco::coco::num::Number::operator>= (const Number & *n*) const [inline]

Definition at line 238 of file search_graph.cc.

10.216.2.37 Number coco::coco::num::Number::operator^ (const Number & *n*) const [inline]

Definition at line 194 of file search_graph.cc.

10.216.2.38 const Number& coco::coco::num::Number::operator^= (const Number & *n*) [inline]

Definition at line 196 of file search_graph.cc.

10.216.2.39 bool coco::coco::num::Number::peq (const Number & *n*) const

10.216.2.40 bool coco::coco::num::Number::pge (const Number & *n*) const

10.216.2.41 bool coco::coco::num::Number::pgt (const Number & *n*) const

10.216.2.42 bool coco::coco::num::Number::ple (const Number & *n*) const

10.216.2.43 bool coco::coco::num::Number::plt (const Number & *n*) const

10.216.2.44 bool coco::coco::num::Number::pne (const Number & *n*) const

10.216.2.45 bool coco::coco::num::Number::seq (const Number & *n*) const

10.216.2.46 bool coco::coco::num::Number::sge (const Number & *n*) const

10.216.2.47 bool coco::coco::num::Number::sgt (const Number & *n*) const

10.216.2.48 bool coco::coco::num::Number::sle (const Number & *n*) const

10.216.2.49 bool coco::coco::num::Number::slt (const Number & *n*) const

10.216.2.50 bool coco::coco::num::Number::sne (const Number & *n*) const

10.216.2.51 str coco::coco::num::Number::tooutputstr ( ) const

10.216.2.52 str coco::coco::num::Number::tostr ( ) const

10.216.3 Friends And Related Function Documentation

10.216.3.1 Number abs (const Number & *x*) [friend]

10.216.3.2 Number acos (const Number & *x*) [friend]

10.216.3.3 Number acosh (const Number & *x*) [friend]

10.216.3.4 Number acot (const Number & *x*) [friend]

10.216.3.5 Number acoth (const Number & *x*) [friend]

10.216.3.6 Number asin (const Number & *x*) [friend]

10.216.3.7 Number asinh (const Number & *x*) [friend]

10.216.3.8 Number atan (const Number & *x*) [friend]

10.216.3.9 Number atanh (const Number & *x*) [friend]

10.216.3.10 Number cos (const Number & *x*) [friend]

10.216.3.11 Number cosh (const Number & *x*) [friend]

10.216.3.12 Number cot (const Number & *x*) [friend]

10.216.3.13 Number coth (const Number & *x*) [friend]

10.216.3.14 Number exp (const Number & *x*) [friend]

10.216.3.15 Number exp10 (const Number & *x*) [friend]

10.216.3.16 Number exp2 (const Number & *x*) [friend]

10.216.3.17 Number expm1 (const Number & *x*) [friend]

10.216.3.18 Number log (const Number & *x*) [friend]

10.216.3.19 Number log10 (const Number & *x*) [friend]

10.216.3.20 Number log1p (const Number & *x*) [friend]

10.216.3.21 Number log2 (const Number & *x*) [friend]

10.216.3.22 std::ostream& operator<< (std::ostream & *os*, const Number & *n*) [friend]

10.216.3.23 std::istream& operator>> (std::istream & *is*, Number & *n*) [friend]

10.216.3.24 Number pow (const Number & *x*, const Number & *y*) [friend]

10.216.3.25 Number power (const Number & *x*, int *n*) [friend]

10.216.3.26 Number sin (const Number & *x*) [friend]

10.216.3.27 Number sinh (const Number & *x*) [friend]

10.216.3.28 Number sqr (const Number & *x*) [friend]

10.216.3.29 Number sqrt (const Number & *x*) [friend]

10.216.3.30 Number tan (const Number & *x*) [friend]

10.216.3.31 Number tanh (const Number & *x*) [friend]

The documentation for this class was generated from the following file:

- Number.h

10.217 num::Number Class Reference

`#include <Number.h>`

Public Member Functions

- Number ()
- Number (int n)
- Number (unsigned n)
- Number (double d)
- Number (cprat q)
- Number (const mpq_class &q)
- Number (const intv &i)
- Number (const str &s)
- Number (const Number &n)
- ~Number ()
- const Number & operator= (int n)
- const Number & operator= (unsigned n)
- const Number & operator= (double d)
- const Number & operator= (cprat q)
- const Number & operator= (const mpq_class &q)
- const Number & operator= (const intv &i)
- const Number & operator= (const str &s)
- const Number & operator= (const Number &n)
- cprat get_q () const
- mpq_class get_q_class () const
- str get_s () const
- intv get_i () const
- double get_r () const
- numtypes get_numtype ()
- const Number & approx ()
- str tostr () const
- str tooutputstr () const
- void fromstr (str s1)
- Number operator+ (const Number &n) const
- Number operator- (const Number &n) const
- Number operator* (const Number &n) const
- Number operator/ (const Number &n) const
- const Number & operator+= (const Number &n)
- const Number & operator-= (const Number &n)
- const Number & operator*= (const Number &n)
- const Number & operator/= (const Number &n)
- Number operator^ (const Number &n) const
- const Number & operator^= (const Number &n)
- bool seq (const Number &n) const
- bool sne (const Number &n) const
- bool sge (const Number &n) const
- bool sgt (const Number &n) const
- bool sle (const Number &n) const

- bool slt (const Number &n) const
- bool ceq (const Number &n) const
- bool cne (const Number &n) const
- bool cge (const Number &n) const
- bool cgt (const Number &n) const
- bool cle (const Number &n) const
- bool clt (const Number &n) const
- bool peq (const Number &n) const
- bool pne (const Number &n) const
- bool pge (const Number &n) const
- bool pgt (const Number &n) const
- bool ple (const Number &n) const
- bool plt (const Number &n) const
- bool operator== (const Number &n) const
- bool operator!= (const Number &n) const
- bool operator>= (const Number &n) const
- bool operator> (const Number &n) const
- bool operator<= (const Number &n) const
- bool operator< (const Number &n) const

Friends

- std::ostream & operator<< (std::ostream &os, const Number &n)
- std::istream & operator>> (std::istream &is, Number &n)
- Number acos (const Number &x)
- Number abs (const Number &x)
- Number acosh (const Number &x)
- Number acot (const Number &x)
- Number acoth (const Number &x)
- Number asin (const Number &x)
- Number asinh (const Number &x)
- Number atan (const Number &x)
- Number atanh (const Number &x)
- Number cos (const Number &x)
- Number cosh (const Number &x)
- Number cot (const Number &x)
- Number coth (const Number &x)
- Number exp (const Number &x)
- Number exp10 (const Number &x)
- Number exp2 (const Number &x)
- Number expm1 (const Number &x)
- Number log (const Number &x)
- Number log10 (const Number &x)
- Number log1p (const Number &x)
- Number log2 (const Number &x)
- Number power (const Number &x, int n)
- Number pow (const Number &x, const Number &y)
- Number sin (const Number &x)
- Number sinh (const Number &x)
- Number sqr (const Number &x)
- Number sqrt (const Number &x)
- Number tan (const Number &x)
- Number tanh (const Number &x)

10.217.1 Constructor & Destructor Documentation

10.217.1.1 num::Number::Number ( )

Definition at line 108 of file Number.cc.

10.217.1.2 num::Number::Number (int *n*)

Definition at line 112 of file Number.cc.

10.217.1.3 num::Number::Number (unsigned *n*)

Definition at line 117 of file Number.cc.

10.217.1.4 num::Number::Number (double *d*)

Definition at line 122 of file Number.cc.

10.217.1.5 num::Number::Number (cprat *q*)

Definition at line 126 of file Number.cc.

10.217.1.6 num::Number::Number (const mpq_class & *q*)

Definition at line 133 of file Number.cc.

10.217.1.7 num::Number::Number (const intv & *i*)

Definition at line 141 of file Number.cc.

10.217.1.8 num::Number::Number (const str & *s*)

Definition at line 146 of file Number.cc.

10.217.1.9 num::Number::Number (const Number & *n*)

Definition at line 152 of file Number.cc.

10.217.1.10 num::Number::∼Number ( )

Definition at line 158 of file Number.cc.

10.217.2 Member Function Documentation

10.217.2.1 const Number& num::Number::approx ( ) [inline]

Definition at line 108 of file Number.h.

10.217.2.2 bool num::Number::ceq (const Number & *n*) const

Definition at line 1022 of file Number.cc.

10.217.2.3 bool num::Number::cge (const Number & *n*) const

Definition at line 1048 of file Number.cc.

10.217.2.4 bool num::Number::cgt (const Number & *n*) const

Definition at line 1058 of file Number.cc.

10.217.2.5 bool num::Number::cle (const Number & *n*) const

Definition at line 1068 of file Number.cc.

10.217.2.6 bool num::Number::clt (const Number & *n*) const

Definition at line 1078 of file Number.cc.

10.217.2.7 bool num::Number::cne (const Number & *n*) const

Definition at line 1035 of file Number.cc.

10.217.2.8 void num::Number::fromstr (str *s1*)

Definition at line 332 of file Number.cc.

10.217.2.9 intv num::Number::get_i ( ) const [inline]

Definition at line 105 of file Number.h.

10.217.2.10 numtypes num::Number::get_numtype ( ) [inline]

Definition at line 107 of file Number.h.

10.217.2.11 cprat num::Number::get_q ( ) const

Definition at line 220 of file Number.cc.

10.217.2.12 mpq_class num::Number::get_q_class ( ) const [inline]

Definition at line 102 of file Number.h.

10.217.2.13 double num::Number::get_r ( ) const [inline]

Definition at line 106 of file Number.h.

10.217.2.14 str num::Number::get_s ( ) const

Definition at line 227 of file Number.cc.

10.217.2.15 bool num::Number::operator!= (const Number & *n*) const [inline]

Definition at line 236 of file Number.h.

10.217.2.16 Number num::Number::operator∗ (const Number & *n*) const

Definition at line 422 of file Number.cc.

10.217.2.17 const Number & num::Number::operator∗= (const Number & *n*)

Definition at line 474 of file Number.cc.

10.217.2.18 Number num::Number::operator+ (const Number & *n*) const

Definition at line 410 of file Number.cc.

10.217.2.19 const Number & num::Number::operator+= (const Number & *n*)

Definition at line 434 of file Number.cc.

10.217.2.20 Number num::Number::operator- (const Number & *n*) const

Definition at line 416 of file Number.cc.

10.217.2.21 const Number & num::Number::operator-= (const Number & *n*)

Definition at line 454 of file Number.cc.

10.217.2.22 Number num::Number::operator/ (const Number & *n*) const

Definition at line 428 of file Number.cc.

10.217.2.23 const Number & num::Number::operator/= (const Number & *n*)

Definition at line 494 of file Number.cc.

10.217.2.24 bool num::Number::operator< (const Number & *n*) const [inline]

Definition at line 244 of file Number.h.

10.217.2.25 bool num::Number::operator<= (const Number & *n*) const [inline]

Definition at line 242 of file Number.h.

10.217.2.26 const Number & num::Number::operator= (int *n*)

Definition at line 162 of file Number.cc.

10.217.2.27 const Number & num::Number::operator= (unsigned *n*)

Definition at line 170 of file Number.cc.

10.217.2.28 const Number & num::Number::operator= (double *d*)

Definition at line 178 of file Number.cc.

10.217.2.29 const Number & num::Number::operator= (cprat *q*)

Definition at line 185 of file Number.cc.

10.217.2.30 const Number & num::Number::operator= (const mpq_class & *q*)

Definition at line 194 of file Number.cc.

10.217.2.31 const Number & num::Number::operator= (const intv & *i*)

Definition at line 199 of file Number.cc.

10.217.2.32 const Number & num::Number::operator= (const str & *s*)

Definition at line 206 of file Number.cc.

10.217.2.33 const Number & num::Number::operator= (const Number & *n*)

Definition at line 211 of file Number.cc.

10.217.2.34 bool num::Number::operator== (const Number & *n*) const [inline]

Definition at line 234 of file Number.h.

10.217.2.35 bool num::Number::operator> (const Number & *n*) const [inline]

Definition at line 240 of file Number.h.

10.217.2.36 bool num::Number::operator>= (const Number & *n*) const [inline]

Definition at line 238 of file Number.h.

10.217.2.37 Number num::Number::operator^ (const Number & *n*) const [inline]

Definition at line 194 of file Number.h.

10.217.2.38 const Number& num::Number::operator^= (const Number & *n*) [inline]

Definition at line 196 of file Number.h.

10.217.2.39 bool num::Number::peq (const Number & *n*) const

Definition at line 1088 of file Number.cc.

10.217.2.40 bool num::Number::pge (const Number & *n*) const

Definition at line 1114 of file Number.cc.

10.217.2.41 bool num::Number::pgt (const Number & *n*) const

Definition at line 1124 of file Number.cc.

10.217.2.42 bool num::Number::ple (const Number & *n*) const

Definition at line 1134 of file Number.cc.

10.217.2.43 bool num::Number::plt (const Number & *n*) const

Definition at line 1144 of file Number.cc.

10.217.2.44 bool num::Number::pne (const Number & *n*) const

Definition at line 1101 of file Number.cc.

10.217.2.45 bool num::Number::seq (const Number & *n*) const

Definition at line 956 of file Number.cc.

10.217.2.46 bool num::Number::sge (const Number & *n*) const

Definition at line 982 of file Number.cc.

10.217.2.47 bool num::Number::sgt (const Number & *n*) const

Definition at line 992 of file Number.cc.

10.217.2.48 bool num::Number::sle (const Number & *n*) const

Definition at line 1002 of file Number.cc.

10.217.2.49 bool num::Number::slt (const Number & *n*) const

Definition at line 1012 of file Number.cc.

10.217.2.50 bool num::Number::sne (const Number & *n*) const

Definition at line 969 of file Number.cc.

10.217.2.51 str num::Number::tooutputstr ( ) const

Definition at line 286 of file Number.cc.

10.217.2.52 str num::Number::tostr ( ) const

Definition at line 263 of file Number.cc.

10.217.3 Friends And Related Function Documentation

10.217.3.1 Number abs (const Number & *x*) [friend]

Definition at line 526 of file Number.cc.

10.217.3.2 Number acos (const Number & *x*) [friend]

Definition at line 515 of file Number.cc.

10.217.3.3 Number acosh (const Number & *x*) [friend]

Definition at line 546 of file Number.cc.

10.217.3.4 Number acot (const Number & *x*) [friend]

Definition at line 557 of file Number.cc.

10.217.3.5 Number acoth (const Number & *x*) [friend]

Definition at line 568 of file Number.cc.

10.217.3.6 Number asin (const Number & *x*) [friend]

Definition at line 579 of file Number.cc.

10.217.3.7 Number asinh (const Number & *x*) [friend]

Definition at line 590 of file Number.cc.

10.217.3.8 Number atan (const Number & *x*) [friend]

Definition at line 601 of file Number.cc.

10.217.3.9 Number atanh (const Number & *x*) [friend]

Definition at line 612 of file Number.cc.

10.217.3.10 Number cos (const Number & *x*) [friend]

Definition at line 623 of file Number.cc.

10.217.3.11 Number cosh (const Number & *x*) [friend]

Definition at line 634 of file Number.cc.

10.217.3.12 Number cot (const Number & *x*) [friend]

Definition at line 645 of file Number.cc.

10.217.3.13 Number coth (const Number & *x*) [friend]

Definition at line 656 of file Number.cc.

10.217.3.14 Number exp (const Number & *x*) [friend]

Definition at line 667 of file Number.cc.

10.217.3.15 Number exp10 (const Number & *x*) [friend]

Definition at line 678 of file Number.cc.

10.217.3.16 Number exp2 (const Number & *x*) [friend]

Definition at line 724 of file Number.cc.

10.217.3.17 Number expm1 (const Number & *x*) [friend]

Definition at line 770 of file Number.cc.

10.217.3.18 Number log (const Number & *x*) [friend]

Definition at line 782 of file Number.cc.

10.217.3.19 Number log10 (const Number & *x*) [friend]

Definition at line 793 of file Number.cc.

10.217.3.20 Number log1p (const Number & *x*) [friend]

Definition at line 805 of file Number.cc.

10.217.3.21 Number log2 (const Number & *x*) [friend]

Definition at line 817 of file Number.cc.

10.217.3.22 std::ostream& operator<< (std::ostream & *os*, const Number & *n*) [friend]

Definition at line 313 of file Number.cc.

10.217.3.23 std::istream& operator>> (std::istream & *is*, Number & *n*) [friend]

Definition at line 403 of file Number.cc.

10.217.3.24 Number pow (const Number & *x*, const Number & *y*) [friend]

Definition at line 862 of file Number.cc.

10.217.3.25 Number power (const Number & *x*, int *n*) [friend]

Definition at line 829 of file Number.cc.

10.217.3.26 Number sin (const Number & *x*) [friend]

Definition at line 878 of file Number.cc.

10.217.3.27 Number sinh (const Number & *x*) [friend]

Definition at line 889 of file Number.cc.

10.217.3.28 Number sqr (const Number & *x*) [friend]

Definition at line 900 of file Number.cc.

10.217.3.29 Number sqrt (const Number & *x*) [friend]

Definition at line 912 of file Number.cc.

10.217.3.30 Number tan (const Number & *x*) [friend]

Definition at line 934 of file Number.cc.

10.217.3.31 Number tanh (const Number & *x*) [friend]

Definition at line 945 of file Number.cc.

The documentation for this class was generated from the following files:

- Number.h
- Number.cc

10.218 coco::coco::num::number_exception Class Reference

Public Member Functions

- number_exception (number_exception_type a, char const ∗m)
- number_exception (number_exception_type a, const str &m)
- virtual ∼number_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual number_exception_type type () const throw ()
- virtual const char ∗ type_str () throw ()
- virtual str message () const throw ()

10.218.1 Constructor & Destructor Documentation

10.218.1.1 coco::coco::num::number_exception::number_exception (number_exception_type *a*, char const ∗ *m*) [inline]

Definition at line 259 of file search_graph.cc.

10.218.1.2 coco::coco::num::number_exception::number_exception (number_exception_type *a*, const str & *m*) [inline]

Definition at line 261 of file search_graph.cc.

10.218.1.3 virtual coco::coco::num::number_exception::∼number_exception ( ) throw () [inline, virtual]

Definition at line 263 of file search_graph.cc.

10.218.2 Member Function Documentation

10.218.2.1 virtual str coco::coco::num::number_exception::message ( ) const throw () [inline, virtual]

Definition at line 275 of file search_graph.cc.

10.218.2.2 virtual number_exception_type coco::coco::num::number_exception::type ( ) const throw () [inline, virtual]

Definition at line 269 of file search_graph.cc.

10.218.2.3 virtual const char* coco::coco::num::number_exception::type_str ( ) throw () [virtual]

10.218.2.4 virtual char const* coco::coco::num::number_exception::what ( ) const throw () [inline, virtual]

Definition at line 265 of file search_graph.cc.

The documentation for this class was generated from the following file:

- Number.h

10.219 coco::num::number_exception Class Reference

`#include <expression.h>`

Public Member Functions

- number_exception (number_exception_type a, char const *m)
- number_exception (number_exception_type a, const str &m)
- virtual ~number_exception () throw ()
- virtual char const * what () const throw ()
- virtual number_exception_type type () const throw ()
- virtual const char * type_str () throw ()
- virtual str message () const throw ()
- number_exception (number_exception_type a, char const *m)
- number_exception (number_exception_type a, const str &m)
- virtual ~number_exception () throw ()
- virtual char const * what () const throw ()
- virtual number_exception_type type () const throw ()
- virtual const char * type_str () throw ()
- virtual str message () const throw ()
- number_exception (number_exception_type a, char const *m)
- number_exception (number_exception_type a, const str &m)
- virtual ~number_exception () throw ()
- virtual char const * what () const throw ()
- virtual number_exception_type type () const throw ()
- virtual const char * type_str () throw ()

- virtual str message () const throw ()
- number_exception (number_exception_type a, char const *m)
- number_exception (number_exception_type a, const str &m)
- virtual ~number_exception () throw ()
- virtual char const * what () const throw ()
- virtual number_exception_type type () const throw ()
- virtual const char * type_str () throw ()
- virtual str message () const throw ()

10.219.1 Detailed Description

Definition at line 253 of file search_graph.cc.

10.219.2 Constructor & Destructor Documentation

10.219.2.1 coco::num::number_exception::number_exception (number_exception_type *a*, char const * *m*) [inline]

Definition at line 259 of file expression.h.

10.219.2.2 coco::num::number_exception::number_exception (number_exception_type *a*, const str & *m*) [inline]

Definition at line 261 of file expression.h.

10.219.2.3 virtual coco::num::number_exception::~number_exception ( ) throw () [inline, virtual]

Definition at line 263 of file expression.h.

10.219.2.4 coco::num::number_exception::number_exception (number_exception_type *a*, char const * *m*) [inline]

Definition at line 259 of file search_graph.cc.

10.219.2.5 coco::num::number_exception::number_exception (number_exception_type *a*, const str & *m*) [inline]

Definition at line 261 of file search_graph.cc.

10.219.2.6 virtual coco::num::number_exception::~number_exception ( ) throw () [inline, virtual]

Definition at line 263 of file search_graph.cc.

10.219.2.7 coco::num::number_exception::number_exception (number_exception_type *a*, char const * *m*) [inline]

Definition at line 259 of file search_graph.cc.

10.219.2.8 coco::num::number_exception::number_exception (number_exception_type *a*, const str & *m*) [inline]

Definition at line 261 of file search_graph.cc.

10.219.2.9 virtual coco::num::number_exception::~number_exception ( ) throw () [inline, virtual]

Definition at line 263 of file search_graph.cc.

10.219.2.10 coco::num::number_exception::number_exception (number_exception_type *a*, char const * *m*) [inline]

Definition at line 259 of file search_graph.cc.

10.219.2.11 coco::num::number_exception::number_exception (number_exception_type *a*, const str & *m*) [inline]

Definition at line 261 of file search_graph.cc.

10.219.2.12 virtual coco::num::number_exception::~number_exception ( ) throw () [inline, virtual]

Definition at line 263 of file search_graph.cc.

10.219.3 Member Function Documentation

10.219.3.1 virtual str coco::num::number_exception::message ( ) const throw () [inline, virtual]

Definition at line 275 of file expression.h.

10.219.3.2 virtual str coco::num::number_exception::message ( ) const throw () [inline, virtual]

Definition at line 275 of file search_graph.cc.

10.219.3.3 virtual str coco::num::number_exception::message ( ) const throw () [inline, virtual]

Definition at line 275 of file search_graph.cc.

10.219.3.4 virtual str coco::num::number_exception::message ( ) const throw () [inline, virtual]

Definition at line 275 of file search_graph.cc.

10.219.3.5 virtual number_exception_type coco::num::number_exception::type ( ) const throw () [inline, virtual]

Definition at line 269 of file expression.h.

10.219.3.6 virtual number_exception_type coco::num::number_exception::type ( ) const throw () [inline, virtual]

Definition at line 269 of file search_graph.cc.

10.219.3.7 virtual number_exception_type coco::num::number_exception::type ( ) const throw () [inline, virtual]

Definition at line 269 of file search_graph.cc.

10.219.3.8 virtual number_exception_type coco::num::number_exception::type ( ) const throw () [inline, virtual]

Definition at line 269 of file search_graph.cc.

10.219.3.9 virtual const char* coco::num::number_exception::type_str ( ) throw () [virtual]

10.219.3.10 virtual const char* coco::num::number_exception::type_str ( ) throw () [virtual]

10.219.3.11 virtual const char* coco::num::number_exception::type_str ( ) throw () [virtual]

10.219.3.12 virtual const char* coco::num::number_exception::type_str ( ) throw () [virtual]

10.219.3.13 virtual char const* coco::num::number_exception::what ( ) const throw () [inline, virtual]

Definition at line 265 of file expression.h.

10.219.3.14 virtual char const* coco::num::number_exception::what ( ) const throw () [inline, virtual]

Definition at line 265 of file search_graph.cc.

10.219.3.15 virtual char const* coco::num::number_exception::what ( ) const throw () [inline, virtual]

Definition at line 265 of file search_graph.cc.

10.219.3.16 virtual char const* coco::num::number_exception::what ( ) const throw () [inline, virtual]

Definition at line 265 of file search_graph.cc.

The documentation for this class was generated from the following file:

- Number.h

10.220 num::number_exception Class Reference

```
#include <Number.h>
```

Public Member Functions

- number_exception (number_exception_type a, char const ∗m)
- number_exception (number_exception_type a, const str &m)
- virtual ∼number_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual number_exception_type type () const throw ()
- virtual const char ∗ type_str () throw ()
- virtual str message () const throw ()

10.220.1 Constructor & Destructor Documentation

10.220.1.1 num::number_exception::number_exception (number_exception_type *a*, char const ∗ *m*) [inline]

Definition at line 259 of file Number.h.

10.220.1.2 num::number_exception::number_exception (number_exception_type *a*, const str & *m*) [inline]

Definition at line 261 of file Number.h.

10.220.1.3 virtual num::number_exception::∼number_exception ( ) throw () [inline, virtual]

Definition at line 263 of file Number.h.

10.220.2 Member Function Documentation

10.220.2.1 virtual str num::number_exception::message ( ) const throw () [inline, virtual]

Definition at line 275 of file Number.h.

10.220.2.2 virtual number_exception_type num::number_exception::type ( ) const throw () [inline, virtual]

Definition at line 269 of file Number.h.

10.220.2.3 const char ∗ num::number_exception::type_str ( ) throw () [virtual]

Definition at line 1154 of file Number.cc.

10.220.2.4 virtual char const∗ num::number_exception::what ( ) const throw () [inline, virtual]

Definition at line 265 of file Number.h.

The documentation for this class was generated from the following files:

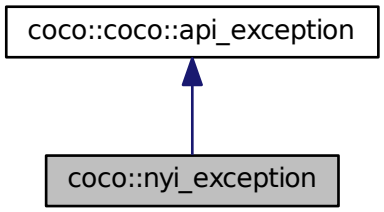
- Number.h
- Number.cc

10.221 coco::nyi_exception Class Reference

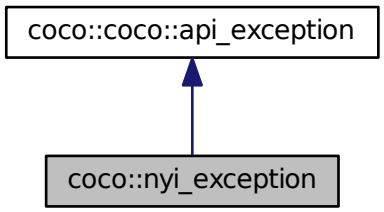
Not Yet Implemented exception class.

`#include <api_exception.h>`

Inheritance diagram for coco::nyi_exception:



Collaboration diagram for coco::nyi_exception:



Public Member Functions

- nyi_exception (char const ∗m)
- nyi_exception (const std::string &m)
- virtual ∼nyi_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual char const ∗ what () const throw ()
- virtual char const ∗ what () const throw ()
- virtual char const ∗ what () const throw ()
- virtual char const ∗ what () const throw ()

- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()

10.221.1 Detailed Description

This is a subclass of api_exception, which is reserved for exceptions, which are thrown because something that should be working is not yet implemented.

10.221.2 Constructor & Destructor Documentation

10.221.2.1 coco::nyi_exception::nyi_exception (char const * *m*) [inline]

Constructor, setting the message to `m`

Definition at line 121 of file api_exception.h.

10.221.2.2 coco::nyi_exception::nyi_exception (const std::string & *m*) [inline]

Constructor, setting the message to `m`

Definition at line 123 of file api_exception.h.

10.221.2.3 virtual coco::nyi_exception::~nyi_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 126 of file api_exception.h.

10.221.3 Member Function Documentation

10.221.3.1 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file expression.h.

10.221.3.2 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.221.3.3 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.221.3.4 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.221.3.5 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.221.3.6 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.221.3.7 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.221.3.8 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file expression.h.

10.221.3.9 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.221.3.10 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.221.3.11 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.221.3.12 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.221.3.13 const char * coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

Definition at line 57 of file api_exception.cc.

10.221.3.14 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.221.3.15 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.221.3.16 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.221.3.17 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.221.3.18 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file expression.h.

10.221.3.19 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.221.3.20 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

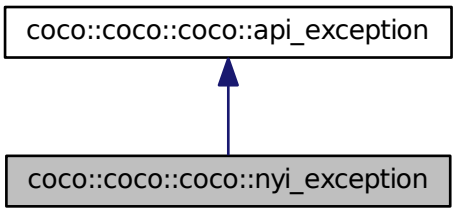
The documentation for this class was generated from the following file:

- api_exception.h

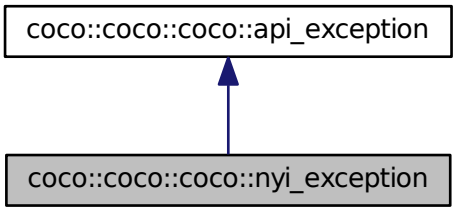
10.222 coco::coco::coco::nyi_exception Class Reference

Not Yet Implemented exception class.

Inheritance diagram for coco::coco::coco::nyi_exception:



Collaboration diagram for coco::coco::coco::nyi_exception:



Public Member Functions

- nyi_exception (char const ∗m)
- nyi_exception (const std::string &m)
- virtual ∼nyi_exception () throw ()
- virtual char const ∗ what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char ∗ type_str () const throw ()
- virtual std::string message () const throw ()

10.222.1 Detailed Description

This is a subclass of api_exception, which is reserved for exceptions, which are thrown because something that should be working is not yet implemented.

10.222.2 Constructor & Destructor Documentation

10.222.2.1 coco::coco::coco::nyi_exception::nyi_exception (char const ∗ *m*) [inline]

Constructor, setting the message to `m`

Definition at line 121 of file search_graph.cc.

10.222.2.2 coco::coco::coco::nyi_exception::nyi_exception (const std::string & *m*) [inline]

Constructor, setting the message to `m`

Definition at line 123 of file search_graph.cc.

10.222.2.3 virtual coco::coco::coco::nyi_exception::~nyi_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 126 of file search_graph.cc.

10.222.3 Member Function Documentation

10.222.3.1 virtual std::string coco::coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.222.3.2 virtual api_exception_type coco::coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.222.3.3 virtual const char* coco::coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.222.3.4 virtual char const* coco::coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following file:

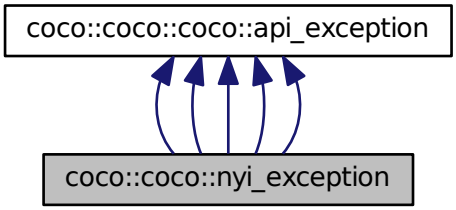
- api_exception.h

10.223 coco::coco::nyi_exception Class Reference

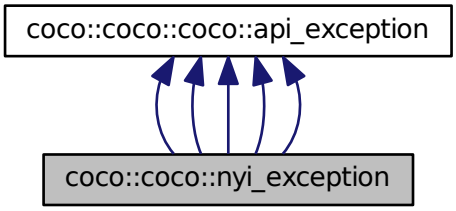
Not Yet Implemented exception class.

`#include <expression.h>`

Inheritance diagram for coco::coco::nyi_exception:



Collaboration diagram for coco::coco::nyi_exception:



Public Member Functions

- nyi_exception (char const ∗m)
- nyi_exception (const std::string &m)
- virtual ∼nyi_exception () throw ()
- nyi_exception (char const ∗m)
- nyi_exception (const std::string &m)
- virtual ∼nyi_exception () throw ()
- nyi_exception (char const ∗m)
- nyi_exception (const std::string &m)
- virtual ∼nyi_exception () throw ()
- nyi_exception (char const ∗m)
- nyi_exception (const std::string &m)
- virtual ∼nyi_exception () throw ()
- nyi_exception (char const ∗m)

- nyi_exception (const std::string &m)
- virtual ~nyi_exception () throw ()
- virtual char const * what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.223.1 Detailed Description

This is a subclass of api_exception, which is reserved for exceptions, which are thrown because something that should be working is not yet implemented.

Definition at line 117 of file search_graph.cc.

10.223.2 Constructor & Destructor Documentation

10.223.2.1 coco::coco::nyi_exception::nyi_exception (char const * *m*) [inline]

Constructor, setting the message to `m`

Definition at line 121 of file expression.h.

10.223.2.2 coco::coco::nyi_exception::nyi_exception (const std::string & *m*) [inline]

Constructor, setting the message to `m`

Definition at line 123 of file expression.h.

10.223.2.3 virtual coco::coco::nyi_exception::~nyi_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 126 of file expression.h.

10.223.2.4 coco::coco::nyi_exception::nyi_exception (char const * *m*) [inline]

Constructor, setting the message to `m`

Definition at line 121 of file search_graph.cc.

10.223.2.5 coco::coco::nyi_exception::nyi_exception (const std::string & *m*) [inline]

Constructor, setting the message to `m`

Definition at line 123 of file search_graph.cc.

10.223.2.6 virtual coco::coco::nyi_exception::~nyi_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 126 of file search_graph.cc.

10.223.2.7 coco::coco::nyi_exception::nyi_exception (char const * *m*) [inline]

Constructor, setting the message to m

Definition at line 121 of file search_graph.cc.

10.223.2.8 coco::coco::nyi_exception::nyi_exception (const std::string & *m*) [inline]

Constructor, setting the message to m

Definition at line 123 of file search_graph.cc.

10.223.2.9 virtual coco::coco::nyi_exception::~nyi_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 126 of file search_graph.cc.

10.223.2.10 coco::coco::nyi_exception::nyi_exception (char const * *m*) [inline]

Constructor, setting the message to m

Definition at line 121 of file search_graph.cc.

10.223.2.11 coco::coco::nyi_exception::nyi_exception (const std::string & *m*) [inline]

Constructor, setting the message to m

Definition at line 123 of file search_graph.cc.

10.223.2.12 virtual coco::coco::nyi_exception::~nyi_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 126 of file search_graph.cc.

10.223.2.13 coco::coco::nyi_exception::nyi_exception (char const * *m*) [inline]

Constructor, setting the message to m

Definition at line 121 of file search_graph.cc.

10.223.2.14 coco::coco::nyi_exception::nyi_exception (const std::string & *m*) [inline]

Constructor, setting the message to m

Definition at line 123 of file search_graph.cc.

10.223.2.15 virtual coco::coco::nyi_exception::~nyi_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 126 of file search_graph.cc.

10.223.3 Member Function Documentation

10.223.3.1 virtual std::string coco::coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.223.3.2 virtual api_exception_type coco::coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.223.3.3 virtual const char* coco::coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.223.3.4 virtual char const* coco::coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following file:

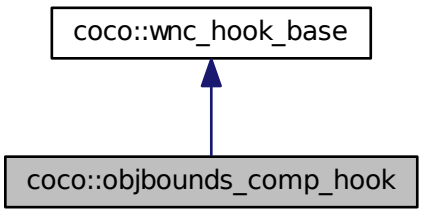
- api_exception.h

10.224 coco::objbounds_comp_hook Class Reference

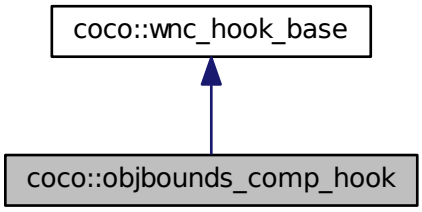
The objective-bounds computation hook (work node computation hook)

`#include <objbounds_hook.h>`

Inheritance diagram for coco::objbounds_comp_hook:



Collaboration diagram for coco::objbounds_comp_hook:



Public Member Functions

- objbounds_comp_hook (bool _ub=false, bool _cd=false)
- virtual ~objbounds_comp_hook ()
- objbounds_comp_hook * new_copy () const
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *add_ds=NULL, std::list< certificate > *add_cs=NULL) const
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)
- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.224.1 Detailed Description

This class is a work node computation hook (

See also

work_node_comp_hook), which calculates lower and upper bounds on the objective function on the work node and keeps them in the columns “obj lowbound” and “obj upbound”. The diameter of the interval is kept in the column “obj diameter”

10.224.2 Constructor & Destructor Documentation

10.224.2.1 coco::objbounds_comp_hook::objbounds_comp_hook (bool *_ub* = false, bool *_cd* = false) [inline]

Standard Constructor

Definition at line 54 of file objbounds_hook.h.

10.224.2.2 virtual coco::objbounds_comp_hook::~objbounds_comp_hook ( ) [inline, virtual]

Standard Destructor

Definition at line 60 of file objbounds_hook.h.

10.224.3 Member Function Documentation

10.224.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected, inherited]

Remove all columns, whose identifiers are stored in ___managed_cols from table stable.

Definition at line 34 of file comp_hook.cc.

10.224.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 238 of file comp_hook.h.

10.224.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 135 of file comp_hook.h.

10.224.3.4 bool coco::objbounds_comp_hook::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon unregistering this hook, destroy the columns “obj lowbound” “obj diameter”, and “obj upbound” in the “search info” table.

Reimplemented from coco::wnc_hook_base.

Definition at line 130 of file objbounds_hook.h.

10.224.3.5 bool coco::objbounds_comp_hook::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the columns “obj lowbound” “obj diameter”, and “obj upbound” in the “search info” table.

Reimplemented from coco::wnc_hook_base.

Definition at line 120 of file objbounds_hook.h.

10.224.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline, inherited]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.224.3.7 objbounds_comp_hook* coco::objbounds_comp_hook::new_copy ( ) const [inline, virtual]

Clone Operation

Implements coco::wnc_hook_base.

Definition at line 63 of file objbounds_hook.h.

10.224.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_id of work_node `wn`.

Definition at line 43 of file comp_hook.cc.

10.224.3.9 void coco::objbounds_comp_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *add_ds* = `NULL`, std::list< certificate > * *add_cs* = `NULL`) const [inline, virtual]

The evaluation operator of this computation hook. It stores the lower and upper bounds on the objective function on the work_node `wn` in dbt_row `dbr`.

Implements coco::wnc_hook_base.

Definition at line 69 of file objbounds_hook.h.

10.224.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_relation of work_node `wn`.

Definition at line 46 of file comp_hook.cc.

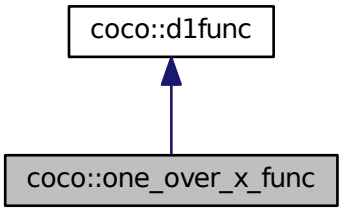
The documentation for this class was generated from the following file:

- objbounds_hook.h

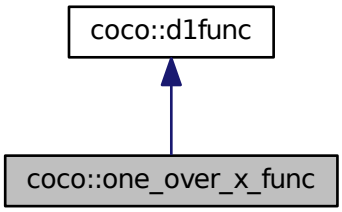
10.225 coco::one_over_x_func Class Reference

```
#include <d1testf.h>
```

Inheritance diagram for coco::one_over_x_func:



Collaboration diagram for coco::one_over_x_func:



Public Types

- enum d1func_point_t { d1fp_unknown = -2, d1fp_regular = -1, d1fp_extremum = 0, d1fp_evenpoleup = 1, d1fp_evenpoledn = 2, d1fp_oddpoleup = 3, d1fp_oddpoledn = 4, d1fp_jumpldef = 5, d1fp_jumpludef = 6, d1fp_point = 7, d1fp_undefined = 8, d1fp_wild = 9, d1fp_jumprdef = 10, d1fp_jumprudef = 11 }
- enum d1solve_t { d1solve_0, d1solve_1, d1solve_2 }
- typedef std::triple< interval, interval, d1func_point_t > point_info
- typedef std::vector< point_info > point_list

Public Member Functions

- one_over_x_func ()
- virtual interval imperfect_eval (const interval &x, unsigned int k) const =0
- virtual double eval (double x, unsigned int k) const =0

- virtual diffNumber eval (const diffNumber &x, unsigned int k) const
- virtual hessNumber eval (const hessNumber &x, unsigned int k) const
- virtual interval eval (const interval &x, unsigned int k) const
- virtual diffI eval (const diffI &x, unsigned int k) const
- virtual ihessNumber eval (const ihessNumber &x, unsigned int k) const
- virtual Islope eval (const Islope &x, unsigned int k) const
- virtual interval_set eval (const interval_set &x, unsigned int k) const
- virtual const char * func_name () const =0
- virtual uint64_t thash ()=0
- virtual uint64_t chash ()=0
- virtual std::string dag_out () const
- virtual std::ostream & print_add (std::ostream &o) const
- virtual std::string print_params () const
- virtual interval eval_slp (const interval &z, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &y, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &x, unsigned int k) const
- virtual void internal_eval_d1fp (const diffI &x, diffI &res, const std::vector< d1func_point_t > &ptin, std::vector< d1func_point_t > &ptres, unsigned int order) const
- virtual interval_set intersect_inv (const interval &x, const interval &r, unsigned int k) const
- virtual interval_set intersect_inv (const interval_set &x, const interval_set &r, unsigned int k) const
- virtual std::pair< double, double > evald (double x, unsigned int k) const
- virtual std::triple< double, double, double > evalt (double x, unsigned int k) const
- virtual void evalf (double x, double *r, unsigned int k) const
- virtual convex_info convexity (const interval &i, unsigned int k) const
- virtual d1func_monotonicity_t monotonicity (const interval &i, unsigned int k) const
- virtual int differentiability (const interval &i, unsigned int k) const
- virtual int degree_update (int in_degree) const
- virtual bool order_is_supported (unsigned int k) const
- virtual unsigned int supported_eval_order () const
- virtual const func_info & ff (unsigned int k) const
- virtual bool operator== (const d1func &b) const
- virtual bool operator!= (const d1func &b) const
- const char * toString (const d1func::d1func_point_t &t) const

Public Attributes

- volatile int MAXLEN

Protected Member Functions

- virtual interval imperfect_eval_f0 (const interval &x) const
- virtual interval imperfect_eval_f1 (const interval &x) const
- virtual interval imperfect_eval_f2 (const interval &x) const
- virtual void raise_order (unsigned int neworder)

Protected Attributes

- double dod_left
- double dod_right
- int basic_diff
- std::vector< func_info > _f

10.225.1 Member Typedef Documentation

10.225.1.1 typedef std::triple<interval, interval, d1func_point_t> coco::d1func::point_info [inherited]

The information on one point .first is the point, .second the value

Definition at line 110 of file d1func.h.

10.225.1.2 typedef std::vector<point_info> coco::d1func::point_list [inherited]

a list of points for function value, first and second derivatives

Definition at line 113 of file d1func.h.

10.225.2 Member Enumeration Documentation

10.225.2.1 enum coco::d1func::d1func_point_t [inherited]

The possible classes for special points in the lists: d1fp_regular: At this point the function is regular (for internal use only!) d1fp_extremum: The point is a local extremum d1fp_evenpoleup: At this point the function has a positive even pole d1fp_evenpoledn: At this point the function has a negative even pole d1fp_oddpoleup: At this point the function has an odd pole like 1/x at 0 d1fp_oddpoledn: At this point the function has an odd pole like -1/x at 0 d1fp_jumpldef: At this point the function has a jump (left side defined) d1fp_jumpludef: At this point the function has a jump (left side not defined) d1fp_point-: At this point the function has the specified value d1fp_undefined: In the given interval the function is undefined d1fp_wild: In the given interval the function has "wild" behaviour d1fp_jumprdef: At this point the function has a jump (right side defined) d1fp_jumprudef: At this point the function has a jump (right side not defined) d1fp_unknown: At this point the function has a unknown behaviour (for internal use only!)

Enumerator:

d1fp_unknown

d1fp_regular

d1fp_extremum

d1fp_evenpoleup

d1fp_evenpoledn

d1fp_oddpoleup

d1fp_oddpoledn

d1fp_jumpldef

d1fp_jumpludef

d1fp_point
d1fp_undefined
d1fp_wild
d1fp_jumprdef
d1fp_jumprudef

Definition at line 95 of file d1func.h.

10.225.2.2 enum coco::d1func::d1solve_t [inherited]

The possible functions for 1D Newton solving: d1solve_0: f(x) = c d1solve_1: f(x) - f(z) - f'(x)(x-z) = 0 d1solve_2: 1/(x-z)(f(x)-f(z))(1+(x-w)(x-z)) - (f(w)-f(z))/(w-z) - (x-w)/(x-z) f'(x) = 0

Enumerator:

d1solve_0
d1solve_1
d1solve_2

Definition at line 106 of file d1func.h.

10.225.3 Constructor & Destructor Documentation

10.225.3.1 coco::one_over_x_func::one_over_x_func ( ) [inline]

Definition at line 107 of file d1testf.h.

10.225.4 Member Function Documentation

10.225.4.1 virtual uint64_t coco::d1func::chash ( ) [pure virtual, inherited]

the const hash value for hash calculation in the DAG

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.225.4.2 convex_info coco::d1func::convexity (const interval & *i*, unsigned int *k*) const [virtual, inherited]

return convexity information on `i`.

Definition at line 2308 of file d1func.cc.

10.225.4.3 virtual std::string coco::d1func::dag_out ( ) const [inline, virtual, inherited]

the parameters written in DAG format (it must contain necessary ':'s)

Reimplemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

Definition at line 205 of file d1func.h.

10.225.4.4 virtual int coco::d1func::degree_update (int *in_degree*) const [inline, virtual, inherited]

return degree information for in-argument degree `in_degree`.

Definition at line 457 of file d1func.h.

10.225.4.5 int coco::d1func::differentiability (const interval & *x*, unsigned int *k*) const [virtual, inherited]

return differentiability information on `i`.

Reimplemented in coco::hsf_func, and coco::xexp_func.

Definition at line 2351 of file d1func.cc.

10.225.4.6 virtual double coco::d1func::eval (double *x*, unsigned int *k*) const [pure virtual, inherited]

the point evaluation

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.225.4.7 diffNumber coco::d1func::eval (const diffNumber & *x*, unsigned int *k*) const [virtual, inherited]

the diffNumber evaluation

Reimplemented in coco::dag_d1func.

Definition at line 349 of file d1func.cc.

10.225.4.8 hessNumber coco::d1func::eval (const hessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the hessNumber evaluation

Definition at line 334 of file d1func.cc.

10.225.4.9 interval coco::d1func::eval (const interval & *x_in*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 390 of file d1func.cc.

10.225.4.10 diffI coco::d1func::eval (const diffI & *x*, unsigned int *k*) const [virtual, inherited]

the interval diffNumber evaluation

Definition at line 715 of file d1func.cc.

10.225.4.11 ihessNumber coco::d1func::eval (const ihessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the interval hessNumber evaluation

the hessNumber evaluation

Definition at line 411 of file d1func.cc.

10.225.4.12 Islope coco::d1func::eval (const Islope & *x*, unsigned int *k*) const [virtual, inherited]

the interval slope evaluation

the interval diffNumber evaluation

Definition at line 429 of file d1func.cc.

10.225.4.13 interval_set coco::d1func::eval (const interval_set & *x*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 794 of file d1func.cc.

10.225.4.14 interval coco::d1func::eval_slp (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval first order slope evaluation at center z

Definition at line 1403 of file d1func.cc.

10.225.4.15 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *y*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at centers z and y

Definition at line 1984 of file d1func.cc.

10.225.4.16 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at double center z

Definition at line 1991 of file d1func.cc.

10.225.4.17 virtual std::pair<double,double> coco::d1func::evald (double *x*, unsigned int *k*) const [inline, virtual, inherited]

the point and derivative evaluation

Reimplemented in coco::xexp_func, and coco::dag_d1func.

Definition at line 437 of file d1func.h.

10.225.4.18 virtual void coco::d1func::evalf (double *x*, double * *r*, unsigned int *k*) const [inline, virtual, inherited]

the point, first to third derivative evaluation

Reimplemented in coco::xexp_func.

Definition at line 443 of file d1func.h.

10.225.4.19 virtual std::triple<double,double,double> coco::d1func::evalt (double *x*, unsigned int *k*) const [inline, virtual, inherited]

the point and first and second derivative evaluation

Reimplemented in coco::xexp_func, and coco::dag_d1func.

Definition at line 440 of file d1func.h.

10.225.4.20 virtual const func_info& coco::d1func::ff (unsigned int *k*) const [inline, virtual, inherited]

retrieve the k th special points list

Definition at line 469 of file d1func.h.

10.225.4.21 virtual const char* coco::d1func::func_name ( ) const [pure virtual, inherited]

the name of the d1func.

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.225.4.22 virtual interval coco::d1func::imperfect_eval (const interval & *x*, unsigned int *k*) const [pure virtual, inherited]

the imperfect evaluation of k th derivative.

Implemented in coco::xexp_func, coco::hsf_func, and coco::dag_d1func.

10.225.4.23 virtual interval coco::one_over_x_func::imperfect_eval_f0 (const interval & *x*) const [inline, protected, virtual]

the imperfect function evaluation

Definition at line 91 of file d1testf.h.

10.225.4.24 virtual interval coco::one_over_x_func::imperfect_eval_f1 (const interval & *x*) const [inline, protected, virtual]

the imperfect first derivative evaluation

Definition at line 96 of file d1testf.h.

10.225.4.25 virtual interval coco::one_over_x_func::imperfect_eval_f2 (const interval & *x*) const [inline, protected, virtual]

the imperfect second derivative evaluation

Definition at line 101 of file d1testf.h.

10.225.4.26 void coco::d1func::internal_eval_d1fp (const diffI & *x*, diffI & *res*, const std::vector< d1func_point_t > & *ptin*, std::vector< d1func_point_t > & *ptres*, unsigned int *k*) const [virtual, inherited]

perform internal evaluations to update eventual point lists of "higher" functions.

Definition at line 168 of file d1func.cc.

10.225.4.27 interval_set coco::d1func::intersect_inv (const interval & *x*, const interval & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1708 of file d1func.cc.

10.225.4.28 interval_set coco::d1func::intersect_inv (const interval_set & *x*, const interval_set & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1778 of file d1func.cc.

10.225.4.29 d1func_monotonicity_t coco::d1func::monotonicity (const interval & *x*, unsigned int *k*) const [virtual, inherited]

return monotonicity information on `i`.

Definition at line 2185 of file d1func.cc.

10.225.4.30 virtual bool coco::d1func::operator!= (const d1func & *b*) const [inline, virtual, inherited]

the other comparison operator

Definition at line 476 of file d1func.h.

10.225.4.31 virtual bool coco::d1func::operator== (const d1func & *b*) const [inline, virtual, inherited]

the comparison operator

Reimplemented in coco::hsf_func, coco::xexp_func, and coco::dag_d1func.

Definition at line 473 of file d1func.h.

10.225.4.32 virtual bool coco::d1func::order_is_supported (unsigned int *k*) const [inline, virtual, inherited]

check whether evaluation of this order is supported.

Definition at line 460 of file d1func.h.

10.225.4.33 virtual std::ostream& coco::d1func::print_add (std::ostream & *o*) const [inline, virtual, inherited]

additional info written in DAG format (it must contain dag lines)

Reimplemented in coco::dag_d1func.

Definition at line 208 of file d1func.h.

10.225.4.34 virtual std::string coco::d1func::print_params ( ) const [inline, virtual, inherited]

the parameters written in printable form

Reimplemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

Definition at line 211 of file d1func.h.

10.225.4.35 virtual void coco::d1func::raise_order (unsigned int *neworder*) [inline, protected, virtual, inherited]

raise the order of the supported function evaluation to `neworder`

Reimplemented in coco::hsf_func, and coco::xexp_func.

Definition at line 215 of file d1func.h.

10.225.4.36 virtual unsigned int coco::d1func::supported_eval_order ( ) const [inline, virtual, inherited]

return the maximal evaluation order supported.

Definition at line 466 of file d1func.h.

10.225.4.37 virtual uint64_t coco::d1func::thash ( ) [pure virtual, inherited]

the pure hash value for hash calculation in the DAG

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.225.4.38 const char * coco::d1func::toString (const d1func::d1func_point_t & *t*) const [inherited]

Definition at line 2415 of file d1func.cc.

10.225.5 Member Data Documentation

10.225.5.1 std::vector<func_info> coco::d1func::_f [protected, inherited]

the function info tables for function values and derivatives.

Definition at line 163 of file d1func.h.

10.225.5.2 int coco::d1func::basic_diff [protected, inherited]

basic differentiability

Definition at line 160 of file d1func.h.

10.225.5.3 double coco::d1func::dod_left [protected, inherited]

the left limit of the DOD

Definition at line 156 of file d1func.h.

10.225.5.4 double coco::d1func::dod_right [protected, inherited]

the right limit of the DOD

Definition at line 158 of file d1func.h.

10.225.5.5 volatile int coco::d1func::MAXLEN [inherited]

maximal lenght of the interval set returned

Definition at line 167 of file d1func.h.

The documentation for this class was generated from the following file:

- d1testf.h

10.226 coco::expression_node::parents_compare Class Reference

`#include <expression.hpp>`

Public Member Functions

- bool operator() (const expression_node &__x, const expression_node &__y) const

10.226.1 Member Function Documentation

10.226.1.1 bool coco::expression_node::parents_compare::operator() (const expression_node & __x, const expression_node & __y) const [inline]

This is the evaluation operator of this function class.

Definition at line 42 of file expression.hpp.

The documentation for this class was generated from the following file:

- expression.hpp

10.227 coco::expression_node::parents_compare_eq Class Reference

`#include <expression.hpp>`

Public Member Functions

- bool operator() (const expression_node &__x, const expression_node &__y) const

10.227.1 Member Function Documentation

10.227.1.1 bool coco::expression_node::parents_compare_eq::operator() (const expression_node & __x, const expression_node & __y) const [inline]

Definition at line 140 of file expression.hpp.

The documentation for this class was generated from the following file:

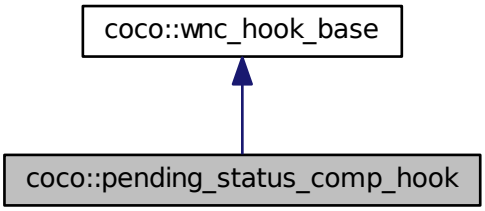
- expression.hpp

10.228 coco::pending_status_comp_hook Class Reference

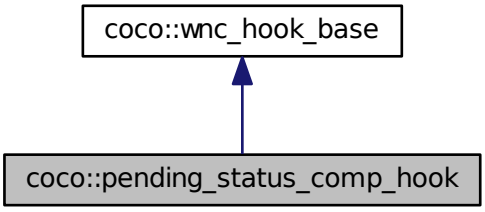
The pending status computation hook (work node computation hook)

`#include <pending_status_hook.h>`

Inheritance diagram for coco::pending_status_comp_hook:



Collaboration diagram for coco::pending_status_comp_hook:



Public Member Functions

- pending_status_comp_hook ()
- virtual ~pending_status_comp_hook ()
- pending_status_comp_hook * new_copy () const
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *a=NULL, std::list< certificate > *b=NULL) const
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)
- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.228.1 Detailed Description

This class is a work node computation hook (see work_node_comp_hook), which calculates the pending status of the work node from all constraints which are not satisfied (= pending) and keeps it in column "pending_status".

10.228.2 Constructor & Destructor Documentation

10.228.2.1 coco::pending_status_comp_hook::pending_status_comp_hook ( ) [inline]

Standard Constructor

Definition at line 48 of file pending_status_hook.h.

10.228.2.2 virtual coco::pending_status_comp_hook::~pending_status_comp_hook ( ) [inline, virtual]

Standard Destructor

Definition at line 51 of file pending_status_hook.h.

10.228.3 Member Function Documentation

10.228.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected, inherited]

Remove all columns, whose identifiers are stored in __managed_cols from table stable.

Definition at line 34 of file comp_hook.cc.

10.228.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 238 of file comp_hook.h.

10.228.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 135 of file comp_hook.h.

10.228.3.4 bool coco::pending_status_comp_hook::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon unregistering this hook, destroy the column "pending_status" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 106 of file pending_status_hook.h.

10.228.3.5 bool coco::pending_status_comp_hook::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the column "pending_status" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 101 of file pending_status_hook.h.

10.228.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline, inherited]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.228.3.7 pending_status_comp_hook* coco::pending_status_comp_hook::new_copy ( ) const [inline, virtual]

Clone Operation

Implements coco::wnc_hook_base.

Definition at line 54 of file pending_status_hook.h.

10.228.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_id of work_node wn.

Definition at line 43 of file comp_hook.cc.

10.228.3.9 void coco::pending_status_comp_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *a* = NULL, std::list< certificate > * *b* = NULL) const [inline, virtual]

The evaluation operator of this computation hook. It stores the pending status of the work_node `wn` in dbt_row `dbr`.

Implements coco::wnc_hook_base.

Definition at line 59 of file pending_status_hook.h.

10.228.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_relation of work_node `wn`.

Definition at line 46 of file comp_hook.cc.

The documentation for this class was generated from the following file:

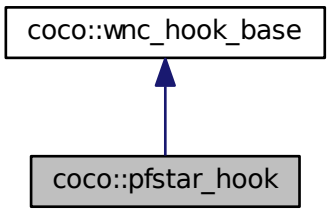
- pending_status_hook.h

10.229 coco::pfstar_hook Class Reference

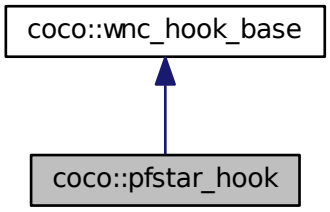
The pfstar computation hook (work node computation hook)

`#include <pfstar_hook.h>`

Inheritance diagram for coco::pfstar_hook:



Collaboration diagram for coco::pfstar_hook:



Public Member Functions

- pfstar_hook ()
- virtual ~pfstar_hook ()
- pfstar_hook * new_copy () const
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *add_ds=NULL, std::list< certificate > *add_cs=NULL) const
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)
- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.229.1 Detailed Description

This class is a work node computation hook (

See also

work_node_comp_hook), which adds a stored procedure for calculating the pfstar value of a box in column “pfstar”.

10.229.2 Constructor & Destructor Documentation

10.229.2.1 coco::pfstar_hook::pfstar_hook ( ) [inline]

Standard Constructor

Definition at line 47 of file pfstar_hook.h.

10.229.2.2 virtual coco::pfstar_hook::~pfstar_hook ( ) [inline, virtual]

Standard Destructor

Definition at line 52 of file pfstar_hook.h.

10.229.3 Member Function Documentation

10.229.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected, inherited]

Remove all columns, whose identifiers are stored in __managed_cols from table stable.

Definition at line 34 of file comp_hook.cc.

10.229.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in __managed_cols.

Definition at line 238 of file comp_hook.h.

10.229.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in __managed_cols.

Definition at line 135 of file comp_hook.h.

10.229.3.4 bool coco::pfstar_hook::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon unregistering this hook, destroy the columns "pfstar"

Reimplemented from coco::wnc_hook_base.

Definition at line 82 of file pfstar_hook.h.

10.229.3.5 bool coco::pfstar_hook::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the column "pfstar"

Reimplemented from coco::wnc_hook_base.

Definition at line 64 of file pfstar_hook.h.

10.229.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline, inherited]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.229.3.7 pfstar_hook* coco::pfstar_hook::new_copy ( ) const [inline, virtual]

Clone Operation

Implements coco::wnc_hook_base.

Definition at line 55 of file pfstar_hook.h.

10.229.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_id of work_node `wn`.

Definition at line 43 of file comp_hook.cc.

10.229.3.9 void coco::pfstar_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *add_ds* = `NULL`, std::list< certificate > * *add_cs* = `NULL`) const [inline, virtual]

The evaluation operator of this computation hook. It does nothing.

Implements coco::wnc_hook_base.

Definition at line 59 of file pfstar_hook.h.

10.229.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_relation of work_node `wn`.

Definition at line 46 of file comp_hook.cc.

The documentation for this class was generated from the following file:

- pfstar_hook.h

10.230 coco::coco::point_check_feasibility Class Reference

Stored procedure checking the feasibility of a point.

Public Types

- typedef work_node_context context
- typedef bool return_type

Public Member Functions

- point_check_feasibility (vdbl::colid _x, vdbl::colid _TL, vdbl::colid _f)
- point_check_feasibility (const point_check_feasibility &_i)
- virtual ~point_check_feasibility ()
- bool operator() () const
- bool def () const
- void setcontext (const context *c, const vdbl::row *r)

10.230.1 Detailed Description

This function class is used as a stored procedure (vdbl::method) for checking the feasibility of a point stored in the "point" table in the evaluation context of a given work_node.

10.230.2 Member Typedef Documentation

10.230.2.1 typedef work_node_context coco::coco::point_check_feasibility::context

The evaluation context type

Definition at line 48 of file search_graph.cc.

10.230.2.2 typedef bool coco::coco::point_check_feasibility::return_type

The return type of the operator(), i.e. the function class.

Definition at line 66 of file search_graph.cc.

10.230.3 Constructor & Destructor Documentation

10.230.3.1 coco::coco::point_check_feasibility::point_check_feasibility (vdbl::colid _x, vdbl::colid _TL, vdbl::colid _f) [inline]

Constructor setting the relevant column ids.

Definition at line 69 of file search_graph.cc.

10.230.3.2 coco::coco::point_check_feasibility::point_check_feasibility (const point_check_feasibility & _i) [inline]

Standard Copy Constructor

Definition at line 72 of file search_graph.cc.

10.230.3.3 virtual coco::coco::point_check_feasibility::~point_check_feasibility ( ) [inline, virtual]

Standard Destructor

Definition at line 76 of file search_graph.cc.

10.230.4 Member Function Documentation

10.230.4.1 bool coco::coco::point_check_feasibility::def ( ) const [inline]

Method, which computes the default value of the stored procedure

Definition at line 81 of file search_graph.cc.

10.230.4.2 bool coco::point_check_feasibility::operator() ( ) const

Evaluation operator, which computes the value of the stored procedure

Definition at line 36 of file dbtools.cc.

10.230.4.3 void coco::coco::point_check_feasibility::setcontext (const context * c, const vdbl::row * r) [inline]

This method initializes the evaluation context and the row, preparing for the evaluation.

Definition at line 84 of file search_graph.cc.

The documentation for this class was generated from the following files:

- dbtools.h
- dbtools.cc

10.231 coco::point_check_feasibility Class Reference

Stored procedure checking the feasibility of a point.

```
#include <dbtools.h>
```

Public Types

- typedef work_node_context context
- typedef bool return_type

Public Member Functions

- point_check_feasibility (vdbl::colid _x, vdbl::colid _TL, vdbl::colid _f)
- point_check_feasibility (const point_check_feasibility &_i)
- virtual ~point_check_feasibility ()
- bool operator() () const
- bool def () const
- void setcontext (const context *c, const vdbl::row *r)

10.231.1 Detailed Description

This function class is used as a stored procedure (vdbl::method) for checking the feasibility of a point stored in the "point" table in the evaluation context of a given work_node.

10.231.2 Member Typedef Documentation

10.231.2.1 typedef work_node_context coco::point_check_feasibility::context

The evaluation context type

Definition at line 48 of file dbtools.h.

10.231.2.2 typedef bool coco::point_check_feasibility::return_type

The return type of the operator(), i.e. the function class.

Definition at line 66 of file dbtools.h.

10.231.3 Constructor & Destructor Documentation

10.231.3.1 coco::point_check_feasibility::point_check_feasibility (vdbl::colid _x, vdbl::colid _TL, vdbl::colid _f) [inline]

Constructor setting the relevant column ids.

Definition at line 69 of file dbtools.h.

10.231.3.2 coco::point_check_feasibility::point_check_feasibility (const point_check_feasibility & _i) [inline]

Standard Copy Constructor

Definition at line 72 of file dbtools.h.

10.231.3.3 virtual coco::point_check_feasibility::~point_check_feasibility ( ) [inline, virtual]

Standard Destructor

Definition at line 76 of file dbtools.h.

10.231.4 Member Function Documentation

10.231.4.1 bool coco::point_check_feasibility::def ( ) const [inline]

Method, which computes the default value of the stored procedure

Definition at line 81 of file dbtools.h.

10.231.4.2 bool coco::point_check_feasibility::operator() ( ) const

Evaluation operator, which computes the value of the stored procedure

10.231.4.3 void coco::point_check_feasibility::setcontext (const context * c, const vdbl::row * r) [inline]

This method initializes the evaluation context and the row, preparing for the evaluation.

Definition at line 84 of file dbtools.h.

The documentation for this class was generated from the following file:

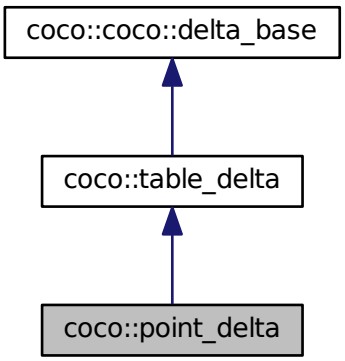
- dbtools.h

10.232 coco::point_delta Class Reference

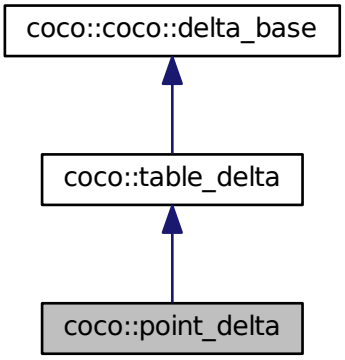
A delta class which adds new points to the search database.

`#include <point_delta.h>`

Inheritance diagram for coco::point_delta:



Collaboration diagram for coco::point_delta:



Public Types

- typedef std::pair< std::string, dbt_row > t_line
- typedef std::vector< t_line > t_ctr

Public Member Functions

- point_delta ()
- point_delta (const dbt_row &__p)
- point_delta (const point_delta &__d)
- point_delta * new_copy () const
- void destroy_copy (delta_base *__d) const
- void create_table (work_node &_x, vdbl::standard_table *&ptb, const std::string &__t) const
- bool apply (work_node &_x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const point_delta &_c) const
- bool operator!= (const point_delta &_c) const
- void add (const t_line &_tl)
- void add (const std::string &_tn, const dbt_row &_r)
- void add (const std::vector< t_line > &_tlv)
- void rm (const annotation &_tr)
- void rm (const std::vector< annotation > &_trv)
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- void convert (work_node &_x, delta_base *&_u)
- virtual void convert (work_node &_x, delta_base *&_d)
- bool operator== (const table_delta &_c) const

- bool operator!= (const table_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- const std::string & get_action () const
- const std::string & get_action () const
- virtual void unkeep ()
- virtual void unkeep ()
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Protected Attributes

- std::string _action

10.232.1 Detailed Description

A specialized delta class which adds new points, e.g. local optima to the search database. It is a subclass of table_delta and is, after storing the information in the search database, converted to an annotation_delta. Therefore, no `point_undelta` class is needed.

10.232.2 Member Typedef Documentation

10.232.2.1 typedef std::vector<t_line> coco::table_delta::t_ctr [inherited]

A variable of this type holds the new table entries.

Definition at line 49 of file table_delta.h.

10.232.2.2 typedef std::pair<std::string,dbt_row> coco::table_delta::t_line [inherited]

This type specifies one row in one table.

Definition at line 47 of file table_delta.h.

10.232.3 Constructor & Destructor Documentation

10.232.3.1 coco::point_delta::point_delta ( ) [inline]

Standard Constructor

Definition at line 52 of file point_delta.h.

10.232.3.2 coco::point_delta::point_delta (const dbt_row & __p) [inline]

Constructor: the point is described, including all additional information as dbt_row in the database format. It is stored in the table point of points. The columns are

Name	Type	R/O	Description	Default
x	vector<double>	required	the coordinates	
L_mult	vector<double>	optional	the Lagrange multipliers	empty
f	double	required	the objective value	
kappa	double	optional	the KJ multiplier	1
best	bool	optional	best point found	false
verified	bool	optional	verified point	false
feasible	bool	optional	feasible point	stored procedure
optimal	bool	optional	local optimal	false
global	bool	optional	global optimal	false
relaxation	bool	optional	from relaxation	false
class	unsigned int	optional	point class	0

Definition at line 72 of file point_delta.h.

10.232.3.3 coco::point_delta::point_delta (const point_delta & __d) [inline]

Standard Copy Constructor

Definition at line 77 of file point_delta.h.

10.232.4 Member Function Documentation

10.232.4.1 void coco::table_delta::add (const t_line & _tl) [inline, inherited]

This method adds one table row.

Definition at line 95 of file table_delta.h.

10.232.4.2 void coco::table_delta::add (const std::string & _tn, const dbt_row & _r) [inline, inherited]

This method adds one row _r to the table _tn.

Definition at line 97 of file table_delta.h.

10.232.4.3 void coco::table_delta::add (const std::vector< t_line > & _tlv) [inline, inherited]

This method adds a list of table rows.

Definition at line 100 of file table_delta.h.

10.232.4.4 bool coco::point_delta::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_did*, size_t & *delta_size*) const [virtual]

Apply the delta with delta_id _d to work node _x. This will never be used, because the point_delta is converted to annotation_delta before the apply is performed.

Reimplemented from coco::table_delta.

Definition at line 36 of file point_delta.cc.

10.232.4.5 virtual bool coco::coco::delta_base::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_di*, size_t & *delta_size*) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.232.4.6 bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.232.4.7 virtual bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.232.4.8 virtual bool coco::coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*, size_t & *delta_size*) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.232.4.9 void coco::table_delta::convert (work_node & *_x*, delta_base *& *_u*) [inherited]

This method converts the table delta to an annotation_delta after the information is stored in the search database. The conversion is based on work_node _x, and the generated annotation_delta is returned via _u.

Definition at line 43 of file table_delta.cc.

10.232.4.10 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.232.4.11 void coco::point_delta::create_table (work_node & _x, vdbl::standard_table *& ptb, const std::string & __t) const [virtual]

This method creates the point table in the search database for the work_node _x. A pointer ptb to the created table is set. The parameter ___t for the table name is ignored.

Reimplemented from coco::table_delta.

Definition at line 45 of file point_delta.cc.

10.232.4.12 void coco::point_delta::destroy_copy (delta_base * __d) const [inline, virtual]

Clone Destructor

Reimplemented from coco::table_delta.

Definition at line 88 of file point_delta.h.

10.232.4.13 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.232.4.14 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.232.4.15 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.232.4.16 delta coco::coco::delta_base::make_delta (const std::string & a) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.232.4.17 delta coco::coco::delta_base::make_delta (const std::string & a) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.232.4.18 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.232.4.19 point_delta* coco::point_delta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::table_delta.

Definition at line 85 of file point_delta.h.

10.232.4.20 bool coco::point_delta::operator!= (const delta_base & *_c*) const [inline]

Reimplemented from coco::table_delta.

Definition at line 106 of file point_delta.h.

10.232.4.21 bool coco::point_delta::operator!= (const point_delta & *_c*) const [inline]

Definition at line 113 of file point_delta.h.

10.232.4.22 bool coco::table_delta::operator!= (const table_delta & *_c*) const [inline, inherited]

Definition at line 144 of file table_delta.h.

10.232.4.23 bool coco::point_delta::operator== (const delta_base & *_c*) const [inline]

Comparison operators

Reimplemented from coco::table_delta.

Definition at line 103 of file point_delta.h.

10.232.4.24 bool coco::point_delta::operator== (const point_delta & *_c*) const [inline]

Comparison operators

Definition at line 111 of file point_delta.h.

10.232.4.25 bool coco::table_delta::operator== (const table_delta & *_c*) const [inline, inherited]

Comparison operators

Definition at line 143 of file table_delta.h.

10.232.4.26 void coco::table_delta::rm (const annotation & *_tr*) [inline, inherited]

This method adds the annotation `_tr` to the list of removed annotations.

Definition at line 104 of file table_delta.h.

10.232.4.27 void coco::table_delta::rm (const std::vector< annotation > & _trv) [inline, inherited]

This method adds the annotations in `_trv` to the list of removed annotations.

Definition at line 107 of file table_delta.h.

10.232.4.28 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.232.4.29 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.232.4.30 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.232.5 Member Data Documentation

10.232.5.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

The documentation for this class was generated from the following files:

- point_delta.h
- point_delta.cc

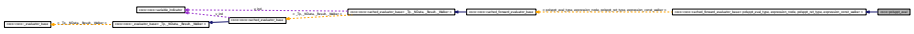
10.233 coco::polsppt_eval Class Reference

Forward function range evaluation.

`#include <dagd1func_pv.hpp>`

Inheritance diagram for coco::polsppt_eval:

Collaboration diagram for coco::polsppt_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- polsppt_eval (const std::vector< interval > &__x, const variable_indicator &__v, const model &_-_m, unsigned int _bs, std::vector< d1func::d1func_point_t > &_bt, const diffI &_bv, const polsppt-_interval_map &_pm, const polsppt_map_map &_pmm)
- polsppt_eval (const polsppt_eval &__v)
- ~polsppt_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_box (const std::vector< interval > &__x, const variable_indicator &__v)
- void new_base (unsigned int _b, std::vector< d1func::d1func_point_t > &_p, const diffI &_v)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const polsppt_ret_type &__rval)
- void d1fp_type_update (const interval &coeff, diffI &range2, std::vector< d1func::d1func_point_t > &op2, int expon)
- std::vector < d1func::d1func_point_t > & d1fp_sum_update (const diffI &range1, const std::vector< d1func::d1func_point_t > &op1, const interval &coeff, diffI range2, std::vector< d1func::d1func_-point_t > op2, int expo, std::vector< d1func::d1func_point_t > &ret)

- std::vector < d1func::d1func_point_t > & d1fp_prod_update (const diffI &range1, const std::vector< d1func::d1func_point_t > &op1, const interval &coeff, diffI range2, std::vector< d1func::d1func_point_t > op2, int expo, std::vector< d1func::d1func_point_t > &ret)
- int update (const expression_node &__data, const polsppt_ret_type &__rval)
- polsppt_ret_type calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.233.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural interval extension.

10.233.2 Member Typedef Documentation

10.233.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.233.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.233.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.233.3 Constructor & Destructor Documentation

10.233.3.1 coco::polsppt_eval::polsppt_eval (const std::vector< interval > & *__x*, const variable_indicator & *__v*, const model & *__m*, unsigned int *_bs*, std::vector< d1func::d1func_point_t > & *_bt*, const diffI & *_bv*, const polsppt_interval_map & *_pm*, const polsppt_map_map & *_pmm*) [inline]

Constructor: __x is the box over which the function range shall be evaluated, __v is a variable indicator specifying the variables that have changed value since the last evaluation, __m specifies the DAG, and __c the cache (may be NULL). The boolean do_i determines whether the natural interval extension will use known node ranges for reducing the ranges.

Definition at line 186 of file dagd1func_pv.hpp.

10.233.3.2 coco::polsppt_eval::polsppt_eval (const polsppt_eval & __v) [inline]

Standard Copy Constructor

Definition at line 211 of file dagd1func_pv.hpp.

10.233.3.3 coco::polsppt_eval::~polsppt_eval ( ) [inline]

Standard Destructor

Definition at line 214 of file dagd1func_pv.hpp.

10.233.4 Member Function Documentation

10.233.4.1 void coco::polsppt_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 390 of file dagd1func_pv.hpp.

10.233.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.233.4.3 polsppt_ret_type coco::polsppt_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< polsppt_eval_type, expression_node, polsppt_ret_type, expression_const_walker >.

Definition at line 2378 of file dagd1func_pv.hpp.

10.233.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.233.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.233.4.6 std::vector<d1func::d1func_point_t>& coco::polsppt_eval::d1fp_prod_update (const diffI & *range1*, const std::vector< d1func::d1func_point_t > & *op1*, const interval & *coeff*, diffI *range2*, std::vector< d1func::d1func_point_t > *op2*, int *expo*, std::vector< d1func::d1func_point_t > & *ret*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 930 of file dagd1func_pv.hpp.

10.233.4.7 std::vector<d1func::d1func_point_t>& coco::polsppt_eval::d1fp_sum_update (const diffI & *range1*, const std::vector< d1func::d1func_point_t > & *op1*, const interval & *coeff*, diffI *range2*, std::vector< d1func::d1func_point_t > *op2*, int *expo*, std::vector< d1func::d1func_point_t > & *ret*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 629 of file dagd1func_pv.hpp.

10.233.4.8 void coco::polsppt_eval::d1fp_type_update (const interval & *coeff*, diffI & *range2*, std::vector< d1func::d1func_point_t > & *op2*, int *expon*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 406 of file dagd1func_pv.hpp.

10.233.4.9 void coco::polsppt_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< polsppt_eval_type, expression_node, polsppt_ret_type, expression_const_walker >.

Definition at line 240 of file dagd1func_pv.hpp.

10.233.4.10 int coco::polsppt_eval::initialize (const expression_node & *__data*) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 242 of file dagd1func_pv.hpp.

10.233.4.11 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The `__data` parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.233.4.12 bool coco::polsppt_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the range for this node is already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< polsppt_eval_type, expression_node, polsppt_ret_type, expression_const_walker >.

Definition at line 109 of file dagd1func_pv.hpp.

10.233.4.13 void coco::polsppt_eval::new_base (unsigned int _b, std::vector< d1func::d1func_point_t > & _p, const diffI & _v) [inline]

This method changes the base node to _b and the base type to _p.

Definition at line 230 of file dagd1func_pv.hpp.

10.233.4.14 void coco::polsppt_eval::new_box (const std::vector< interval > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation box to ___x. The parameter ___v specifies which variables have changed value since the last evaluation.

Definition at line 223 of file dagd1func_pv.hpp.

10.233.4.15 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.233.4.16 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.233.4.17 void coco::polsppt_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 398 of file dagd1func_pv.hpp.

10.233.4.18 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.233.4.19 expression_const_walker coco::polsppt_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 217 of file dagd1func_pv.hpp.

10.233.4.20 int coco::polsppt_eval::update (const polsppt_ret_type & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 400 of file dagd1func_pv.hpp.

10.233.4.21 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.233.4.22 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.233.4.23 int coco::polsppt_eval::update (const expression_node & __data, const polsppt_ret_type & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 1397 of file dagd1func_pv.hpp.

10.233.4.24 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter false.

Definition at line 763 of file search_graph.cc.

10.233.4.25 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.233.4.26 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.233.4.27 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

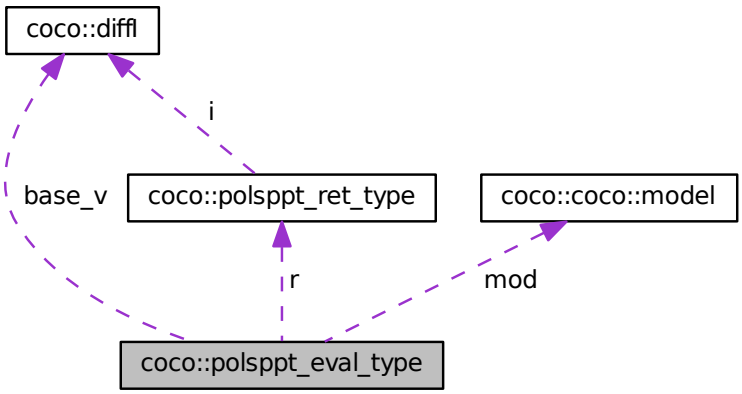
- dagd1func_pv.hpp

10.234 coco::polsppt_eval_type Struct Reference

Visitor data for polsppt_eval.

```
#include <dagd1func_pv.hpp>
```

Collaboration diagram for coco::polsppt_eval_type:



Classes

- struct help_t

Public Attributes

- const std::vector< interval > * x
- const model * mod
- union {
  void * p
  int info
  } u
- polsppt_ret_type r
- int order
- unsigned int base
- const diffI * base_v
- std::vector < d1func::d1func_point_t > * base_t
- const polsppt_interval_map * pbb_map
- const polsppt_map_map * pbb_imp
- unsigned int n

10.234.1 Detailed Description

This class is the visitor data type for the polsppt_eval evaluator.

10.234.2 Member Data Documentation

10.234.2.1 unsigned int coco::polsppt_eval_type::base

current base node

Definition at line 82 of file dagd1func_pv.hpp.

10.234.2.2 std::vector<d1func::d1func_point_t>* coco::polsppt_eval_type::base_t

current base node type

Definition at line 85 of file dagd1func_pv.hpp.

10.234.2.3 const diffI* coco::polsppt_eval_type::base_v

value at the base node

Definition at line 83 of file dagd1func_pv.hpp.

10.234.2.4 int coco::polsppt_eval_type::info

Definition at line 79 of file dagd1func_pv.hpp.

10.234.2.5 const model* coco::polsppt_eval_type::mod

the DAG

Definition at line 78 of file dagd1func_pv.hpp.

10.234.2.6 unsigned int coco::polsppt_eval_type::n

children counter

Definition at line 88 of file dagd1func_pv.hpp.

10.234.2.7 int coco::polsppt_eval_type::order

maximal order

Definition at line 81 of file dagd1func_pv.hpp.

10.234.2.8 void* coco::polsppt_eval_type::p

Definition at line 79 of file dagd1func_pv.hpp.

10.234.2.9 const polsppt_map_map* coco::polsppt_eval_type::pbb_imp

interesting nodes

Definition at line 87 of file dagd1func_pv.hpp.

10.234.2.10 const polsppt_interval_map* coco::polsppt_eval_type::pbb_map

special node(s) for the evaluator

Definition at line 86 of file dagd1func_pv.hpp.

10.234.2.11 polsppt_ret_type coco::polsppt_eval_type::r

return value

Definition at line 80 of file dagd1func_pv.hpp.

10.234.2.12 union { ... } coco::polsppt_eval_type::u

additional data for complex nodes

10.234.2.13 const std::vector<interval>* coco::polsppt_eval_type::x

the box

Definition at line 77 of file dagd1func_pv.hpp.

The documentation for this struct was generated from the following file:

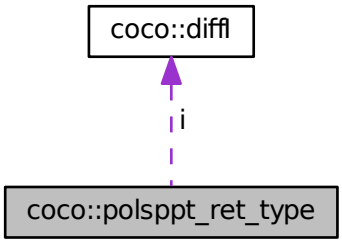
- dagd1func_pv.hpp

10.235 coco::polsppt_ret_type Struct Reference

Return type for polsppt_eval.

`#include <dagd1func_pv.hpp>`

Collaboration diagram for coco::polsppt_ret_type:



Public Attributes

- std::vector < d1func::d1func_point_t > p

- diffI i
- unsigned int n

10.235.1 Detailed Description

This class is the return data type for the polsppt_eval evaluator.

10.235.2 Member Data Documentation

10.235.2.1 diffI coco::polsppt_ret_type::i

Definition at line 52 of file dagd1func_pv.hpp.

10.235.2.2 unsigned int coco::polsppt_ret_type::n

Definition at line 53 of file dagd1func_pv.hpp.

10.235.2.3 std::vector<d1func::d1func_point_t> coco::polsppt_ret_type::p

Definition at line 51 of file dagd1func_pv.hpp.

The documentation for this struct was generated from the following file:

- dagd1func_pv.hpp

10.236 coco::prep_d_eval Class Reference

Preparation Evaluator for derivatives.

`#include <der_evaluator.h>`

Inheritance diagram for coco::prep_d_eval:

Collaboration diagram for coco::prep_d_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- prep_d_eval (std::vector< std::vector< double > > &__d, unsigned int _num_of_nodes)
- prep_d_eval (const prep_d_eval &__x)
- ~prep_d_eval ()
- bool is_cached (const expression_node &__data)
- int initialize (const expression_node &__data)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void retrieve_from_cache (const expression_node &__data)
- void initialize ()
- void calculate (const expression_node &__data)
- int update (bool __rval)
- int update (const expression_node &__data, bool __rval)
- bool calculate_value (bool eval_all)

10.236.1 Detailed Description

This evaluator is used to prepare the data structure needed for computing derivatives on a DAG. For a given DAG this evaluator need only be used once. It is a cached forward evaluator.

10.236.2 Member Typedef Documentation

10.236.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.236.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.236.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.236.3 Constructor & Destructor Documentation

10.236.3.1 coco::prep_d_eval::prep_d_eval (std::vector< std::vector< double > > & __d, unsigned int _num_of_nodes) [inline]

Constructor, which takes __d as derivative data structure which is to be prepared, and the number of nodes _num_of_nodes of the DAG.

Definition at line 76 of file der_evaluator.h.

10.236.3.2 coco::prep_d_eval::prep_d_eval (const prep_d_eval & __x) [inline]

Standard Copy Constructor

Definition at line 86 of file der_evaluator.h.

10.236.3.3 coco::prep_d_eval::~prep_d_eval ( ) [inline]

Standard Destructor

Definition at line 89 of file der_evaluator.h.

10.236.4 Member Function Documentation

10.236.4.1 void coco::prep_d_eval::calculate (const expression_node & __data) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 112 of file der_evaluator.h.

10.236.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.236.4.3 bool coco::prep_d_eval::calculate_value (bool eval_all) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 119 of file der_evaluator.h.

10.236.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.236.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.236.4.6 int coco::prep_d_eval::initialize (const expression_node & __data) [inline]

Initialize method for normal nodes

Definition at line 98 of file der_evaluator.h.

10.236.4.7 void coco::prep_d_eval::initialize ( ) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 110 of file der_evaluator.h.

10.236.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The ___data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.236.4.9 bool coco::prep_d_eval::is_cached (const expression_node & __data) [inline]

Check whether this node has already been visited

Definition at line 92 of file der_evaluator.h.

10.236.4.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & __data) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.236.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.236.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.236.4.13 void coco::prep_d_eval::retrieve_from_cache (const expression_node & __data) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 108 of file der_evaluator.h.

10.236.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.236.4.15 int coco::prep_d_eval::update (bool __rval) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 114 of file der_evaluator.h.

10.236.4.16 int coco::prep_d_eval::update (const expression_node & __data, bool __rval) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 116 of file der_evaluator.h.

10.236.4.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.236.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.236.4.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.236.4.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.236.4.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.236.4.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- der_evaluator.h

10.237 coco::prep_id_eval Class Reference

Preparation Evaluator for interval derivatives.

`#include <ider_evaluator.h>`

Inheritance diagram for coco::prep_id_eval:

Collaboration diagram for coco::prep_id_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- prep_id_eval (std::vector< std::vector< interval > > &__d, unsigned int _num_of_nodes)
- prep_id_eval (const prep_id_eval &__x)
- ~prep_id_eval ()
- bool is_cached (const expression_node &__data)
- int initialize (const expression_node &__data)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()

- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void retrieve_from_cache (const expression_node &__data)
- void initialize ()
- void calculate (const expression_node &__data)
- int update (bool __rval)
- int update (const expression_node &__data, bool __rval)
- bool calculate_value (bool eval_all)

10.237.1 Detailed Description

This evaluator is used to prepare the data structure needed for computing interval derivatives on a DAG. For a given DAG this evaluator need only be used once. It is a cached forward evaluator.

10.237.2 Member Typedef Documentation

10.237.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.237.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.237.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.237.3 Constructor & Destructor Documentation

10.237.3.1 coco::prep_id_eval::prep_id_eval (std::vector< std::vector< interval > > & __d, unsigned int _num_of_nodes) [inline]

Constructor, which takes ___d as interval derivative data structure which is to be prepared, and the number of nodes _num_of_nodes of the DAG.

Definition at line 79 of file ider_evaluator.h.

10.237.3.2 coco::prep_id_eval::prep_id_eval (const prep_id_eval & __x) [inline]

Standard Copy Constructor

Definition at line 89 of file ider_evaluator.h.

10.237.3.3 coco::prep_id_eval::~prep_id_eval ( ) [inline]

Standard Destructor

Definition at line 92 of file ider_evaluator.h.

10.237.4 Member Function Documentation

10.237.4.1 void coco::prep_id_eval::calculate (const expression_node & __data) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 115 of file ider_evaluator.h.

10.237.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.237.4.3 bool coco::prep_id_eval::calculate_value (bool eval_all) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 122 of file ider_evaluator.h.

10.237.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.237.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.237.4.6 int coco::prep_id_eval::initialize (const expression_node & *__data*) [inline]

Initialize method for normal nodes

Definition at line 101 of file ider_evaluator.h.

10.237.4.7 void coco::prep_id_eval::initialize ( ) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 113 of file ider_evaluator.h.

10.237.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & *__data*) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.237.4.9 bool coco::prep_id_eval::is_cached (const expression_node & *__data*) [inline]

Check whether this node has already been visited

Definition at line 95 of file ider_evaluator.h.

10.237.4.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & *__data*) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.237.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.237.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.237.4.13 void coco::prep_id_eval::retrieve_from_cache (const expression_node & __data) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 111 of file ider_evaluator.h.

10.237.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.237.4.15 int coco::prep_id_eval::update (bool __rval) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 117 of file ider_evaluator.h.

10.237.4.16 int coco::prep_id_eval::update (const expression_node & __data, bool __rval) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 119 of file ider_evaluator.h.

10.237.4.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.237.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.237.4.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.237.4.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.237.4.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.237.4.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- ider_evaluator.h

10.238 coco::prep_islp2_eval Class Reference

`#include <islp2_evaluator.h>`

Inheritance diagram for coco::prep_islp2_eval:

Collaboration diagram for coco::prep_islp2_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- prep_islp2_eval (std::vector< std::vector< interval > > &__c, std::vector< std::vector< interval > > &__d, std::vector< std::vector< interval > > &__h, unsigned int _num_of_nodes)
- prep_islp2_eval (const prep_islp2_eval &__x)
- ~prep_islp2_eval ()
- void initialize ()
- bool is_cached (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- int update (bool __rval)
- int update (const expression_node &__data, bool __rval)
- bool calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)

- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

10.238.1 Member Typedef Documentation

10.238.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.238.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.238.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.238.2 Constructor & Destructor Documentation

10.238.2.1 coco::prep_islp2_eval::prep_islp2_eval (std::vector< std::vector< interval > > & __c, std::vector< std::vector< interval > > & __d, std::vector< std::vector< interval > > & __h, unsigned int _num_of_nodes) [inline]

Definition at line 83 of file islp2_evaluator.h.

10.238.2.2 coco::prep_islp2_eval::prep_islp2_eval (const prep_islp2_eval & __x) [inline]

Definition at line 104 of file islp2_evaluator.h.

10.238.2.3 coco::prep_islp2_eval::~prep_islp2_eval ( ) [inline]

Definition at line 109 of file islp2_evaluator.h.

10.238.3 Member Function Documentation

10.238.3.1 void coco::prep_islp2_eval::calculate (const expression_node & __data) [inline]

Definition at line 132 of file islp2_evaluator.h.

10.238.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.238.3.3 bool coco::prep_islp2_eval::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< prep_islp2_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 144 of file islp2_evaluator.h.

10.238.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.238.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.238.3.6 void coco::prep_islp2_eval::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< prep_islp2_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 111 of file islp2_evaluator.h.

10.238.3.7 int coco::prep_islp2_eval::initialize (const expression_node & __data) [inline]

Definition at line 120 of file islp2_evaluator.h.

10.238.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.238.3.9 bool coco::prep_islp2_eval::is_cached (const expression_node & *__data*) [inline]

Definition at line 113 of file islp2_evaluator.h.

10.238.3.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & *__data*) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.238.3.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.238.3.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.238.3.13 void coco::prep_islp2_eval::retrieve_from_cache (const expression_node & *__data*) [inline]

Definition at line 118 of file islp2_evaluator.h.

10.238.3.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.238.3.15 int coco::prep_islp2_eval::update (bool *__rval*) [inline]

Definition at line 134 of file islp2_evaluator.h.

10.238.3.16 int coco::prep_islp2_eval::update (const expression_node & *__data*, bool *__rval*) [inline]

Definition at line 136 of file islp2_evaluator.h.

10.238.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.238.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.238.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.238.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.238.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.238.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- islp2_evaluator.h

10.239 coco::prep_islp2_eval_type Struct Reference

`#include <islp2_evaluator.h>`

Public Attributes

- std::vector< std::vector < interval > > * c
- std::vector< std::vector < interval > > * d
- std::vector< std::vector < interval > > * h

10.239.1 Member Data Documentation

10.239.1.1 std::vector<std::vector<interval> >* coco::prep_islp2_eval_type::c

Definition at line 69 of file islp2_evaluator.h.

10.239.1.2 std::vector<std::vector<interval> >* coco::prep_islp2_eval_type::d

Definition at line 70 of file islp2_evaluator.h.

10.239.1.3 std::vector<std::vector<interval> >* coco::prep_islp2_eval_type::h

Definition at line 71 of file islp2_evaluator.h.

The documentation for this struct was generated from the following file:

- islp2_evaluator.h

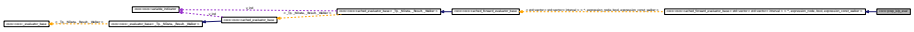
10.240 coco::prep_islp_eval Class Reference

Preparation Evaluator for first order slopes.

`#include <islp_evaluator.h>`

Inheritance diagram for coco::prep_islp_eval:

Collaboration diagram for coco::prep_islp_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- prep_islp_eval (std::vector< std::vector< interval > > &__d, unsigned int _num_of_nodes)
- prep_islp_eval (const prep_islp_eval &__x)
- ~prep_islp_eval ()
- bool is_cached (const expression_node &__data)
- int initialize (const expression_node &__data)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- void retrieve_from_cache (const expression_node &__data)
- void calculate (const expression_node &__data)
- int update (bool __rval)
- int update (const expression_node &__data, bool __rval)
- bool calculate_value (bool eval_all)

10.240.1 Detailed Description

This evaluator is used to prepare the data structure needed for computing first order slopes on a DAG. For a given DAG this evaluator need only be used once. It is a cached forward evaluator.

10.240.2 Member Typedef Documentation

10.240.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.240.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.240.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.240.3 Constructor & Destructor Documentation

10.240.3.1 coco::prep_islp_eval::prep_islp_eval (std::vector< std::vector< interval > > & *__d,* unsigned int *_num_of_nodes*) [inline]

Constructor, which takes __d as first order slope data structure which is to be prepared, and the number of nodes _num_of_nodes of the DAG.

Definition at line 91 of file islp_evaluator.h.

10.240.3.2 coco::prep_islp_eval::prep_islp_eval (const prep_islp_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 101 of file islp_evaluator.h.

10.240.3.3 coco::prep_islp_eval::~prep_islp_eval ( ) [inline]

Standard Destructor

Definition at line 104 of file islp_evaluator.h.

10.240.4 Member Function Documentation

10.240.4.1 void coco::prep_islp_eval::calculate (const expression_node & __data) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 127 of file islp_evaluator.h.

10.240.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.240.4.3 bool coco::prep_islp_eval::calculate_value (bool *eval_all*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 134 of file islp_evaluator.h.

10.240.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The ___data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.240.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.240.4.6 int coco::prep_islp_eval::initialize (const expression_node & __data) [inline]

Initialize method for normal nodes

Definition at line 113 of file islp_evaluator.h.

10.240.4.7 void coco::prep_islp_eval::initialize ( ) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 123 of file islp_evaluator.h.

10.240.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.240.4.9 bool coco::prep_islp_eval::is_cached (const expression_node & __data) [inline]

Check whether this node has already been visited

Definition at line 107 of file islp_evaluator.h.

10.240.4.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & __data) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.240.4.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.240.4.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.240.4.13 void coco::prep_islp_eval::retrieve_from_cache (const expression_node & __data) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 125 of file islp_evaluator.h.

10.240.4.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.240.4.15 int coco::prep_islp_eval::update (bool __rval) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 129 of file islp_evaluator.h.

10.240.4.16 int coco::prep_islp_eval::update (const expression_node & __data, bool __rval) [inline]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Definition at line 131 of file islp_evaluator.h.

10.240.4.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.240.4.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The ___data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.240.4.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter

false.

Definition at line 763 of file search_graph.cc.

10.240.4.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.240.4.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.240.4.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter true.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- islp_evaluator.h

10.241 coco::proj_rational Class Reference

#include <prointerval.h>

Public Member Functions

- proj_rational (int n=0, unsigned int d=1)
- proj_rational (const proj_rational &R)
- ~proj_rational ()
- proj_rational & operator= (const proj_rational &R)
- proj_rational & operator= (int n)
- proj_rational operator- () const
- proj_rational operator+ (const proj_rational &R) const
- proj_rational operator- (const proj_rational &R) const
- proj_rational operator* (const proj_rational &R) const
- proj_rational operator/ (const proj_rational &R) const
- proj_rational & operator+= (const proj_rational &R)
- proj_rational & operator-= (const proj_rational &R)
- proj_rational & operator*= (const proj_rational &R)

- proj_rational & operator/= (const proj_rational &R)
- interval ival () const
- bool operator== (const proj_rational &R) const
- bool operator!= (const proj_rational &R) const
- bool operator< (const proj_rational &R) const
- bool operator> (const proj_rational &R) const
- bool operator<= (const proj_rational &R) const
- bool operator>= (const proj_rational &R) const
- bool operator== (int n) const
- bool operator!= (int n) const
- bool operator< (int n) const
- bool operator> (int n) const
- bool operator<= (int n) const
- bool operator>= (int n) const
- int nominator () const
- int denominator () const
- proj_rational & set (int n, unsigned int d)

Friends

- proj_rational abs (const proj_rational &r)

10.241.1 Constructor & Destructor Documentation

10.241.1.1 coco::proj_rational::proj_rational (int *n* = 0, unsigned int *d* = 1) [inline]

Definition at line 49 of file prointerval.h.

10.241.1.2 coco::proj_rational::proj_rational (const proj_rational & *R*) [inline]

Definition at line 51 of file prointerval.h.

10.241.1.3 coco::proj_rational::~proj_rational ( ) [inline]

Definition at line 53 of file prointerval.h.

10.241.2 Member Function Documentation

10.241.2.1 int coco::proj_rational::denominator ( ) const [inline]

Definition at line 107 of file prointerval.h.

10.241.2.2 interval coco::proj_rational::ival ( ) const [inline]

Definition at line 74 of file prointerval.h.

10.241.2.3 int coco::proj_rational::nominator ( ) const [inline]

Definition at line 106 of file prointerval.h.

10.241.2.4 bool coco::proj_rational::operator!= (const proj_rational & *R*) const [inline]

Definition at line 79 of file prointerval.h.

10.241.2.5 bool coco::proj_rational::operator!= (int *n*) const [inline]

Definition at line 96 of file prointerval.h.

10.241.2.6 proj_rational coco::proj_rational::operator∗ (const proj_rational & *R*) const [inline]

Definition at line 96 of file prointerval.hpp.

10.241.2.7 proj_rational & coco::proj_rational::operator∗= (const proj_rational & *R*) [inline]

Definition at line 148 of file prointerval.hpp.

10.241.2.8 proj_rational coco::proj_rational::operator+ (const proj_rational & *R*) const [inline]

Definition at line 74 of file prointerval.hpp.

10.241.2.9 proj_rational & coco::proj_rational::operator+= (const proj_rational & *R*) [inline]

Definition at line 126 of file prointerval.hpp.

10.241.2.10 proj_rational coco::proj_rational::operator- ( ) const [inline]

Definition at line 61 of file prointerval.h.

10.241.2.11 proj_rational coco::proj_rational::operator- (const proj_rational & *R*) const [inline]

Definition at line 85 of file prointerval.hpp.

10.241.2.12 proj_rational & coco::proj_rational::operator-= (const proj_rational & *R*) [inline]

Definition at line 137 of file prointerval.hpp.

10.241.2.13 proj_rational coco::proj_rational::operator/ (const proj_rational & *R*) const [inline]

Definition at line 109 of file prointerval.hpp.

10.241.2.14 proj_rational & coco::proj_rational::operator/= (const proj_rational & *R*) [inline]

Definition at line 161 of file prointerval.hpp.

10.241.2.15 bool coco::proj_rational::operator< (const proj_rational & *R*) const [inline]

Definition at line 82 of file prointerval.h.

10.241.2.16 bool coco::proj_rational::operator< (int *n*) const [inline]

Definition at line 98 of file prointerval.h.

10.241.2.17 bool coco::proj_rational::operator<= (const proj_rational & *R*) const [inline]

Definition at line 88 of file prointerval.h.

10.241.2.18 bool coco::proj_rational::operator<= (int *n*) const [inline]

Definition at line 102 of file prointerval.h.

10.241.2.19 proj_rational& coco::proj_rational::operator= (const proj_rational & *R*) [inline]

Definition at line 55 of file prointerval.h.

10.241.2.20 proj_rational& coco::proj_rational::operator= (int *n*) [inline]

Definition at line 58 of file prointerval.h.

10.241.2.21 bool coco::proj_rational::operator== (const proj_rational & *R*) const [inline]

Definition at line 76 of file prointerval.h.

10.241.2.22 bool coco::proj_rational::operator== (int *n*) const [inline]

Definition at line 94 of file prointerval.h.

10.241.2.23 bool coco::proj_rational::operator> (const proj_rational & *R*) const [inline]

Definition at line 85 of file prointerval.h.

10.241.2.24 bool coco::proj_rational::operator> (int *n*) const [inline]

Definition at line 100 of file prointerval.h.

10.241.2.25 bool coco::proj_rational::operator>= (const proj_rational & *R*) const [inline]

Definition at line 91 of file prointerval.h.

10.241.2.26 bool coco::proj_rational::operator>= (int *n*) const [inline]

Definition at line 104 of file prointerval.h.

10.241.2.27 proj_rational& coco::proj_rational::set (int *n*, unsigned int *d*) [inline]

Definition at line 108 of file prointerval.h.

10.241.3 Friends And Related Function Documentation

10.241.3.1 proj_rational abs (const proj_rational & *r*) [friend]

Definition at line 120 of file prointerval.h.

The documentation for this class was generated from the following files:

- prointerval.h
- prointerval.hpp

10.242 coco::projective_interval Class Reference

`#include <prointerval.h>`

Public Member Functions

- projective_interval ()
- projective_interval (const proj_interval &_p)
- projective_interval (const interval &_t)
- projective_interval (const interval &_i, const interval &_t)
- projective_interval (const interval &_i, int _m, const interval &_t)
- projective_interval (const interval &_i, const proj_rational &_r, const interval &_t)
- projective_interval (const interval &_i, const proj_interval &_p)
- projective_interval (const interval &_i, int _m, const proj_interval &_p)
- projective_interval (const interval &_i, const proj_rational &_r, const proj_interval &_p)
- projective_interval (double lo, double up, const proj_interval &_p)
- projective_interval (double lo, double up, const proj_rational &_r, const proj_interval &_p)
- projective_interval (double lo, double up, int _m, const proj_interval &_p)
- projective_interval (double d, const proj_interval &_p)
- projective_interval (double d, const proj_rational &_r, const proj_interval &_p)
- projective_interval (double d, int _m, const proj_interval &_p)
- projective_interval (int d, const proj_interval &_p)
- projective_interval (unsigned d, const proj_interval &_p)
- projective_interval (long d, const proj_interval &_p)
- projective_interval (unsigned long d, const proj_interval &_p)
- virtual ~projective_interval ()
- void set (double lo, double up, const proj_interval &_p)
- void set_i (double lo, double up)
- void set_i (const interval &j)
- void set_t (double lo, double up)
- void set_t (const interval &u)
- void set_r (int n, unsigned int d)
- void set_r (const proj_rational &s)
- const interval & pi () const
- const interval & pt () const
- const proj_rational & pr () const
- double i_inf () const
- double i_sup () const
- double t_inf () const
- double t_sup () const
- double t_exp () const
- double t_exp_n () const
- double t_exp_d () const
- bool empty () const
- bool is_empty () const
- bool is_i_thin () const

- bool is_t_thin () const
- bool is_unbounded_below () const
- bool is_unbounded_above () const
- bool is_entire () const
- bool is_bounded () const
- bool i_subset (const proj_interval &x) const
- bool t_subset (const proj_interval &x) const
- bool certain_subset (const proj_interval &x) const
- bool possible_subset (const proj_interval &x) const
- bool i_superset (const proj_interval &x) const
- bool t_superset (const proj_interval &x) const
- bool certain_superset (const proj_interval &x) const
- bool possible_superset (const proj_interval &x) const
- bool i_proper_subset (const interval &x) const
- bool t_proper_subset (const interval &x) const
- bool certain_proper_subset (const proj_interval &x) const
- bool possible_proper_subset (const proj_interval &x) const
- bool i_proper_superset (const proj_interval &x) const
- bool t_proper_superset (const proj_interval &x) const
- bool certain_proper_superset (const proj_interval &x) const
- bool possible_proper_superset (const proj_interval &x) const
- bool disjoint (const proj_interval &x) const
- double rel_width () const
- proj_interval & intersectwith (const proj_interval &x)
- proj_interval & hull (const proj_interval &x)
- proj_interval & hulldiff (const proj_interval &x)
- bool certainly_contains (const interval &x) const
- bool probably_contains (const interval &x) const
- void set_bounds (double a, double b)
- void set_lb (double d)
- void set_ub (double d)
- proj_interval & operator= (const proj_interval &x)
- proj_interval & operator= (double d)
- proj_interval & operator= (int d)
- proj_interval & operator= (unsigned d)
- proj_interval & operator= (long d)
- proj_interval & operator= (unsigned long d)
- proj_interval operator- () const
- proj_interval & ipow (int n)
- proj_interval & imin (const proj_interval &x)
- proj_interval & imax (const proj_interval &x)
- proj_interval & round_to_integer ()
- proj_interval & intersect_div (const proj_interval &__i, const proj_interval &__j)
- proj_interval & intersect_power (const proj_interval &__i, int __n)
- proj_interval & intersect_invsqr_wc (const proj_interval &__i, double __l, double __d)
- proj_interval & intersect_invpower_wc (const proj_interval &__i, double __l, int __n)
- proj_interval & intersect_invsin_wc (const proj_interval &__i, double __l, double __d)
- proj_interval & intersect_invcos_wc (const proj_interval &__i, double __l, double __d)
- proj_interval & intersect_invcosh_wc (const proj_interval &__i, double __l, double __d)

- proj_interval & intersect_invgauss_wc (const proj_interval &__i, double __l, double __m, double __s)
- double project (double __d) const
- proj_interval & operator+= (const proj_interval &a)
- proj_interval & operator+= (double a)
- proj_interval & operator-= (const proj_interval &a)
- proj_interval & operator-= (double a)
- proj_interval & operator*= (const proj_interval &a)
- proj_interval & operator*= (double a)
- proj_interval & operator/= (const proj_interval &a)
- proj_interval & operator/= (double a)
- double mid () const
- double diam () const
- double width () const
- double relDiam () const
- double rad () const
- double mig () const
- double mag () const
- double dist (const proj_interval &x) const
- double safeguarded_mid () const
- bool operator== (const proj_interval &a) const
- bool operator!= (const proj_interval &a) const

Static Public Member Functions

- static proj_interval EMPTY (const proj_interval &p)

Friends

- template<class J >
  projective_interval< J > operator+ (const projective_interval< J > &a, const projective_interval< J > &b)
- template<class J >
  projective_interval< J > operator+ (const projective_interval< J > &a, double b)
- template<class J >
  projective_interval< J > operator+ (double b, const projective_interval< J > &a)
- template<class J >
  projective_interval< J > operator- (const projective_interval< J > &a, const projective_interval< J > &b)
- template<class J >
  projective_interval< J > operator- (const projective_interval< J > &a, double b)
- template<class J >
  projective_interval< J > operator- (double b, const projective_interval< J > &a)
- template<class J >
  projective_interval< J > cancel (const projective_interval< J > &a, const projective_interval< J > &b)
- template<class J >
  projective_interval< J > operator* (const projective_interval< J > &a, const projective_interval< J > &b)

- template<class J >
  projective_interval< J > operator∗ (const projective_interval< J > &a, double b)
- template<class J >
  projective_interval< J > operator∗ (double b, const projective_interval< J > &a)
- template<class J >
  projective_interval< J > operator/ (const projective_interval< J > &a, const projective_interval< J > &b)
- template<class J >
  projective_interval< J > operator/ (const projective_interval< J > &a, double b)
- template<class J >
  projective_interval< J > operator/ (double b, const projective_interval< J > &a)
- template<class J >
  projective_interval< J > acos (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > abs (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > acosh (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > acot (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > acoth (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > asin (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > asinh (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > atan (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > atanh (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > cos (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > cosh (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > cot (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > coth (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > exp (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > exp10 (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > exp2 (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > expm1 (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > log (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > log10 (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > log1p (const projective_interval< J > &x)

- template<class J >
  projective_interval< J > log2 (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > power (const projective_interval< J > &x, int n)
- template<class J >
  projective_interval< J > pow (const projective_interval< J > &x, const projective_interval< J > &y)
- template<class J >
  projective_interval< J > sin (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > sinh (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > sqr (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > sqrt (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > tan (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > tanh (const projective_interval< J > &x)
- template<class J >
  projective_interval< J > imax (const projective_interval< J > &x, const projective_interval< J > &y)
- template<class J >
  projective_interval< J > imin (const projective_interval< J > &x, const projective_interval< J > &y)
- template<class J >
  projective_interval< J > atan2 (const projective_interval< J > &x, const projective_interval< J > &y)
- template<class J >
  projective_interval< J > division_part1 (const projective_interval< J > &x, const projective_interval< J > &y, bool &b)
- template<class J >
  projective_interval< J > division_part2 (const projective_interval< J > &x, const projective_interval< J > &y, bool b)

10.242.1 Constructor & Destructor Documentation

10.242.1.1 coco::projective_interval::projective_interval ( ) [inline]

empty unusable projective interval

Definition at line 143 of file prointerval.h.

10.242.1.2 coco::projective_interval::projective_interval (const proj_interval & _p) [inline]

standard copy constructor

Definition at line 145 of file prointerval.h.

10.242.1.3 coco::projective_interval::projective_interval (const interval & _t) [inline]

constructors for the first projective interval

Definition at line 148 of file prointerval.h.

10.242.1.4 coco::projective_interval::projective_interval (const interval & _i, const interval & _t) [inline]

Definition at line 150 of file prointerval.h.

10.242.1.5 coco::projective_interval::projective_interval (const interval & _i, int _m, const interval & _t) [inline]

Definition at line 152 of file prointerval.h.

10.242.1.6 coco::projective_interval::projective_interval (const interval & _i, const proj_rational & _r, const interval & _t) [inline]

Definition at line 154 of file prointerval.h.

10.242.1.7 coco::projective_interval::projective_interval (const interval & _i, const proj_interval & _p) [inline]

constructors for dependent projective intervals

Definition at line 158 of file prointerval.h.

10.242.1.8 coco::projective_interval::projective_interval (const interval & _i, int _m, const proj_interval & _p) [inline]

Definition at line 160 of file prointerval.h.

10.242.1.9 coco::projective_interval::projective_interval (const interval & _i, const proj_rational & _r, const proj_interval & _p) [inline]

Definition at line 162 of file prointerval.h.

10.242.1.10 coco::projective_interval::projective_interval (double lo, double up, const proj_interval & _p) [inline]

Definition at line 164 of file prointerval.h.

10.242.1.11 coco::projective_interval::projective_interval (double lo, double up, const proj_rational & _r, const proj_interval & _p) [inline]

Definition at line 166 of file prointerval.h.

10.242.1.12 coco::projective_interval::projective_interval (double lo, double up, int _m, const proj_interval & _p) [inline]

Definition at line 168 of file prointerval.h.

10.242.1.13 coco::projective_interval::projective_interval (double d, const proj_interval & _p) [inline]

Definition at line 170 of file prointerval.h.

10.242.1.14 coco::projective_interval::projective_interval (double *d,* const proj_rational & *_r,* const proj_interval & *_p*) [inline]

Definition at line 172 of file prointerval.h.

10.242.1.15 coco::projective_interval::projective_interval (double *d,* int *_m,* const proj_interval & *_p*) [inline]

Definition at line 174 of file prointerval.h.

10.242.1.16 coco::projective_interval::projective_interval (int *d,* const proj_interval & *_p*) [inline]

Definition at line 176 of file prointerval.h.

10.242.1.17 coco::projective_interval::projective_interval (unsigned *d,* const proj_interval & *_p*) [inline]

Definition at line 178 of file prointerval.h.

10.242.1.18 coco::projective_interval::projective_interval (long *d,* const proj_interval & *_p*) [inline]

Definition at line 180 of file prointerval.h.

10.242.1.19 coco::projective_interval::projective_interval (unsigned long *d,* const proj_interval & *_p*) [inline]

Definition at line 182 of file prointerval.h.

10.242.1.20 virtual coco::projective_interval::~projective_interval ( ) [inline, virtual]

standard destructor

Definition at line 186 of file prointerval.h.

10.242.2 Member Function Documentation

10.242.2.1 bool coco::projective_interval::certain_proper_subset (const proj_interval & *x*) const

Definition at line 289 of file prointerval.hpp.

10.242.2.2 bool coco::projective_interval::certain_proper_superset (const proj_interval & *x*) const [inline]

Definition at line 246 of file prointerval.h.

10.242.2.3 bool coco::projective_interval::certain_subset (const proj_interval & *x*) const

Definition at line 271 of file prointerval.hpp.

10.242.2.4 bool coco::projective_interval::certain_superset (const proj_interval & *x*) const [inline]

Definition at line 232 of file prointerval.h.

10.242.2.5 bool coco::projective_interval::certainly_contains (const interval & *x*) const [inline]

Definition at line 262 of file prointerval.h.

10.242.2.6 double coco::projective_interval::diam ( ) const

Definition at line 651 of file prointerval.hpp.

10.242.2.7 bool coco::projective_interval::disjoint (const proj_interval & *x*) const

Definition at line 307 of file prointerval.hpp.

10.242.2.8 double coco::projective_interval::dist (const proj_interval & *x*) const

Returns an upper bound for the Hausdorff distance of this proj_interval and the proj_interval x, i.e.

x.dist(y) == max{ abs(x.inf()-y.inf()), abs(x.sup() - y.sup()) }

Special cases in the extended system:

- x.dist(y) == NaN for x == [EMPTY] or y == [EMPTY]

Definition at line 688 of file prointerval.hpp.

10.242.2.9 static proj_interval coco::projective_interval::EMPTY (const proj_interval & *p*) [inline, static]

Definition at line 210 of file prointerval.h.

10.242.2.10 bool coco::projective_interval::empty ( ) const [inline]

Definition at line 213 of file prointerval.h.

10.242.2.11 projective_interval< I > & coco::projective_interval::hull (const proj_interval & *x*)

Definition at line 348 of file prointerval.hpp.

10.242.2.12 projective_interval< I > & coco::projective_interval::hulldiff (const proj_interval & *x*)

Definition at line 366 of file prointerval.hpp.

10.242.2.13 double coco::projective_interval::i_inf ( ) const [inline]

Definition at line 202 of file prointerval.h.

10.242.2.14 bool coco::projective_interval::i_proper_subset (const interval & *x*) const [inline]

Definition at line 236 of file prointerval.h.

10.242.2.15 bool coco::projective_interval::i_proper_superset (const proj_interval & *x*) const [inline]

Definition at line 242 of file prointerval.h.

10.242.2.16 bool coco::projective_interval::i_subset (const proj_interval & *x*) const [inline]

Definition at line 224 of file prointerval.h.

10.242.2.17 double coco::projective_interval::i_sup ( ) const [inline]

Definition at line 203 of file prointerval.h.

10.242.2.18 bool coco::projective_interval::i_superset (const proj_interval & *x*) const [inline]

Definition at line 228 of file prointerval.h.

10.242.2.19 projective_interval< I > & coco::projective_interval::imax (const proj_interval & *x*)

The method changes this proj_interval to the maximum of itself and x.

Definition at line 407 of file prointerval.hpp.

10.242.2.20 projective_interval< I > & coco::projective_interval::imin (const proj_interval & *x*)

The method changes this proj_interval to the minimum of itself and x.

Definition at line 388 of file prointerval.hpp.

10.242.2.21 projective_interval< I > & coco::projective_interval::intersect_div (const proj_interval & *__i*, const proj_interval & *__j*)

Definition at line 435 of file prointerval.hpp.

10.242.2.22 projective_interval< I > & coco::projective_interval::intersect_invcos_wc (const proj_interval & *__i*, double *__l*, double *__d*)

Definition at line 492 of file prointerval.hpp.

10.242.2.23 projective_interval< I > & coco::projective_interval::intersect_invcosh_wc (const proj_interval & *__i*, double *__l*, double *__d*)

Definition at line 506 of file prointerval.hpp.

10.242.2.24 projective_interval< I > & coco::projective_interval::intersect_invgauss_wc (const proj_interval & *__i*, double *__l*, double *__m*, double *__s*)

Definition at line 515 of file prointerval.hpp.

10.242.2.25 projective_interval< I > & coco::projective_interval::intersect_invpower_wc (const proj_interval & *__i*, double *__l*, int *__n*)

Definition at line 467 of file prointerval.hpp.

10.242.2.26 projective_interval< I > & coco::projective_interval::intersect_invsin_wc (const proj_interval & __i, double __l, double __d)

Definition at line 476 of file prointerval.hpp.

10.242.2.27 projective_interval< I > & coco::projective_interval::intersect_invsqr_wc (const proj_interval & __i, double __l, double __d)

Definition at line 458 of file prointerval.hpp.

10.242.2.28 projective_interval< I > & coco::projective_interval::intersect_power (const proj_interval & __i, int __n)

Definition at line 445 of file prointerval.hpp.

10.242.2.29 projective_interval< I > & coco::projective_interval::intersectwith (const proj_interval & x)

Definition at line 329 of file prointerval.hpp.

10.242.2.30 projective_interval< I > & coco::projective_interval::ipow (int n)

The method changes this interval to the `n-th` power.

Definition at line 380 of file prointerval.hpp.

10.242.2.31 bool coco::projective_interval::is_bounded ( ) const [inline]

Definition at line 222 of file prointerval.h.

10.242.2.32 bool coco::projective_interval::is_empty ( ) const [inline]

Definition at line 214 of file prointerval.h.

10.242.2.33 bool coco::projective_interval::is_entire ( ) const [inline]

Definition at line 221 of file prointerval.h.

10.242.2.34 bool coco::projective_interval::is_i_thin ( ) const [inline]

Definition at line 215 of file prointerval.h.

10.242.2.35 bool coco::projective_interval::is_t_thin ( ) const [inline]

Definition at line 216 of file prointerval.h.

10.242.2.36 bool coco::projective_interval::is_unbounded_above ( ) const [inline]

Definition at line 219 of file prointerval.h.

10.242.2.37 bool coco::projective_interval::is_unbounded_below ( ) const [inline]

Definition at line 217 of file prointerval.h.

10.242.2.38 double coco::projective_interval::mag ( ) const

Returns the magnitude of this proj_interval, i.e.

a.mag() == max{abs(y) : y in a.i }

Special cases in the extended system:

- a.mag() == NaN for a == [EMPTY]
- a.mag() == +INF for any infinite proj_interval

Definition at line 682 of file prointerval.hpp.

10.242.2.39 double coco::projective_interval::mid ( ) const

Returns the midpoint of the interval part of this proj_interval, i.e.

a.mid() == (a.i.inf()+a.i.sup())/2

The following special cases are distinguished:

- a.mid() == NaN for a == [EMPTY]
- a.mid() == 0 for a == [ENTIRE]
- a.mid() == +INF for a == [a, INFTY]
- a.mid() == -INF for a == [-INFTY, a]

Definition at line 645 of file prointerval.hpp.

10.242.2.40 double coco::projective_interval::mig ( ) const

Returns the mignitude of this proj_interval, i.e.

a.mig() == min{abs(y) : y in a.i }

Special cases in the extended system:

- a.mig() == NaN for a == [EMPTY]

Definition at line 676 of file prointerval.hpp.

10.242.2.41 bool coco::projective_interval::operator!= (const proj_interval & *a*) const [inline]

Definition at line 437 of file prointerval.h.

10.242.2.42 projective_interval< I > & coco::projective_interval::operator*= (const proj_interval & *a*)

Definition at line 615 of file prointerval.hpp.

10.242.2.43 projective_interval< I > & coco::projective_interval::operator*= (double *a*)

Definition at line 623 of file prointerval.hpp.

10.242.2.44 projective_interval< I > & coco::projective_interval::operator+= (const proj_interval & *a*)

Definition at line 543 of file prointerval.hpp.

10.242.2.45 projective_interval< I > & coco::projective_interval::operator+= (double *a*)

Definition at line 561 of file prointerval.hpp.

10.242.2.46 proj_interval coco::projective_interval::operator- ( ) const [inline]

Definition at line 285 of file prointerval.h.

10.242.2.47 projective_interval< I > & coco::projective_interval::operator-= (const proj_interval & *a*)

Definition at line 579 of file prointerval.hpp.

10.242.2.48 projective_interval< I > & coco::projective_interval::operator-= (double *a*)

Definition at line 597 of file prointerval.hpp.

10.242.2.49 projective_interval< I > & coco::projective_interval::operator/= (const proj_interval & *a*)

Definition at line 630 of file prointerval.hpp.

10.242.2.50 projective_interval< I > & coco::projective_interval::operator/= (double *a*)

Definition at line 638 of file prointerval.hpp.

10.242.2.51 proj_interval& coco::projective_interval::operator= (const proj_interval & *x*) [inline]

Definition at line 276 of file prointerval.h.

10.242.2.52 proj_interval& coco::projective_interval::operator= (double *d*) [inline]

Definition at line 277 of file prointerval.h.

10.242.2.53 proj_interval& coco::projective_interval::operator= (int *d*) [inline]

Definition at line 278 of file prointerval.h.

10.242.2.54 proj_interval& coco::projective_interval::operator= (unsigned *d*) [inline]

Definition at line 279 of file prointerval.h.

10.242.2.55 proj_interval& coco::projective_interval::operator= (long *d*) [inline]

Definition at line 280 of file prointerval.h.

10.242.2.56 proj_interval& coco::projective_interval::operator= (unsigned long *d*) [inline]

Definition at line 281 of file prointerval.h.

10.242.2.57 bool coco::projective_interval::operator== (const proj_interval & *a*) const [inline]

Definition at line 434 of file prointerval.h.

10.242.2.58 const interval& coco::projective_interval::pi ( ) const [inline]

Definition at line 198 of file prointerval.h.

10.242.2.59 bool coco::projective_interval::possible_proper_subset (const proj_interval & *x*) const

Definition at line 298 of file prointerval.hpp.

10.242.2.60 bool coco::projective_interval::possible_proper_superset (const proj_interval & *x*) const [inline]

Definition at line 248 of file prointerval.h.

10.242.2.61 bool coco::projective_interval::possible_subset (const proj_interval & *x*) const

Definition at line 280 of file prointerval.hpp.

10.242.2.62 bool coco::projective_interval::possible_superset (const proj_interval & *x*) const [inline]

Definition at line 234 of file prointerval.h.

10.242.2.63 const proj_rational& coco::projective_interval::pr ( ) const [inline]

Definition at line 200 of file prointerval.h.

10.242.2.64 bool coco::projective_interval::probably_contains (const interval & *x*) const [inline]

Definition at line 267 of file prointerval.h.

10.242.2.65 double coco::projective_interval::project (double *__d*) const

Definition at line 529 of file prointerval.hpp.

10.242.2.66 const interval& coco::projective_interval::pt ( ) const [inline]

Definition at line 199 of file prointerval.h.

10.242.2.67 double coco::projective_interval::rad ( ) const

Returns an upper bound for the radius of this proj_interval, i.e.

a.rad() = (a.i.sup() - a.i.inf()) / 2

Special cases in the extended system:

- a.rad() == NaN for a == [EMPTY]
- a.rad() == +INF for any infinite proj_interval

Definition at line 670 of file prointerval.hpp.

10.242.2.68 double coco::projective_interval::rel_width ( ) const [inline]

Definition at line 252 of file prointerval.h.

10.242.2.69 double coco::projective_interval::relDiam ( ) const

Returns an upper bound for the relative diameter (width) of this proj_interval, i.e.

a.relDiam == a.diam() if a.i.mig() is less than the smallest normalized number

a.relDiam == a.diam() / a.i.mig() else

Special cases in the extended system:

- a.relDiam() == NaN for a == [EMPTY]
- a.relDiam() == +INF for any infinite proj_interval

Definition at line 664 of file prointerval.hpp.

10.242.2.70 projective_interval< I > & coco::projective_interval::round_to_integer ( )

This method rounds the proj_interval inward to integer borders.

Definition at line 426 of file prointerval.hpp.

10.242.2.71 double coco::projective_interval::safeguarded_mid ( ) const

Returns the safeguarded mid of this proj_interval, i.e.

Definition at line 696 of file prointerval.hpp.

10.242.2.72 void coco::projective_interval::set (double *lo*, double *up*, const proj_interval & *_p*)

Definition at line 221 of file prointerval.hpp.

10.242.2.73 void coco::projective_interval::set_bounds (double *a*, double *b*) [inline]

Definition at line 272 of file prointerval.h.

10.242.2.74 void coco::projective_interval::set_i (double *lo*, double *up*)

Definition at line 230 of file prointerval.hpp.

10.242.2.75 void coco::projective_interval::set_i (const interval & *j*)

Definition at line 237 of file prointerval.hpp.

10.242.2.76 void coco::projective_interval::set_lb (double *d*) [inline]

Definition at line 273 of file prointerval.h.

10.242.2.77 void coco::projective_interval::set_r (int *n*, unsigned int *d*)

Definition at line 258 of file prointerval.hpp.

10.242.2.78 void coco::projective_interval::set_r (const proj_rational & *s*)

Definition at line 265 of file prointerval.hpp.

10.242.2.79 void coco::projective_interval::set_t (double *lo,* double *up*)

Definition at line 244 of file prointerval.hpp.

10.242.2.80 void coco::projective_interval::set_t (const interval & *u*)

Definition at line 251 of file prointerval.hpp.

10.242.2.81 void coco::projective_interval::set_ub (double *d*) [inline]

Definition at line 274 of file prointerval.h.

10.242.2.82 double coco::projective_interval::t_exp ( ) const [inline]

Definition at line 206 of file prointerval.h.

10.242.2.83 double coco::projective_interval::t_exp_d ( ) const [inline]

Definition at line 208 of file prointerval.h.

10.242.2.84 double coco::projective_interval::t_exp_n ( ) const [inline]

Definition at line 207 of file prointerval.h.

10.242.2.85 double coco::projective_interval::t_inf ( ) const [inline]

Definition at line 204 of file prointerval.h.

10.242.2.86 bool coco::projective_interval::t_proper_subset (const interval & *x*) const [inline]

Definition at line 238 of file prointerval.h.

10.242.2.87 bool coco::projective_interval::t_proper_superset (const proj_interval & *x*) const [inline]

Definition at line 244 of file prointerval.h.

10.242.2.88 bool coco::projective_interval::t_subset (const proj_interval & *x*) const [inline]

Definition at line 225 of file prointerval.h.

10.242.2.89 double coco::projective_interval::t_sup ( ) const [inline]

Definition at line 205 of file prointerval.h.

10.242.2.90 bool coco::projective_interval::t_superset (const proj_interval & *x*) const [inline]

Definition at line 230 of file prointerval.h.

10.242.2.91 double coco::projective_interval::width ( ) const

Definition at line 658 of file prointerval.hpp.

10.242.3 Friends And Related Function Documentation

10.242.3.1 template<class J > projective_interval<J> abs (const projective_interval< J > & x) [friend]

10.242.3.2 template<class J > projective_interval<J> acos (const projective_interval< J > & x) [friend]

10.242.3.3 template<class J > projective_interval<J> acosh (const projective_interval< J > & x) [friend]

10.242.3.4 template<class J > projective_interval<J> acot (const projective_interval< J > & x) [friend]

10.242.3.5 template<class J > projective_interval<J> acoth (const projective_interval< J > & x) [friend]

10.242.3.6 template<class J > projective_interval<J> asin (const projective_interval< J > & x) [friend]

10.242.3.7 template<class J > projective_interval<J> asinh (const projective_interval< J > & x) [friend]

10.242.3.8 template<class J > projective_interval<J> atan (const projective_interval< J > & x) [friend]

10.242.3.9 template<class J > projective_interval<J> atan2 (const projective_interval< J > & x, const projective_interval< J > & y) [friend]

10.242.3.10 template<class J > projective_interval<J> atanh (const projective_interval< J > & x) [friend]

10.242.3.11 template<class J > projective_interval<J> cancel (const projective_interval< J > & a, const projective_interval< J > & b) [friend]

10.242.3.12 template<class J > projective_interval<J> cos (const projective_interval< J > & x) [friend]

10.242.3.13 template<class J > projective_interval<J> cosh (const projective_interval< J > & x) [friend]

10.242.3.14 template<class J > projective_interval<J> cot (const projective_interval< J > & x) [friend]

10.242.3.15 template<class J > projective_interval<J> coth (const projective_interval< J > & x) [friend]

10.242.3.16 template<class J > projective_interval<J> division_part1 (const projective_interval< J > & *x*, const projective_interval< J > & *y*, bool & *b*) [friend]

10.242.3.17 template<class J > projective_interval<J> division_part2 (const projective_interval< J > & *x*, const projective_interval< J > & *y*, bool *b*) [friend]

10.242.3.18 template<class J > projective_interval<J> exp (const projective_interval< J > & *x*) [friend]

10.242.3.19 template<class J > projective_interval<J> exp10 (const projective_interval< J > & *x*) [friend]

10.242.3.20 template<class J > projective_interval<J> exp2 (const projective_interval< J > & *x*) [friend]

10.242.3.21 template<class J > projective_interval<J> expm1 (const projective_interval< J > & *x*) [friend]

10.242.3.22 template<class J > projective_interval<J> imax (const projective_interval< J > & *x*, const projective_interval< J > & *y*) [friend]

10.242.3.23 template<class J > projective_interval<J> imin (const projective_interval< J > & *x*, const projective_interval< J > & *y*) [friend]

10.242.3.24 template<class J > projective_interval<J> log (const projective_interval< J > & *x*) [friend]

10.242.3.25 template<class J > projective_interval<J> log10 (const projective_interval< J > & *x*) [friend]

10.242.3.26 template<class J > projective_interval<J> log1p (const projective_interval< J > & *x*) [friend]

10.242.3.27 template<class J > projective_interval<J> log2 (const projective_interval< J > & *x*) [friend]

10.242.3.28 template<class J > projective_interval<J> operator* (const projective_interval< J > & *a*, const projective_interval< J > & *b*) [friend]

10.242.3.29 template<class J > projective_interval<J> operator* (const projective_interval< J > & *a*, double *b*) [friend]

10.242.3.30 template<class J > projective_interval<J> operator* (double *b*, const projective_interval< J > & *a*) [friend]

10.242.3.31 template<class J > projective_interval<J> operator+ (const projective_interval< J > & *a*, const projective_interval< J > & *b*) [friend]

10.242.3.32 template<class J > projective_interval<J> operator+ (const projective_interval< J > & *a*, double *b*) [friend]

10.242.3.33 template<class J > projective_interval<J> operator+ (double *b*, const projective_interval< J > & *a*) [friend]

10.242.3.34 template<class J > projective_interval<J> operator- (const projective_interval< J > & *a,* const projective_interval< J > & *b*) [friend]

10.242.3.35 template<class J > projective_interval<J> operator- (const projective_interval< J > & *a,* double *b*) [friend]

10.242.3.36 template<class J > projective_interval<J> operator- (double *b,* const projective_interval< J > & *a*) [friend]

10.242.3.37 template<class J > projective_interval<J> operator/ (const projective_interval< J > & *a,* const projective_interval< J > & *b*) [friend]

10.242.3.38 template<class J > projective_interval<J> operator/ (const projective_interval< J > & *a,* double *b*) [friend]

10.242.3.39 template<class J > projective_interval<J> operator/ (double *b,* const projective_interval< J > & *a*) [friend]

10.242.3.40 template<class J > projective_interval<J> pow (const projective_interval< J > & *x,* const projective_interval< J > & *y*) [friend]

10.242.3.41 template<class J > projective_interval<J> power (const projective_interval< J > & *x,* int *n*) [friend]

10.242.3.42 template<class J > projective_interval<J> sin (const projective_interval< J > & *x*) [friend]

10.242.3.43 template<class J > projective_interval<J> sinh (const projective_interval< J > & *x*) [friend]

10.242.3.44 template<class J > projective_interval<J> sqr (const projective_interval< J > & *x*) [friend]

10.242.3.45 template<class J > projective_interval<J> sqrt (const projective_interval< J > & *x*) [friend]

10.242.3.46 template<class J > projective_interval<J> tan (const projective_interval< J > & *x*) [friend]

10.242.3.47 template<class J > projective_interval<J> tanh (const projective_interval< J > & *x*) [friend]

The documentation for this class was generated from the following files:

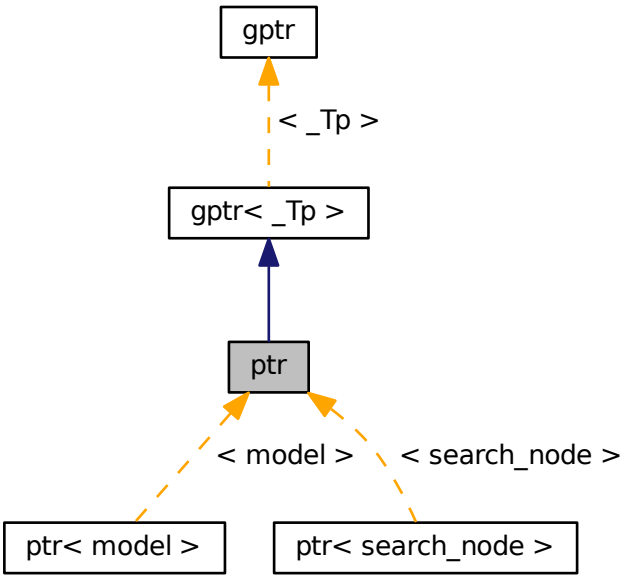
- prointerval.h
- prointerval.hpp

10.243 ptr Class Reference

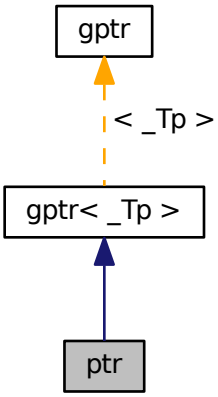
Local global pointer class.

`#include <gptr.h>`

Inheritance diagram for ptr:



Collaboration diagram for ptr:



Public Member Functions

- ptr (_Tp &__p)
- ptr (_Tp *__p)
- ptr (const _Self &__p)
- ~ptr ()
- reference operator* ()
- const_reference operator* () const
- pointer operator-> ()
- const_pointer operator-> () const
- pointer get_local_copy ()
- const_pointer get_local_copy () const
- _Self & operator= (const _Self &__p)
- _Self & operator= (_Tp &__p)

10.243.1 Detailed Description

This class is a specialization of the global pointer class gptr, which points to a **local** object.

10.243.2 Constructor & Destructor Documentation

10.243.2.1 ptr::ptr (_Tp & __p) [inline]

This constructor initializes the ptr with a pointer to the local data structure .

Definition at line 99 of file gptr.h.

10.243.2.2 ptr::ptr (_Tp * __p) [inline]

This constructor initializes the ptr with a pointer to the local data structure pointed at by .

Definition at line 102 of file gptr.h.

10.243.2.3 ptr::ptr (const _Self & __p) [inline]

Standard Copy Constructor

Definition at line 104 of file gptr.h.

10.243.2.4 ptr::~ptr ( ) [inline]

Standard Destructor

Definition at line 107 of file gptr.h.

10.243.3 Member Function Documentation

10.243.3.1 pointer ptr::get_local_copy ( ) [inline]

This method just returns the pointer to the **local** data structur since it need not perform any data retrieval.

Definition at line 121 of file gptr.h.

10.243.3.2 const_pointer ptr::get_local_copy ( ) const [inline]

This method just returns the pointer to the **local** data structur since it need not perform any data retrieval.

Definition at line 124 of file gptr.h.

10.243.3.3 reference ptr::operator* ( ) [inline]

This is the dereferentiation operator.

Definition at line 110 of file gptr.h.

10.243.3.4 const_reference ptr::operator* ( ) const [inline]

This is the const dereferentiation operator.

Definition at line 112 of file gptr.h.

10.243.3.5 pointer ptr::operator-> ( ) [inline]

This is the pointer operator.

Definition at line 115 of file gptr.h.

10.243.3.6 const_pointer ptr::operator-> ( ) const [inline]

This is the const pointer operator.

Definition at line 117 of file gptr.h.

10.243.3.7 _Self& ptr::operator= (const _Self & __p) [inline]

Standard Assignment Operator

Definition at line 127 of file gptr.h.

10.243.3.8 _Self& ptr::operator= (_Tp & __p) [inline]

This operator assigns a ptr object from a data structure ___p.

Definition at line 129 of file gptr.h.

The documentation for this class was generated from the following file:

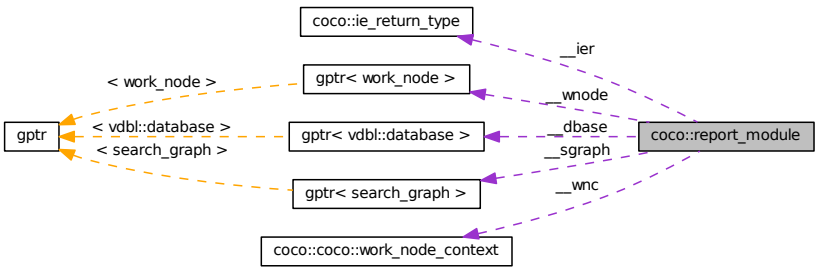
- gptr.h

10.244 coco::report_module Class Reference

Report module base class.

`#include <report_module.h>`

Collaboration diagram for coco::report_module:



Public Member Functions

- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const search_inspector &si, const gptr< vdbl::database > &dbase, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const search_inspector &si, const gptr< vdbl::database > &dbase, const ie_return_type &_ir, bool norestrict)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const search_inspector &si, const gptr< vdbl::database > &dbase, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const search_inspector &si, const gptr< vdbl::database > &dbase, const ie_return_type &_ir, bool norestrict)

- report_module (const std::string &__n, const gptr< search_graph > &sgraph, const search_inspector &si, const gptr< vdbl::database > &dbase, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const gptr< vdbl::database > &dbase, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const gptr< vdbl::database > &dbase, const ie_return_type &_ir, bool norestrict)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const search_inspector &si, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const search_inspector &si, const ie_return_type &_ir, bool norestrict)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const search_inspector &si, const gptr< vdbl::database > &dbase)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const search_inspector &si, const gptr< vdbl::database > &dbase, bool norestrict)
- report_module (const std::string &__n, const gptr< search_graph > &sgraph, const gptr< vdbl::database > &dbase, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< search_graph > &sgraph, const search_inspector &si, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< search_graph > &sgraph, const search_inspector &si, const gptr< vdbl::database > &dbase)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const ie_return_type &_ir, bool norestrict)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const gptr< vdbl::database > &dbase)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const gptr< vdbl::database > &dbase, bool norestrict)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const search_inspector &si)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, const search_inspector &si, bool norestrict)
- report_module (const std::string &__n, const gptr< search_graph > &sgraph, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< search_graph > &sgraph, const gptr< vdbl::database > &dbase)
- report_module (const std::string &__n, const gptr< search_graph > &sgraph, const search_inspector &si)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< search_graph > &sgraph, bool norestrict)
- report_module (const std::string &__n, const search_inspector &si, const gptr< vdbl::database > &dbase, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< vdbl::database > &dbase, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< vdbl::database > &dbase, const ie_return_type &_ir, bool norestrict)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const search_inspector &si, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const search_inspector &si, const ie_return_type &_ir, bool norestrict)

- report_module (const std::string &__n, const gptr< work_node > &wnode, const search_inspector &si, const gptr< vdbl::database > &dbase)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const search_inspector &si, const gptr< vdbl::database > &dbase, bool norestrict)
- report_module (const std::string &__n, const gptr< vdbl::database > &dbase, const ie_return_type &_ir)
- report_module (const std::string &__n, const search_inspector &si, const ie_return_type &_ir)
- report_module (const std::string &__n, const search_inspector &si, const gptr< vdbl::database > &dbase)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const ie_return_type &_ir, bool norestrict)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< vdbl::database > &dbase)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const gptr< vdbl::database > &dbase, bool norestrict)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const search_inspector &si)
- report_module (const std::string &__n, const gptr< work_node > &wnode, const search_inspector &si, bool norestrict)
- report_module (const std::string &__n, const ie_return_type &_ir)
- report_module (const std::string &__n, const gptr< vdbl::database > &dbase)
- report_module (const std::string &__n, const search_inspector &si)
- report_module (const std::string &__n, const gptr< work_node > &wnode)
- report_module (const std::string &__n, const gptr< work_node > &wnode, bool norestrict)
- virtual ~report_module ()
- const model * get_model () const
- virtual void print (const control_data &__c, std::ostream &o=std::cout) const
- const std::string & get_name () const

Protected Attributes

- std::string __name
- const gptr< work_node > * __wnode
- const gptr< search_graph > * __sgraph
- search_inspector * __sgroot
- const gptr< vdbl::database > * __dbase
- const work_node_context * __wnc
- const vdbl::viewdbase * __vdb
- const ie_return_type * __ier

10.244.1 Detailed Description

This is the base class of all COCONUT report modules. A report module produces output. Report modules are, e.g., the AMPL writer and the solution report.

10.244.2 Constructor & Destructor Documentation

10.244.2.1 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const gptr< vdbl::database > & *dbase,* const ie_return_type & *_ir*) [inline]

This is the most general constructor for a report module containing parameters for setting all internal variables. The parameter __n sets __name, wnode sets __wnode, sgraph sets __sgraph, si sets __sgroot, dbase sets __dbase, and _ir sets __ier. This call restricts the search database view to the specialized view for the work node.

Definition at line 102 of file report_module.h.

10.244.2.2 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const gptr< vdbl::database > & *dbase,* const ie_return_type & *_ir,* bool *norestrict*) [inline]

This is the most general constructor for a report module containing parameters for setting all internal variables. The parameter __n sets __name, wnode sets __wnode, si sets __sgroot, dbase sets __dbase, and _ir sets __ier. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 121 of file report_module.h.

10.244.2.3 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const search_inspector & *si,* const gptr< vdbl::database > & *dbase,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, si sets __sgroot, dbase sets __dbase, and _ir sets __ier. This call restricts the search database view to the specialized view for the work node.

Definition at line 138 of file report_module.h.

10.244.2.4 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const search_inspector & *si,* const gptr< vdbl::database > & *dbase,* const ie_return_type & *_ir,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, si sets __sgroot, dbase sets __dbase, and _ir sets __ier. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 155 of file report_module.h.

10.244.2.5 coco::report_module::report_module (const std::string & *__n,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const gptr< vdbl::database > & *dbase,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, si sets __sgroot, dbase sets __dbase, and _ir sets __ier.

Definition at line 170 of file report_module.h.

10.244.2.6 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const gptr< vdbl::database > & *dbase,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `wnode` sets __wnode, `dbase` sets __dbase, and `_ir` sets ___ier. This call restricts the search database view to the specialized view for the work node.

Definition at line 185 of file report_module.h.

10.244.2.7 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const gptr< vdbl::database > & *dbase,* const ie_return_type & *_ir,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `wnode` sets __wnode, `dbase` sets __dbase, and `_ir` sets ___ier. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 201 of file report_module.h.

10.244.2.8 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `wnode` sets __wnode, `si` sets __sgroot, and `_ir` sets ___ier. This call restricts the search database view to the specialized view for the work node.

Definition at line 216 of file report_module.h.

10.244.2.9 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const ie_return_type & *_ir,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `wnode` sets __wnode, `si` sets __sgroot, and `_ir` sets ___ier. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 232 of file report_module.h.

10.244.2.10 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `wnode` sets __wnode, `si` sets __sgroot, and `dbase` sets __dbase. This call restricts the search database view to the specialized view for the work node.

Definition at line 248 of file report_module.h.

10.244.2.11 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const gptr< vdbl::database > & *dbase,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, si sets __sgroot, and dbase sets __dbase. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 264 of file report_module.h.

10.244.2.12 coco::report_module::report_module (const std::string & *__n,* const gptr< search_graph > & *sgraph,* const gptr< vdbl::database > & *dbase,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, dbase sets __dbase, and _ir sets __ier.

Definition at line 278 of file report_module.h.

10.244.2.13 coco::report_module::report_module (const std::string & *__n,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, si sets __sgroot, and _ir sets __ier.

Definition at line 289 of file report_module.h.

10.244.2.14 coco::report_module::report_module (const std::string & *__n,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* const gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, si sets __sgroot, and dbase sets __dbase.

Definition at line 299 of file report_module.h.

10.244.2.15 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and _ir sets __ier. This call restricts the search database view to the specialized view for the work node.

Definition at line 311 of file report_module.h.

10.244.2.16 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const ie_return_type & *_ir,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and _ir sets __ier. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 326 of file report_module.h.

10.244.2.17 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and dbase sets __dbase. This call restricts the search database view to the specialized view for the work node.

Definition at line 341 of file report_module.h.

10.244.2.18 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const gptr< vdbl::database > & *dbase,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and dbase sets __dbase. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 357 of file report_module.h.

10.244.2.19 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const search_inspector & *si*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and si sets __sgroot. This call restricts the search database view to the specialized view for the work node.

Definition at line 372 of file report_module.h.

10.244.2.20 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const gptr< search_graph > & *sgraph,* const search_inspector & *si,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and si sets __sgroot. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 388 of file report_module.h.

10.244.2.21 coco::report_module::report_module (const std::string & *__n,* const gptr< search_graph > & *sgraph,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, and _ir sets __ier.

Definition at line 401 of file report_module.h.

10.244.2.22 coco::report_module::report_module (const std::string & *__n,* const gptr< search_graph > & *sgraph,* const gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, and dbase sets __dbase.

Definition at line 410 of file report_module.h.

10.244.2.23 coco::report_module::report_module (const std::string & __n, const gptr< search_graph > & sgraph, const search_inspector & si) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, and `si` sets __sgroot.

Definition at line 419 of file report_module.h.

10.244.2.24 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const gptr< search_graph > & sgraph) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, and `wnode` sets __wnode. This call restricts the search database view to the specialized view for the work node.

Definition at line 430 of file report_module.h.

10.244.2.25 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const gptr< search_graph > & sgraph, bool norestrict) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, and `wnode` sets __wnode. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 444 of file report_module.h.

10.244.2.26 coco::report_module::report_module (const std::string & __n, const search_inspector & si, const gptr< vdbl::database > & dbase, const ie_return_type & _ir) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `si` sets __sgroot, `dbase` sets __dbase, and `_ir` sets ___ier.

Definition at line 457 of file report_module.h.

10.244.2.27 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const gptr< vdbl::database > & dbase, const ie_return_type & _ir) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `wnode` sets __wnode, `dbase` sets __dbase, and `_ir` sets ___ier. This call restricts the search database view to the specialized view for the work node.

Definition at line 471 of file report_module.h.

10.244.2.28 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const gptr< vdbl::database > & dbase, const ie_return_type & _ir, bool norestrict) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `wnode` sets __wnode, `dbase` sets __dbase, and `_ir` sets ___ier. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 486 of file report_module.h.

10.244.2.29 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const search_inspector & si, const ie_return_type & _ir) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, si sets __sgroot, and _ir sets __ier. This call restricts the search database view to the specialized view for the work node.

Definition at line 500 of file report_module.h.

10.244.2.30 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const search_inspector & si, const ie_return_type & _ir, bool norestrict) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, si sets __sgroot, and _ir sets __ier. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 515 of file report_module.h.

10.244.2.31 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const search_inspector & si, const gptr< vdbl::database > & dbase) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, si sets __sgroot, and dbase sets __dbase. This call restricts the search database view to the specialized view for the work node.

Definition at line 530 of file report_module.h.

10.244.2.32 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const search_inspector & si, const gptr< vdbl::database > & dbase, bool norestrict) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, si sets __sgroot, and dbase sets __dbase. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 545 of file report_module.h.

10.244.2.33 coco::report_module::report_module (const std::string & __n, const gptr< vdbl::database > & dbase, const ie_return_type & _ir) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, dbase sets __dbase, and _ir sets __ier.

Definition at line 558 of file report_module.h.

10.244.2.34 coco::report_module::report_module (const std::string & __n, const search_inspector & si, const ie_return_type & _ir) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, si sets __sgroot, and _ir sets __ier.

Definition at line 568 of file report_module.h.

10.244.2.35 coco::report_module::report_module (const std::string & __n, const search_inspector & si, const gptr< vdbl::database > & dbase) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, si sets __sgroot, and dbase sets __dbase.

Definition at line 578 of file report_module.h.

10.244.2.36 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const ie_return_type & _ir) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and _ir sets __ier. This call restricts the search database view to the specialized view for the work node.

Definition at line 590 of file report_module.h.

10.244.2.37 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const ie_return_type & _ir, bool norestrict) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and _ir sets __ier. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 604 of file report_module.h.

10.244.2.38 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const gptr< vdbl::database > & dbase) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and dbase sets __dbase. This call restricts the search database view to the specialized view for the work node.

Definition at line 617 of file report_module.h.

10.244.2.39 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const gptr< vdbl::database > & dbase, bool norestrict) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and dbase sets __dbase. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 631 of file report_module.h.

10.244.2.40 coco::report_module::report_module (const std::string & __n, const gptr< work_node > & wnode, const search_inspector & si) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter __n sets __name, wnode sets __wnode, and si sets __sgroot. This call restricts the search database view to the specialized view for the work node.

Definition at line 644 of file report_module.h.

10.244.2.41 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* const search_inspector & *si,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, `wnode` sets __wnode, and `si` sets __sgroot. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 659 of file report_module.h.

10.244.2.42 coco::report_module::report_module (const std::string & *__n,* const ie_return_type & *_ir*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, and `_ir` sets ___ier.

Definition at line 671 of file report_module.h.

10.244.2.43 coco::report_module::report_module (const std::string & *__n,* const gptr< vdbl::database > & *dbase*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, and `dbase` sets __dbase.

Definition at line 679 of file report_module.h.

10.244.2.44 coco::report_module::report_module (const std::string & *__n,* const search_inspector & *si*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, and `si` sets __sgroot.

Definition at line 687 of file report_module.h.

10.244.2.45 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, and `wnode` sets __wnode. This call restricts the search database view to the specialized view for the work node.

Definition at line 697 of file report_module.h.

10.244.2.46 coco::report_module::report_module (const std::string & *__n,* const gptr< work_node > & *wnode,* bool *norestrict*) [inline]

This is a constructor for a report module containing parameters for setting some internal variables. The parameter ___n sets __name, and `wnode` sets __wnode. This call does not restrict the search database view to the specialized view for the work node, i.e. all the search database entries can be accessed.

Definition at line 709 of file report_module.h.

10.244.2.47 virtual coco::report_module::~report_module ( ) [inline, virtual]

Standard Destructor

Definition at line 718 of file report_module.h.

10.244.3 Member Function Documentation

10.244.3.1 const model* coco::report_module::get_model ( ) const [inline]

A call to this method returns a pointer to the model of the work node.

Definition at line 722 of file report_module.h.

10.244.3.2 const std::string& coco::report_module::get_name ( ) const [inline]

The get_name method returns the identifier string of the report module.

Definition at line 733 of file report_module.h.

10.244.3.3 virtual void coco::report_module::print (const control_data & *__c*, std::ostream & *o* = std::cout) const [inline, virtual]

This method is the main method of a report module. It is supposed to write the output to the ostream o. This method is overloaded by the various subclasses. Service information and parameters are provided via the control_data structure __c.

Definition at line 729 of file report_module.h.

10.244.4 Member Data Documentation

10.244.4.1 const gptr<vdbl::database>* coco::report_module::__dbase [protected]

This variable contains a global pointer to the search database.

Definition at line 86 of file report_module.h.

10.244.4.2 const ie_return_type* coco::report_module::__ier [protected]

This is a pointer to an ie_return_type.

Definition at line 93 of file report_module.h.

10.244.4.3 std::string coco::report_module::__name [protected]

This is the identifier string for a management module.

Definition at line 78 of file report_module.h.

10.244.4.4 const gptr<search_graph>* coco::report_module::__sgraph [protected]

This variable contains a global pointer to the work node.

Definition at line 82 of file report_module.h.

10.244.4.5 search_inspector* coco::report_module::__sgroot [protected]

This is a search inspector pointing to the root of the search graph.

Definition at line 84 of file report_module.h.

10.244.4.6 const vdbl::viewdbase* coco::report_module::__vdb [protected]

This is a view to the search database.

Definition at line 91 of file report_module.h.

10.244.4.7 const work_node_context* coco::report_module::__wnc [protected]

This is the work_node_context for extracting information from the search database.

Definition at line 89 of file report_module.h.

10.244.4.8 const gptr<work_node>* coco::report_module::__wnode [protected]

This variable contains a global pointer to the work node.

Definition at line 80 of file report_module.h.

The documentation for this class was generated from the following file:

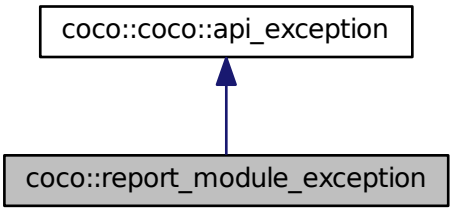
- report_module.h

10.245 coco::report_module_exception Class Reference

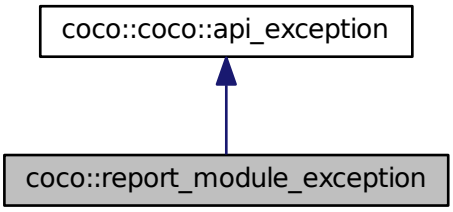
Report module exception class.

`#include <report_module.h>`

Inheritance diagram for coco::report_module_exception:



Collaboration diagram for coco::report_module_exception:



Public Member Functions

- report_module_exception (const std::string &msg)
- report_module_exception (const char *msg)
- virtual ~report_module_exception () throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual char const * what () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()
- virtual std::string message () const throw ()

10.245.1 Detailed Description

This is the exceptions class thrown by report module subclasses. Every properly coded COCONUT report module should only throw exceptions of this type.

10.245.2 Constructor & Destructor Documentation

10.245.2.1 coco::report_module_exception::report_module_exception (const std::string & *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 58 of file report_module.h.

10.245.2.2 coco::report_module_exception::report_module_exception (const char * *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 61 of file report_module.h.

10.245.2.3 virtual coco::report_module_exception::~report_module_exception ( ) throw () [inline, virtual]

Standard Destructor

Definition at line 65 of file report_module.h.

10.245.3 Member Function Documentation

10.245.3.1 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file expression.h.

10.245.3.2 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.245.3.3 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.245.3.4 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.245.3.5 virtual std::string coco::coco::api_exception::message ( ) const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file search_graph.cc.

10.245.3.6 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.245.3.7 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.245.3.8 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file expression.h.

10.245.3.9 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.245.3.10 virtual api_exception_type coco::coco::api_exception::type ( ) const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file search_graph.cc.

10.245.3.11 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.245.3.12 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.245.3.13 const char * coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

Definition at line 57 of file api_exception.cc.

10.245.3.14 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.245.3.15 virtual const char* coco::coco::api_exception::type_str ( ) const throw () [virtual, inherited]

This method returns the exception type as C-string.

10.245.3.16 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.245.3.17 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.245.3.18 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file expression.h.

10.245.3.19 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

10.245.3.20 virtual char const* coco::coco::api_exception::what ( ) const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file search_graph.cc.

The documentation for this class was generated from the following file:

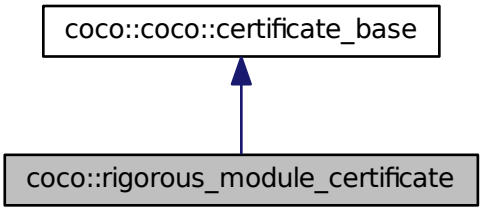
- report_module.h

10.246 coco::rigorous_module_certificate Class Reference

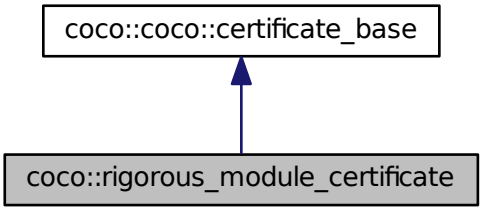
The certificate for deltas computed by rigorous inference engines.

`#include <api_cert.h>`

Inheritance diagram for coco::rigorous_module_certificate:



Collaboration diagram for coco::rigorous_module_certificate:



Public Member Functions

- rigorous_module_certificate (const std::string &_name)
- rigorous_module_certificate (const rigorous_module_certificate &__c)
- virtual ~rigorous_module_certificate ()
- rigorous_module_certificate * new_copy () const
- void destroy_copy (certificate_base *__c) const
- bool verify (const work_node &_x) const
- std::string get_contents () const
- bool operator== (const certificate_base &_c) const
- bool operator!= (const certificate_base &_c) const
- bool operator== (const rigorous_module_certificate &_c) const
- bool operator!= (const rigorous_module_certificate &_c) const

- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)

Protected Attributes

- std::string _contents

10.246.1 Detailed Description

The rigorous module certificate for inference engines, which do not produce certificates but rather always work in a rigorous mode!

10.246.2 Constructor & Destructor Documentation

10.246.2.1 coco::rigorous_module_certificate::rigorous_module_certificate (const std::string & *_name*) [inline]

Constructor with parameter _name specifying the module_name

Definition at line 270 of file api_cert.h.

10.246.2.2 coco::rigorous_module_certificate::rigorous_module_certificate (const rigorous_module_certificate & *__c*) [inline]

Standard Copy Constructor

Definition at line 273 of file api_cert.h.

10.246.2.3 virtual coco::rigorous_module_certificate::~rigorous_module_certificate ( ) [inline, virtual]

Standard Destructor

Definition at line 282 of file api_cert.h.

10.246.3 Member Function Documentation

10.246.3.1 void coco::rigorous_module_certificate::destroy_copy (certificate_base * *__c*) const [inline]

Clone Destructor

Definition at line 293 of file api_cert.h.

10.246.3.2 std::string coco::rigorous_module_certificate::get_contents ( ) const [inline, virtual]

Retrieve the contents information (the certificate type) for this certificate. This method includes the module-_name in the return value.

Reimplemented from coco::coco::certificate_base.

Definition at line 303 of file api_cert.h.

10.246.3.3 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.246.3.4 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.246.3.5 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.246.3.6 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.246.3.7 rigorous_module_certificate* coco::rigorous_module_certificate::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::certificate_base.

Definition at line 285 of file api_cert.h.

10.246.3.8 bool coco::rigorous_module_certificate::operator!= (const certificate_base & *_c*) const [inline]

Definition at line 310 of file api_cert.h.

10.246.3.9 bool coco::rigorous_module_certificate::operator!= (const rigorous_module_certificate & *_c*) const [inline]

Definition at line 317 of file api_cert.h.

10.246.3.10 bool coco::rigorous_module_certificate::operator== (const certificate_base & _c) const [inline]

Comparison operators

Definition at line 307 of file api_cert.h.

10.246.3.11 bool coco::rigorous_module_certificate::operator== (const rigorous_module_certificate & _c) const [inline]

Comparison operators

Definition at line 315 of file api_cert.h.

10.246.3.12 bool coco::rigorous_module_certificate::verify (const work_node & _x) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. In this class: `true`.

Reimplemented from coco::coco::certificate_base.

Definition at line 298 of file api_cert.h.

10.246.4 Member Data Documentation

10.246.4.1 std::string coco::coco::certificate_base::_contents [protected, inherited]

The contents (descriptive string, type) of this certificate

Definition at line 128 of file search_graph.cc.

The documentation for this class was generated from the following file:

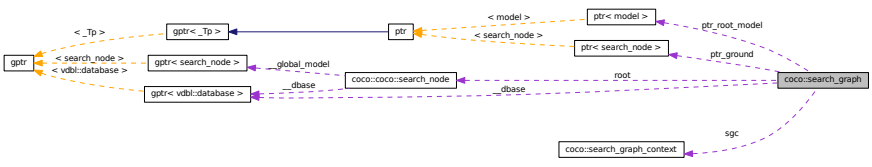
- api_cert.h

10.247 coco::search_graph Class Reference

The search graph.

`#include <search_graph.h>`

Collaboration diagram for coco::search_graph:



Public Member Functions

- search_graph (model &root_model, gptr< vdbl::database > &_db)
- search_graph (model &root_model, gptr< vdbl::database > &_db, const std::vector< annotation > &_a)
- ~search_graph ()
- search_node_id get_node_id ()
- search_graph_context * get_search_graph_context ()
- const search_graph_context * get_search_graph_context () const
- vdbl::viewdbase & get_viewdbase ()
- const vdbl::viewdbase & get_viewdbase () const
- inference_module_cache & get_inf_mod_cache ()
- bool register_hook (const work_node_comp_hook &wnch)
- bool unregister_hook (const std::string &name)
- void apply_hooks (work_node &w, const search_node_id &id, vdbl::rowid &rid, std::list< delta_id > &add_dis)

Focus and Inspector Management

The methods described in this section are used for managing the search focusses and search inspectors.

- bool possible_focus (const search_inspector &_si, const search_focus &not_this) const
- bool possible_focus (const search_inspector &_si, const std::list< search_focus >::const_iterator *not_this=NULL) const
- search_focus & new_focus (const search_inspector &_si)
- const search_focus & get_focus (const search_inspector &_si, bool &isNew)
- void destroy_focus (const search_focus &_sf)
- search_focus & set_focus (search_focus &_fc, const search_inspector &_si)
- search_inspector & new_inspector (const search_inspector &inspector_to_add)
- search_inspector & new_inspector ()
- void destroy_inspector (const search_inspector &inspector_to_destroy)
- search_inspector child (search_inspector &n, unsigned int i)
- search_inspector parent (search_inspector &n, unsigned int i)
- search_inspector get (search_node_id id) const

Methods for search node handling

The methods described in this section are used for managing search nodes on the search graph.

- work_node extract (const search_inspector &where) const
- work_node extract (const search_focus &where) const
- void extract (work_node &wnode, const search_inspector &where) const
- void extract (work_node &wnode, const search_focus &where) const
- search_focus & update (work_node &wnode, search_focus &foc, const search_inspector &where)
- search_focus & update_and_delete (work_node &wnode, search_focus &foc, const search_inspector &where, bool just_this, bool relaxation_kills)
- search_inspector & insert (const search_focus &where, const search_node &what)
- search_inspector & insert (const search_focus &where, search_node *what)
- search_inspector & replace (search_focus &where, const search_node &what)
- bool add_deltas (const search_focus &where, const std::vector< delta_id > &del_vec)
- void remove (search_focus &_sf)
- void remove_upwards (search_focus &_sf, bool relaxation_kills)
- search_focus & promote (search_focus &_sf)

Public Attributes

- std::list< walker > focus
- std::list< const_walker > inspector
- search_node * root
- search_inspector inspector_for_root
- ptr< search_node > ptr_ground
- ptr< model > ptr_root_model
- std::list< search_node * > search_nodes_to_deallocate
- std::list< delta_id > db_deltas_to_remove
- gptr< vdbl::database > * __dbase
- vdbl::userid __dbuser
- search_graph_context * sgc
- vdbl::viewdbase __vdbf

10.247.1 Detailed Description

This is the class holding the search graph. It is basically a DAG of search_node pointers pointing to either delta_node or full_node entries. There can be more than one search graph in the same strategy.

10.247.2 Constructor & Destructor Documentation

10.247.2.1 coco::search_graph::search_graph (model & *root_model*, gptr< vdbl::database > & *_db*) [inline]

Constructor, which generates a new search graph from an original model `root_model`. The search database `_db` must already be created and is likewise passed to the constructor.

Definition at line 35 of file search_graph.hpp.

10.247.2.2 coco::search_graph::search_graph (model & *root_model*, gptr< vdbl::database > & *_db*, const std::vector< annotation > & *_a*) [inline]

Constructor, which generates a new search graph from an original model `root_model` together with initial model annotations `_a`. The search database `_db` must already be created and is likewise passed to the constructor.

Definition at line 54 of file search_graph.hpp.

10.247.2.3 coco::search_graph::~search_graph ( ) [inline]

Destructor

Definition at line 74 of file search_graph.hpp.

10.247.3 Member Function Documentation

10.247.3.1 bool coco::search_graph::add_deltas (const search_focus & *where*, const std::vector< delta_id > & *del_vec*)

The add_deltas method adds the additional deltas in `del_vec` to the search_node at position `where`. The method returns `true` if adding the deltas was successful, i.e. if the node at `where` is a delta_node.

Definition at line 881 of file search_graph.cc.

10.247.3.2 void coco::search_graph::apply_hooks (work_node & *w*, const search_node_id & *id*, vdbl::rowid & *rid*, std::list< delta_id > & *add_dis*)

The apply_hooks method is called right after a split for all the generated nodes right before they are stored in the search graph. It calls all registered work node computation hooks in order to store information in the "search info" table.

Definition at line 951 of file search_graph.cc.

10.247.3.3 search_graph::search_inspector coco::search_graph::child (search_inspector & *n*, unsigned int *i*) [inline]

The child method creates a new search inspector pointing to the `i`th child of the search node at `n`.

Definition at line 102 of file search_graph.hpp.

10.247.3.4 void coco::search_graph::destroy_focus (const search_focus & *_sf*)

This method destroys the search focus `_sf`.

Definition at line 184 of file search_graph.cc.

10.247.3.5 void coco::search_graph::destroy_inspector (const search_inspector & *inspector_to_destroy*)

Calling the destroy_inspector with parameter `inspector_to_destroy` destroys the search_inspector passed.

Definition at line 225 of file search_graph.cc.

10.247.3.6 work_node coco::search_graph::extract (const search_inspector & *where*) const

This method extracts a work_node from the search_node at position `where`.

Definition at line 404 of file search_graph.cc.

10.247.3.7 work_node coco::search_graph::extract (const search_focus & *where*) const

This method extracts a work_node from the search_node at position `where`.

Definition at line 434 of file search_graph.cc.

10.247.3.8 void coco::search_graph::extract (work_node & *wnode*, const search_inspector & *where*) const

This method extracts a work_node from the search_node at position `where` and puts it into `wnode`.

Definition at line 382 of file search_graph.cc.

10.247.3.9 void coco::search_graph::extract (work_node & *wnode,* const search_focus & *where*) const

This method extracts a work_node from the search_node at position `where` and puts it into `wnode`.

Definition at line 393 of file search_graph.cc.

10.247.3.10 search_graph::search_inspector coco::search_graph::get (search_node_id *id*) const [inline]

A call to the get method retrieves the position of the search node with search_node_id `id` in the search graph.

Definition at line 120 of file search_graph.hpp.

10.247.3.11 const search_focus & coco::search_graph::get_focus (const search_inspector & *_si,* bool & *isNew*)

This method creates a new focus to `_si` or returns an existing one if present. `isNew` is set to true if the focus was newly created and has to be destroyed.

Definition at line 163 of file search_graph.cc.

10.247.3.12 inference_module_cache& coco::search_graph::get_inf_mod_cache ( ) [inline]

This method returns a reference to the inference module cache.

Definition at line 169 of file search_graph.h.

10.247.3.13 search_node_id coco::search_graph::get_node_id ( ) [inline]

This method returns the next available search_node_id.

Definition at line 154 of file search_graph.h.

10.247.3.14 search_graph_context* coco::search_graph::get_search_graph_context ( ) [inline]

This method returns the actual search graph context

Definition at line 157 of file search_graph.h.

10.247.3.15 const search_graph_context* coco::search_graph::get_search_graph_context ( ) const [inline]

This method returns the actual search graph context

Definition at line 160 of file search_graph.h.

10.247.3.16 vdbl::viewdbase& coco::search_graph::get_viewdbase ( ) [inline]

This method returns the actual view database.

Definition at line 163 of file search_graph.h.

10.247.3.17 const vdbl::viewdbase& coco::search_graph::get_viewdbase ( ) const [inline]

This method returns the actual view database.

Definition at line 166 of file search_graph.h.

10.247.3.18 search_inspector & coco::search_graph::insert (const search_focus & *where,* const search_node & *what*)

A call to the insert method stores the search_node `what` as a child of the search_node at position `where` in the search graph.

Definition at line 495 of file search_graph.cc.

10.247.3.19 search_inspector & coco::search_graph::insert (const search_focus & *where,* search_node * *what*)

A call to the insert method stores the search_node `what` as a child of the search_node at position `where` in the search graph. The pointer `what` is marked for deletion on destruction of the search graph.

Definition at line 510 of file search_graph.cc.

10.247.3.20 search_focus & coco::search_graph::new_focus (const search_inspector & *_si*)

This method creates a new focus and sets it on the node to which the search_inspector `_si` points.

Definition at line 151 of file search_graph.cc.

10.247.3.21 search_inspector & coco::search_graph::new_inspector (const search_inspector & *inspector_to_add*)

This method creates a new search_inspector pointing to `inspector_to_add`, unless there is already such a search inspector.

Definition at line 214 of file search_graph.cc.

10.247.3.22 search_inspector& coco::search_graph::new_inspector ( ) [inline]

This method creates a new search_inspector pointing to the ground node of the search graph.

Definition at line 202 of file search_graph.h.

10.247.3.23 search_graph::search_inspector coco::search_graph::parent (search_inspector & *n,* unsigned int *i*) [inline]

The parent method creates a new search inspector pointing to the `i`th parent of the search node at `n`.

Definition at line 111 of file search_graph.hpp.

10.247.3.24 bool coco::search_graph::possible_focus (const search_inspector & *_si,* const search_focus & *not_this*) const

This method checks whether a new focus can be created on the node to which the search_inspector `_si` points.

Definition at line 143 of file search_graph.cc.

10.247.3.25 bool coco::search_graph::possible_focus (const search_inspector & *_si,* const std::list< search_focus >::const_iterator * *not_this* = NULL) const

This method checks whether a new focus can be created on the node to which the search_inspector _si points.

Definition at line 107 of file search_graph.cc.

10.247.3.26 search_focus & coco::search_graph::promote (search_focus & *_sf*)

This method promotes a reduction node to its parent position if and only if the parent of the node is unique. The search_focus _sf points to the node to be promoted.

Definition at line 695 of file search_graph.cc.

10.247.3.27 bool coco::search_graph::register_hook (const work_node_comp_hook & *wnch*)

This method registers the new work node computation hook wnch.

Definition at line 897 of file search_graph.cc.

10.247.3.28 void coco::search_graph::remove (search_focus & *_sf*)

A call to remove deletes the search_node at position _sf from the search graph.

Definition at line 681 of file search_graph.cc.

10.247.3.29 void coco::search_graph::remove_upwards (search_focus & *_sf,* bool *relaxation_kills*)

A call to remove_upwards deletes the search_node at position _sf from the search graph and then recursively all its parents as long as the parents have either search_node_relation snr_glue or snr_reduction. The parent is also deleted if the search_node_relation is snr_relaxation and relaxation_kills is true.

Definition at line 830 of file search_graph.cc.

10.247.3.30 search_inspector & coco::search_graph::replace (search_focus & *where,* const search_node & *what*)

The replace method replaces the search_node at position where by the search_node what.

Definition at line 527 of file search_graph.cc.

10.247.3.31 search_focus & coco::search_graph::set_focus (search_focus & *_fc,* const search_inspector & *_si*)

A call to this method moves the search_focus _fc from its current position to the search node to which _si points.

Definition at line 193 of file search_graph.cc.

10.247.3.32 bool coco::search_graph::unregister_hook (const std::string & *name*)

This method unregisters the work node computation hook with name name.

Definition at line 931 of file search_graph.cc.

10.247.3.33 search_focus & coco::search_graph::update (work_node & *wnode,* search_focus & *foc,* const search_inspector & *where*)

This method updates the search focus `foc` to the new position `where` and updates the work_node `wnode` accordingly.

Definition at line 484 of file search_graph.cc.

10.247.3.34 search_focus & coco::search_graph::update_and_delete (work_node & *wnode,* search_focus & *foc,* const search_inspector & *where,* bool *just_this,* bool *relaxation_kills*)

This method updates the search focus `foc` to the new position `where` and updates the work_node `wnode` accordingly. Then it removes the node at the old focus and if `just_this` is false it recursively deletes all its parents as long as the parents have either search_node_relation snr_glue or snr_reduction. The parent is also deleted if the search_node_relation is snr_relaxation and relaxation_kills is `true`.

Definition at line 464 of file search_graph.cc.

10.247.4 Member Data Documentation

10.247.4.1 gptr<vdbl::database>* coco::search_graph::__dbase

The __dbase variable holds a pointer to the global pointer to the search database.

Definition at line 89 of file search_graph.h.

10.247.4.2 vdbl::userid coco::search_graph::__dbuser

The __dbuser variable holds the database user id for accessing the search database.

Definition at line 92 of file search_graph.h.

10.247.4.3 vdbl::viewdbase coco::search_graph::__vdbf

This is a full window view to the search database to be used by the inference engines and report modules.

Definition at line 98 of file search_graph.h.

10.247.4.4 std::list<delta_id> coco::search_graph::db_deltas_to_remove

This list contains all delta_id entries which have to be removed from the search database during search_node deletion.

Definition at line 86 of file search_graph.h.

10.247.4.5 std::list<walker> coco::search_graph::focus

This variable holds the list of currently active search focusses.

Definition at line 68 of file search_graph.h.

10.247.4.6 std::list<const_walker> coco::search_graph::inspector

This variable holds the list of currently registered search inspectors.

Definition at line 70 of file search_graph.h.

10.247.4.7 search_inspector coco::search_graph::inspector_for_root

This member holds a search_inspector pointing to the root node.

Definition at line 75 of file search_graph.h.

10.247.4.8 ptr<search_node> coco::search_graph::ptr_ground

This member points to the ground of the search graph.

Definition at line 77 of file search_graph.h.

10.247.4.9 ptr<model> coco::search_graph::ptr_root_model

This member contains a generalized local pointer pointing to the root model (the original model).

Definition at line 80 of file search_graph.h.

10.247.4.10 search_node* coco::search_graph::root

This member points to the root of the search graph.

Definition at line 73 of file search_graph.h.

10.247.4.11 std::list<search_node*> coco::search_graph::search_nodes_to_deallocate

This list contains all search nodes whose memory has to be freed during search_node deletion.

Definition at line 83 of file search_graph.h.

10.247.4.12 search_graph_context* coco::search_graph::sgc

This is the search graph context corresponding to the current work node.

Definition at line 95 of file search_graph.h.

The documentation for this class was generated from the following files:

- search_graph.h
- search_graph.cc
- search_graph.hpp

10.248 coco::search_graph_context Class Reference

An evaluation context when retrieving from the search database.

```
#include <sgraphctx.h>
```

Public Member Functions

- search_graph_context ()

- search_graph_context (const gptr< search_graph > *_i)
- search_graph_context (search_graph &_i)
- search_graph_context (const search_graph_context &_s)
- virtual ~search_graph_context ()
- search_graph_context & operator= (const search_graph_context &_s)
- const gptr< search_graph > * sg () const

10.248.1 Detailed Description

This class provides the evaluation context when a view to the search database is created for a graph analyzer. This mainly influences the stored procedures (vdbl::method).

10.248.2 Constructor & Destructor Documentation

10.248.2.1 coco::search_graph_context::search_graph_context ( ) [inline]

Standard Constructor

Definition at line 54 of file sgraphctx.h.

10.248.2.2 coco::search_graph_context::search_graph_context (const gptr< search_graph > * _i) [inline]

Constructor which initializes the search_graph to _i.

Definition at line 56 of file sgraphctx.h.

10.248.2.3 coco::search_graph_context::search_graph_context (search_graph & _i) [inline]

Constructor which initializes the search_graph to _i.

Definition at line 58 of file sgraphctx.h.

10.248.2.4 coco::search_graph_context::search_graph_context (const search_graph_context & _s) [inline]

Standard Copy Constructor

Definition at line 61 of file sgraphctx.h.

10.248.2.5 virtual coco::search_graph_context::~search_graph_context ( ) [inline, virtual]

Standard Destructor

Definition at line 64 of file sgraphctx.h.

10.248.3 Member Function Documentation

10.248.3.1 search_graph_context& coco::search_graph_context::operator= (const search_graph_context & _s) [inline]

Standard Assignment Operator

Definition at line 67 of file sgraphctx.h.

10.248.3.2 const gptr<search_graph>* coco::search_graph_context::sg ( ) const [inline]

This method returns a global pointer to the search_graph of this context.

Definition at line 72 of file sgraphctx.h.

The documentation for this class was generated from the following file:

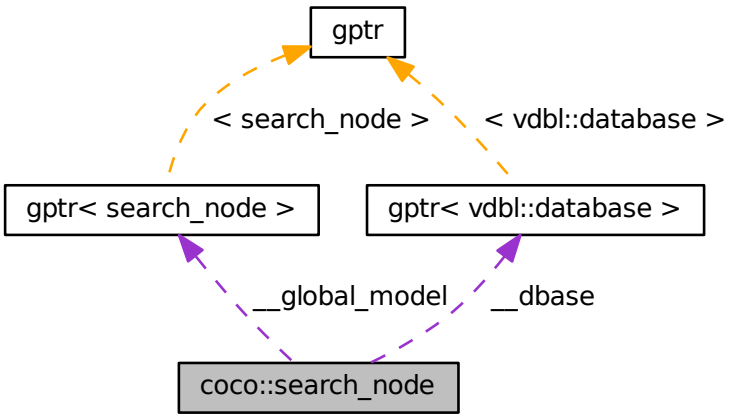
- sgraphctx.h

10.249 coco::search_node Class Reference

Base type of the nodes in the search graph.

`#include <search_node.h>`

Collaboration diagram for coco::search_node:



Public Member Functions

- virtual bool is_delta () const
- search_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< search_node > &_gm, gptr< vdbl::database > &_db, search_node_relation __snr=snr_reduction)
- search_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< search_node > *_gm, gptr< vdbl::database > &_db, search_node_relation __snr=snr_reduction)
- search_node (const search_node &__sn)
- search_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< vdbl::database > &_db, search_node_relation __snr=snr_root)

- virtual ~search_node ()
- search_node & operator= (const search_node &__w)
- vdbl::userid get_dbuserid () const
- gptr< search_node > * global_model () const
- gptr< vdbl::database > * database () const
- search_node_id get_id () const
- void set_id (const search_node_id &i)
- vdbl::rowid get_rowid () const
- void set_rowid (const vdbl::rowid &i)
- void keep (const annotation &_an)
- void keep (const std::vector< annotation > &_anv)
- void unkeep (const annotation &_an)
- void unkeep (const std::vector< annotation > &_anv)

Protected Member Functions

- search_node_relation parent_relation () const

Protected Attributes

- gptr< search_node > * __global_model
- gptr< vdbl::database > * __dbase
- vdbl::userid _dbuser
- search_node_relation _snr
- search_node_id _id
- std::vector< annotation > _keep
- vdbl::rowid _rid

Friends

- class search_graph

10.249.1 Detailed Description

This is the base type of the nodes stored in the search graph and analyzed by the inference engines. Its subclasses delta_node and full_node are used in the search graph. The special subclass work_node of full_node is used to present the models to be analyzed to the inference engines.

10.249.2 Constructor & Destructor Documentation

10.249.2.1 coco::search_node::search_node (const search_node_id & *_i*, const vdbl::userid & *_dui*, gptr< search_node > & *_gm*, gptr< vdbl::database > & *_db*, search_node_relation *__snr* = snr_reduction)

This constructor generates a new search node (a reduction node by default) with search_node_id `i` and search_node_relation `__snr`. The parameters `_gm`, `_db`, and `_dui` initialize the global model, search database, and the database user id, respectively.

10.249.2.2 coco::search_node::search_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< search_node > * _gm, gptr< vdbl::database > & _db, search_node_relation __snr = snr_reduction)

This constructor generates a new search node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively.

10.249.2.3 coco::search_node::search_node (const search_node & __sn)

Standard Copy Constructor

10.249.2.4 coco::search_node::search_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< vdbl::database > & _db, search_node_relation __snr = snr_root)

This constructor generates a new search node (the graph's root node by default) with search_node_id i and search_node_relation ___snr. The parameters _db, and _dui initialize the search database and the database user id, respectively. The global model pointer is set to NULL. The constructor can be used for standalone nodes, as well.

10.249.2.5 virtual coco::search_node::~search_node ( ) [virtual]

Standard Destructor

10.249.3 Member Function Documentation

10.249.3.1 gptr<vdbl::database>* coco::search_node::database ( ) const [inline]

This is the accessor method for the search database.

Definition at line 157 of file search_node.h.

10.249.3.2 vdbl::userid coco::search_node::get_dbuserid ( ) const [inline]

This is the accessor method for the database user id.

Definition at line 151 of file search_node.h.

10.249.3.3 search_node_id coco::search_node::get_id ( ) const [inline]

This is the accessor method for the search node id of this node.

Definition at line 160 of file search_node.h.

10.249.3.4 vdbl::rowid coco::search_node::get_rowid ( ) const [inline]

This is the accessor method for the row id of this node.

Definition at line 166 of file search_node.h.

10.249.3.5 gptr<search_node>* coco::search_node::global_model ( ) const [inline]

This is the accessor method for the global model.

Definition at line 154 of file search_node.h.

10.249.3.6 virtual bool coco::search_node::is_delta ( ) const [inline, virtual]

This method is called to determine whether a search node stores only deltas (delta_node) or a full model (full_node).

See also

delta.

Definition at line 115 of file search_node.h.

10.249.3.7 void coco::search_node::keep (const annotation & _an) [inline]

A call to this method informs the search node to be the keeper of the annotation _an.

Definition at line 173 of file search_node.h.

10.249.3.8 void coco::search_node::keep (const std::vector< annotation > & _anv) [inline]

A call to this method informs the search node to be the keeper of all the annotations in _anv.

Definition at line 177 of file search_node.h.

10.249.3.9 search_node& coco::search_node::operator= (const search_node & __w)

Standard Assignment Operator

10.249.3.10 search_node_relation coco::search_node::parent_relation ( ) const [inline, protected]

A call to this method returns the relation of this node to its parent in the search graph.

Definition at line 110 of file search_node.h.

10.249.3.11 void coco::search_node::set_id (const search_node_id & i) [inline]

This is the set method for the search node id of this node.

Definition at line 163 of file search_node.h.

10.249.3.12 void coco::search_node::set_rowid (const vdbl::rowid & i) [inline]

This is the set method for the row id of this node.

Definition at line 169 of file search_node.h.

10.249.3.13 void coco::search_node::unkeep (const annotation & _an)

A call to this method informs the search node to no longer be the keeper of the annotation _an.

10.249.3.14 void coco::search_node::unkeep (const std::vector< annotation > & _anv)

A call to this method informs the search node to no longer be the keeper of all the annotations in _anv.

10.249.4 Friends And Related Function Documentation

10.249.4.1 friend class search_graph [friend]

Definition at line 188 of file search_node.h.

10.249.5 Member Data Documentation

10.249.5.1 gptr<vdbl::database>* coco::search_node::__dbase [protected]

This is a pointer to a global pointer to the search database associated to this search graph.

Definition at line 89 of file search_node.h.

10.249.5.2 gptr<search_node>* coco::search_node::__global_model [protected]

This is a pointer to a global pointer to the top of the search graph. It is NULL for a standalone node or the top of the graph.

Definition at line 86 of file search_node.h.

10.249.5.3 vdbl::userid coco::search_node::_dbuser [protected]

This member stores the database user id under which this search is performed. For the in-memory database this is irrelevant.

Definition at line 92 of file search_node.h.

10.249.5.4 search_node_id coco::search_node::_id [protected]

This is the unique identifier of this search node.

Definition at line 97 of file search_node.h.

10.249.5.5 std::vector<annotation> coco::search_node::_keep [protected]

The _keep member is a list of all those annotations which are kept by this search node. If the node is destroyed all kept annotations are removed from the search database.

Definition at line 101 of file search_node.h.

10.249.5.6 vdbl::rowid coco::search_node::_rid [protected]

This row id specifies the row in the search info table where the hook information for this search_node is stored

Definition at line 105 of file search_node.h.

10.249.5.7 search_node_relation coco::search_node::_snr [protected]

This member specifies the relation of the search node to its parent(s) in the search graph.

Definition at line 95 of file search_node.h.

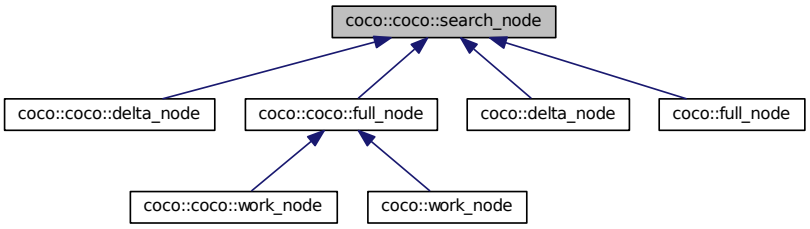
The documentation for this class was generated from the following file:

- search_node.h

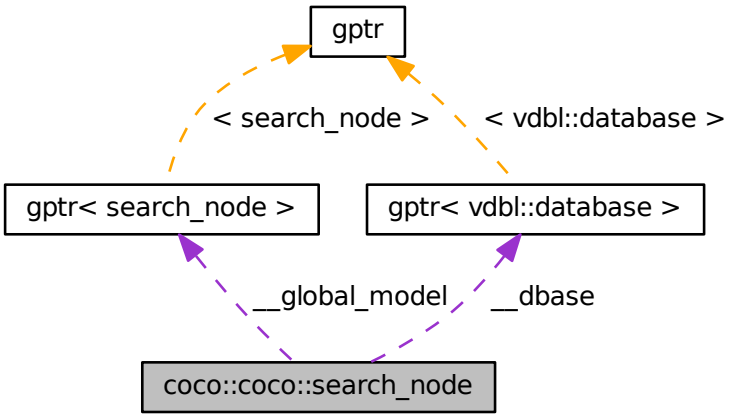
10.250 coco::coco::search_node Class Reference

Base type of the nodes in the search graph.

Inheritance diagram for coco::coco::search_node:



Collaboration diagram for coco::coco::search_node:



Public Member Functions

- virtual bool is_delta () const
- search_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< search_node > &_gm, gptr< vdbl::database > &_db, search_node_relation __snr=snr_reduction)

- search_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< search_node > *_gm, gptr< vdbl::database > &_db, search_node_relation __snr=snr_reduction)
- search_node (const search_node &__sn)
- search_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< vdbl::database > &_db, search_node_relation __snr=snr_root)
- virtual ~search_node ()
- search_node & operator= (const search_node &__w)
- vdbl::userid get_dbuserid () const
- gptr< search_node > * global_model () const
- gptr< vdbl::database > * database () const
- search_node_id get_id () const
- void set_id (const search_node_id &i)
- vdbl::rowid get_rowid () const
- void set_rowid (const vdbl::rowid &i)
- void keep (const annotation &_an)
- void keep (const std::vector< annotation > &_anv)
- void unkeep (const annotation &_an)
- void unkeep (const std::vector< annotation > &_anv)

Protected Member Functions

- search_node_relation parent_relation () const

Protected Attributes

- gptr< search_node > * __global_model
- gptr< vdbl::database > * __dbase
- vdbl::userid _dbuser
- search_node_relation _snr
- search_node_id _id
- std::vector< annotation > _keep
- vdbl::rowid _rid

Friends

- class search_graph

10.250.1 Detailed Description

This is the base type of the nodes stored in the search graph and analyzed by the inference engines. Its subclasses delta_node and full_node are used in the search graph. The special subclass work_node of full_node is used to present the models to be analyzed to the inference engines.

10.250.2 Constructor & Destructor Documentation

10.250.2.1 coco::search_node::search_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< search_node > & _gm, gptr< vdbl::database > & _db, search_node_relation __snr = snr_reduction) [inline]

This constructor generates a new search node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively.

Definition at line 35 of file search_node.hpp.

10.250.2.2 coco::search_node::search_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< search_node > * _gm, gptr< vdbl::database > & _db, search_node_relation __snr = snr_reduction) [inline]

This constructor generates a new search node (a reduction node by default) with search_node_id i and search_node_relation ___snr. The parameters _gm, _db, and _dui initialize the global model, search database, and the database user id, respectively.

Definition at line 43 of file search_node.hpp.

10.250.2.3 coco::search_node::search_node (const search_node & __sn) [inline]

Standard Copy Constructor

Definition at line 51 of file search_node.hpp.

10.250.2.4 coco::search_node::search_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< vdbl::database > & _db, search_node_relation __snr = snr_root) [inline]

This constructor generates a new search node (the graph's root node by default) with search_node_id i and search_node_relation ___snr. The parameters _db, and _dui initialize the search database and the database user id, respectively. The global model pointer is set to NULL. The constructor can be used for standalone nodes, as well.

Definition at line 59 of file search_node.hpp.

10.250.2.5 coco::search_node::~search_node ( ) [inline, virtual]

Standard Destructor

Definition at line 66 of file search_node.hpp.

10.250.3 Member Function Documentation

10.250.3.1 gptr<vdbl::database>* coco::coco::search_node::database ( ) const [inline]

This is the accessor method for the search database.

Definition at line 157 of file search_graph.cc.

10.250.3.2 vdbl::userid coco::coco::search_node::get_dbuserid ( ) const [inline]

This is the accessor method for the database user id.

Definition at line 151 of file search_graph.cc.

10.250.3.3 search_node_id coco::coco::search_node::get_id ( ) const [inline]

This is the accessor method for the search node id of this node.

Definition at line 160 of file search_graph.cc.

10.250.3.4 vdbl::rowid coco::coco::search_node::get_rowid ( ) const [inline]

This is the accessor method for the row id of this node.

Definition at line 166 of file search_graph.cc.

10.250.3.5 gptr<search_node>* coco::coco::search_node::global_model ( ) const [inline]

This is the accessor method for the global model.

Definition at line 154 of file search_graph.cc.

10.250.3.6 virtual bool coco::coco::search_node::is_delta ( ) const [inline, virtual]

This method is called to determine whether a search node stores only deltas (delta_node) or a full model (full_node).

See also

delta.

Reimplemented in coco::coco::full_node, coco::full_node, coco::coco::delta_node, and coco::delta_node.

Definition at line 115 of file search_graph.cc.

10.250.3.7 void coco::coco::search_node::keep (const annotation & _an) [inline]

A call to this method informs the search node to be the keeper of the annotation _an.

Definition at line 173 of file search_graph.cc.

10.250.3.8 void coco::coco::search_node::keep (const std::vector< annotation > & _anv) [inline]

A call to this method informs the search node to be the keeper of all the annotations in _anv.

Definition at line 177 of file search_graph.cc.

10.250.3.9 search_node & coco::search_node::operator= (const search_node & __w) [inline]

Standard Assignment Operator

Definition at line 175 of file search_node.hpp.

10.250.3.10 search_node_relation coco::coco::search_node::parent_relation ( ) const [inline, protected]

A call to this method returns the relation of this node to its parent in the search graph.

Definition at line 110 of file search_graph.cc.

10.250.3.11 void coco::coco::search_node::set_id (const search_node_id & *i*) [inline]

This is the set method for the search node id of this node.

Definition at line 163 of file search_graph.cc.

10.250.3.12 void coco::coco::search_node::set_rowid (const vdbl::rowid & *i*) [inline]

This is the set method for the row id of this node.

Definition at line 169 of file search_graph.cc.

10.250.3.13 void coco::search_node::unkeep (const annotation & *_an*) [inline]

A call to this method informs the search node to no longer be the keeper of the annotation _an.

Definition at line 82 of file search_node.hpp.

10.250.3.14 void coco::search_node::unkeep (const std::vector< annotation > & *_anv*) [inline]

A call to this method informs the search node to no longer be the keeper of all the annotations in _anv.

Definition at line 93 of file search_node.hpp.

10.250.4 Friends And Related Function Documentation

10.250.4.1 friend class search_graph [friend]

Definition at line 188 of file search_graph.cc.

10.250.5 Member Data Documentation

10.250.5.1 gptr<vdbl::database>* coco::coco::search_node::__dbase [protected]

This is a pointer to a global pointer to the search database associated to this search graph.

Definition at line 89 of file search_graph.cc.

10.250.5.2 gptr<search_node>* coco::coco::search_node::__global_model [protected]

This is a pointer to a global pointer to the top of the search graph. It is NULL for a standalone node or the top of the graph.

Definition at line 86 of file search_graph.cc.

10.250.5.3 vdbl::userid coco::coco::search_node::_dbuser [protected]

This member stores the database user id under which this search is performed. For the in-memory database this is irrelevant.

Definition at line 92 of file search_graph.cc.

10.250.5.4 search_node_id coco::coco::search_node::_id [protected]

This is the unique identifier of this search node.

Definition at line 97 of file search_graph.cc.

10.250.5.5 std::vector<annotation> coco::coco::search_node::_keep [protected]

The _keep member is a list of all those annotations which are kept by this search node. If the node is destroyed all kept annotations are removed from the search database.

Definition at line 101 of file search_graph.cc.

10.250.5.6 vdbl::rowid coco::coco::search_node::_rid [protected]

This row id specifies the row in the search info table where the hook information for this search_node is stored

Definition at line 105 of file search_graph.cc.

10.250.5.7 search_node_relation coco::coco::search_node::_snr [protected]

This member specifies the relation of the search node to its parent(s) in the search graph.

Definition at line 95 of file search_graph.cc.

The documentation for this class was generated from the following files:

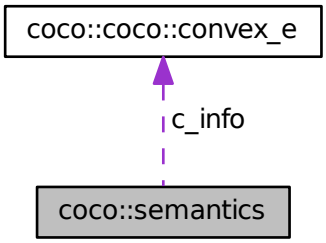
- search_node.h
- search_node.hpp

10.251 coco::semantics Class Reference

Expression Semantics.

`#include <semantics.h>`

Collaboration diagram for coco::semantics:



Public Member Functions

- semantics ()
- semantics (const semantics &__s)
- ~semantics ()
- void read (char *c)
- void merge (const semantics &__s)

Access and Storage Methods

These methods are for access and storage of all those flags that are not automatically stored by the simplifier.

- const convex_e & convexity () const
- void convexity (const convex_e &__c)
- const tristate & separability () const
- void separability (const tristate &__c)
- const activity_descr & activity () const
- void activity (const activity_descr &__c)
- bool kj_flag () const
- void kj_flag (bool __c)
- bool integer_flag () const
- void integer_flag (bool __c)
- bool hard_flag () const
- void hard_flag (bool __c)
- bool is_at_either_bound () const
- void is_at_either_bound (bool __c)
- const type_annotation & type () const
- void type (const type_annotation &__c)
- bool redundancy () const
- bool inactive_hi () const
- bool inactive_lo () const
- bool inactive () const

Public Attributes

- struct {
  convex_e c_info
  activity_descr act
  tristate separable
  bool is_at_either_bound
  } property_flags

- struct {
  bool kj
  bool integer
  type_annotation type
  bool hard
  } annotation_flags

- struct {
  tristate has_0chnbase
  } info_flags

- unsigned int _0chnbase
- int addinfo
- int degree
- int dim
- int stage

Friends

- std::ostream & operator<< (std::ostream &o, const semantics &__s)

  C++ stream output operator for semantics.

10.251.1 Detailed Description

This class is used to describe additional semantics information associated to an expression_node in the expression DAG.

10.251.2 Constructor & Destructor Documentation

10.251.2.1 coco::semantics::semantics ( )

Standard Constructor

10.251.2.2 coco::semantics::semantics (const semantics & __s) [inline]

Standard Copy Constructor

Definition at line 140 of file semantics.h.

10.251.2.3 coco::semantics::~semantics ( ) [inline]

Standard Destructor

Definition at line 148 of file semantics.h.

10.251.3 Member Function Documentation

10.251.3.1 const activity_descr& coco::semantics::activity ( ) const [inline]

This is the accessor for the activity description in the property_flags structure.

Definition at line 206 of file semantics.h.

10.251.3.2 void coco::semantics::activity (const activity_descr & __c) [inline]

This is the storage method for the activity description in the property_flags structure.

Definition at line 209 of file semantics.h.

10.251.3.3 const convex_e& coco::semantics::convexity ( ) const [inline]

This is the accessor for the convex_e specifier in the property_flags structure.

Definition at line 192 of file semantics.h.

10.251.3.4 void coco::semantics::convexity (const convex_e & __c) [inline]

This is the storage method for the convex_e specifier in the property_flags structure.

Definition at line 195 of file semantics.h.

10.251.3.5 bool coco::semantics::hard_flag ( ) const [inline]

This is the accessor for the hard (constraint) flag in the annotation_flags structure.

Definition at line 227 of file semantics.h.

10.251.3.6 void coco::semantics::hard_flag (bool __c) [inline]

This is the storage method for the hard (constraint) flag in the annotation_flags structure.

Definition at line 230 of file semantics.h.

10.251.3.7 bool coco::semantics::inactive ( ) const [inline]

This method returns whether the constraint is known to be inactive.

Definition at line 258 of file semantics.h.

10.251.3.8 bool coco::semantics::inactive_hi ( ) const [inline]

This method returns whether the constraint is known to be inactive at the upper bound.

Definition at line 251 of file semantics.h.

10.251.3.9 bool coco::semantics::inactive_lo ( ) const [inline]

This method returns whether the constraint is known to be inactive at the lower bound.

Definition at line 255 of file semantics.h.

10.251.3.10 bool coco::semantics::integer_flag ( ) const [inline]

This is the accessor for the integer flag in the annotation_flags structure.

Definition at line 220 of file semantics.h.

10.251.3.11 void coco::semantics::integer_flag (bool __c) [inline]

This is the storage method for the integer flag in the annotation_flags structure.

Definition at line 223 of file semantics.h.

10.251.3.12 bool coco::semantics::is_at_either_bound ( ) const [inline]

This is the accessor for the is_at_either_bound flag in the property_flags structure.

Definition at line 234 of file semantics.h.

10.251.3.13 void coco::semantics::is_at_either_bound (bool __c) [inline]

This is the storage method for the is_at_either_bound flag in the property_flags structure.

Definition at line 237 of file semantics.h.

10.251.3.14 bool coco::semantics::kj_flag ( ) const [inline]

This is the accessor for the KJ flag in the annotation_flags structure.

Definition at line 213 of file semantics.h.

10.251.3.15 void coco::semantics::kj_flag (bool __c) [inline]

This is the storage method for the KJ flag in the annotation_flags structure.

Definition at line 216 of file semantics.h.

10.251.3.16 void coco::semantics::merge (const semantics & __s)

This method merges this semantics structure with the structure __s.

10.251.3.17 void coco::semantics::read (char * c)

A call to this method reads a semantics structure from the C string c (.dag format).

10.251.3.18 bool coco::semantics::redundancy ( ) const [inline]

This method returns whether the constraint is known to be redundant.

Definition at line 247 of file semantics.h.

10.251.3.19 const tristate& coco::semantics::separability ( ) const [inline]

This is the accessor for the seperable entry in the property_flags structure.

Definition at line 199 of file semantics.h.

10.251.3.20 void coco::semantics::separability (const tristate & __c) [inline]

This is the storage method for the seperable entry in the property_flags structure.

Definition at line 202 of file semantics.h.

10.251.3.21 const type_annotation& coco::semantics::type ( ) const [inline]

This is the accessor for the type specifier in the annotation_flags structure.

Definition at line 241 of file semantics.h.

10.251.3.22 void coco::semantics::type (const type_annotation & __c) [inline]

This is the storage method for the type specifier in the annotation_flags structure.

Definition at line 244 of file semantics.h.

10.251.4 Friends And Related Function Documentation

10.251.4.1 std::ostream& operator<< (std::ostream & o, const semantics & __s) [friend]

This operator writes a semantics expression in .dag format to an `ostream`.

Definition at line 224 of file semantics.cc.

10.251.5 Member Data Documentation

10.251.5.1 unsigned int coco::semantics::_0chnbase

This member specifies the node number of the base of the 0-chain (univariate subgraph) leading to this node.

Definition at line 120 of file semantics.h.

10.251.5.2 activity_descr coco::semantics::act

The activity information for constraints

Definition at line 99 of file semantics.h.

10.251.5.3 int coco::semantics::addinfo

for KJ variables this numbe is the corresponding constraint number, -1 if kappa (for objective)

Definition at line 124 of file semantics.h.

10.251.5.4 struct { ... } coco::semantics::annotation_flags

The annotation_flags describe model specific properties

10.251.5.5 convex_e coco::semantics::c_info

The convexity structure for any node

Definition at line 98 of file semantics.h.

10.251.5.6 int coco::semantics::degree

This member holds the polynomial degree of the expression this node represents, and it is -1 in case of a non-polynomial non-linearity

Definition at line 128 of file semantics.h.

10.251.5.7 int coco::semantics::dim

The dim member stores the number of variables involved in that expression (the apparent dimension).

Definition at line 131 of file semantics.h.

10.251.5.8 bool coco::semantics::hard

a hard or soft constraint?

Definition at line 110 of file semantics.h.

10.251.5.9 tristate coco::semantics::has_0chnbase

node has a 0-chain base?

Definition at line 115 of file semantics.h.

10.251.5.10 struct { ... } coco::semantics::info_flags

The info_flags describe algorithmic properties

10.251.5.11 bool coco::semantics::integer

integer (not real) value?

Definition at line 108 of file semantics.h.

10.251.5.12 bool coco::semantics::is_at_either_bound

Whether a constraint is known to be on either bound (unknown on which)

Definition at line 101 of file semantics.h.

10.251.5.13 bool coco::semantics::kj

from KJ conditions?

Definition at line 107 of file semantics.h.

10.251.5.14 struct { ... } coco::semantics::property_flags

The property_flags describe mathematical constraint properties

10.251.5.15 tristate coco::semantics::separable

The seperability information for any node.

Definition at line 100 of file semantics.h.

10.251.5.16 int coco::semantics::stage

This describes the stage to which this node belongs in a multistage problem.

Definition at line 134 of file semantics.h.

10.251.5.17 type_annotation coco::semantics::type

a exists, forall, free, stochastic type

Definition at line 109 of file semantics.h.

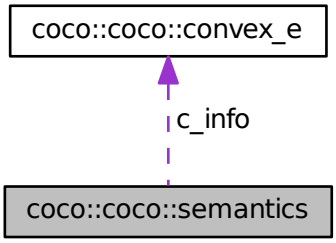
The documentation for this class was generated from the following file:

- semantics.h

10.252 coco::coco::semantics Class Reference

Expression Semantics.

Collaboration diagram for coco::coco::semantics:



Public Member Functions

- semantics ()

- semantics (const semantics &__s)
- ~semantics ()
- void read (char *c)
- void merge (const semantics &__s)

Access and Storage Methods

These methods are for access and storage of all those flags that are not automatically stored by the simplifier.

- const convex_e & convexity () const
- void convexity (const convex_e &__c)
- const tristate & separability () const
- void separability (const tristate &__c)
- const activity_descr & activity () const
- void activity (const activity_descr &__c)
- bool kj_flag () const
- void kj_flag (bool __c)
- bool integer_flag () const
- void integer_flag (bool __c)
- bool hard_flag () const
- void hard_flag (bool __c)
- bool is_at_either_bound () const
- void is_at_either_bound (bool __c)
- const type_annotation & type () const
- void type (const type_annotation &__c)
- bool redundancy () const
- bool inactive_hi () const
- bool inactive_lo () const
- bool inactive () const

Public Attributes

- struct {
  convex_e c_info
  activity_descr act
  tristate separable
  bool is_at_either_bound
  } property_flags

- struct {
  bool kj
  bool integer
  type_annotation type
  bool hard
  } annotation_flags

- struct {
  tristate has_0chnbase
  } info_flags

- unsigned int _0chnbase
- int addinfo
- int degree
- int dim
- int stage

Friends

- std::ostream & operator<< (std::ostream &o, const semantics &__s)

  C++ stream output operator for semantics.

10.252.1 Detailed Description

This class is used to describe additional semantics information associated to an expression_node in the expression DAG.

10.252.2 Constructor & Destructor Documentation

10.252.2.1 coco::semantics::semantics ( ) [inline]

Standard Constructor

Definition at line 263 of file search_graph.cc.

10.252.2.2 coco::coco::semantics::semantics (const semantics & __s) [inline]

Standard Copy Constructor

Definition at line 140 of file search_graph.cc.

10.252.2.3 coco::coco::semantics::~semantics ( ) [inline]

Standard Destructor

Definition at line 148 of file search_graph.cc.

10.252.3 Member Function Documentation

10.252.3.1 const activity_descr& coco::coco::semantics::activity ( ) const [inline]

This is the accessor for the activity description in the property_flags structure.

Definition at line 206 of file search_graph.cc.

10.252.3.2 void coco::coco::semantics::activity (const activity_descr & __c) [inline]

This is the storage method for the activity description in the property_flags structure.

Definition at line 209 of file search_graph.cc.

10.252.3.3 const convex_e& coco::coco::semantics::convexity ( ) const [inline]

This is the accessor for the convex_e specifier in the property_flags structure.

Definition at line 192 of file search_graph.cc.

10.252.3.4 void coco::coco::semantics::convexity (const convex_e & __c) [inline]

This is the storage method for the convex_e specifier in the property_flags structure.

Definition at line 195 of file search_graph.cc.

10.252.3.5 bool coco::coco::semantics::hard_flag ( ) const [inline]

This is the accessor for the hard (constraint) flag in the annotation_flags structure.

Definition at line 227 of file search_graph.cc.

10.252.3.6 void coco::coco::semantics::hard_flag (bool __c) [inline]

This is the storage method for the hard (constraint) flag in the annotation_flags structure.

Definition at line 230 of file search_graph.cc.

10.252.3.7 bool coco::coco::semantics::inactive ( ) const [inline]

This method returns whether the constraint is known to be inactive.

Definition at line 258 of file search_graph.cc.

10.252.3.8 bool coco::coco::semantics::inactive_hi ( ) const [inline]

This method returns whether the constraint is known to be inactive at the upper bound.

Definition at line 251 of file search_graph.cc.

10.252.3.9 bool coco::coco::semantics::inactive_lo ( ) const [inline]

This method returns whether the constraint is known to be inactive at the lower bound.

Definition at line 255 of file search_graph.cc.

10.252.3.10 bool coco::coco::semantics::integer_flag ( ) const [inline]

This is the accessor for the integer flag in the annotation_flags structure.

Definition at line 220 of file search_graph.cc.

10.252.3.11 void coco::coco::semantics::integer_flag (bool __c) [inline]

This is the storage method for the integer flag in the annotation_flags structure.

Definition at line 223 of file search_graph.cc.

10.252.3.12 bool coco::coco::semantics::is_at_either_bound ( ) const [inline]

This is the accessor for the is_at_either_bound flag in the property_flags structure.

Definition at line 234 of file search_graph.cc.

10.252.3.13 void coco::coco::semantics::is_at_either_bound (bool __c) [inline]

This is the storage method for the is_at_either_bound flag in the property_flags structure.

Definition at line 237 of file search_graph.cc.

10.252.3.14 bool coco::coco::semantics::kj_flag ( ) const [inline]

This is the accessor for the KJ flag in the annotation_flags structure.

Definition at line 213 of file search_graph.cc.

10.252.3.15 void coco::coco::semantics::kj_flag (bool __c) [inline]

This is the storage method for the KJ flag in the annotation_flags structure.

Definition at line 216 of file search_graph.cc.

10.252.3.16 void coco::semantics::merge (const semantics & __s) [inline]

This method merges this semantics structure with the structure ___s.

Definition at line 322 of file search_graph.cc.

10.252.3.17 void coco::semantics::read (char * c)

A call to this method reads a semantics structure from the C string c (.dag format).

Definition at line 129 of file semantics.cc.

10.252.3.18 bool coco::coco::semantics::redundancy ( ) const [inline]

This method returns whether the constraint is known to be redundant.

Definition at line 247 of file search_graph.cc.

10.252.3.19 const tristate& coco::coco::semantics::separability ( ) const [inline]

This is the accessor for the seperable entry in the property_flags structure.

Definition at line 199 of file search_graph.cc.

10.252.3.20 void coco::coco::semantics::separability (const tristate & __c) [inline]

This is the storage method for the seperable entry in the property_flags structure.

Definition at line 202 of file search_graph.cc.

10.252.3.21 const type_annotation& coco::coco::semantics::type ( ) const [inline]

This is the accessor for the type specifier in the annotation_flags structure.

Definition at line 241 of file search_graph.cc.

10.252.3.22 void coco::coco::semantics::type (const type_annotation & __c) [inline]

This is the storage method for the type specifier in the annotation_flags structure.

Definition at line 244 of file search_graph.cc.

10.252.4 Friends And Related Function Documentation

10.252.4.1 std::ostream& operator<< (std::ostream & o, const semantics & __s) [friend]

This operator writes a semantics expression in .dag format to an `ostream`.

Definition at line 224 of file semantics.cc.

10.252.5 Member Data Documentation

10.252.5.1 unsigned int coco::coco::semantics::_0chnbase

This member specifies the node number of the base of the 0-chain (univariate subgraph) leading to this node.

Definition at line 120 of file search_graph.cc.

10.252.5.2 activity_descr coco::coco::semantics::act

The activity information for constraints

Definition at line 99 of file search_graph.cc.

10.252.5.3 int coco::coco::semantics::addinfo

for KJ variables this numbe is the corresponding constraint number, -1 if kappa (for objective)

Definition at line 124 of file search_graph.cc.

10.252.5.4 struct { ... } coco::coco::semantics::annotation_flags

The annotation_flags describe model specific properties

10.252.5.5 convex_e coco::coco::semantics::c_info

The convexity structure for any node

Definition at line 98 of file search_graph.cc.

10.252.5.6 int coco::coco::semantics::degree

This member holds the polynomial degree of the expression this node represents, and it is -1 in case of a non-polynomial non-linearity

Definition at line 128 of file search_graph.cc.

10.252.5.7 int coco::coco::semantics::dim

The dim member stores the number of variables involved in that expression (the apparent dimension).

Definition at line 131 of file search_graph.cc.

10.252.5.8 bool coco::coco::semantics::hard

a hard or soft constraint?

Definition at line 110 of file search_graph.cc.

10.252.5.9 tristate coco::coco::semantics::has_0chnbase

node has a 0-chain base?

Definition at line 115 of file search_graph.cc.

10.252.5.10 struct { ... } coco::coco::semantics::info_flags

The info_flags describe algorithmic properties

10.252.5.11 bool coco::coco::semantics::integer

integer (not real) value?

Definition at line 108 of file search_graph.cc.

10.252.5.12 bool coco::coco::semantics::is_at_either_bound

Whether a constraint is known to be on either bound (unknown on which)

Definition at line 101 of file search_graph.cc.

10.252.5.13 bool coco::coco::semantics::kj

from KJ conditions?

Definition at line 107 of file search_graph.cc.

10.252.5.14 struct { ... } coco::coco::semantics::property_flags

The property_flags describe mathematical constraint properties

10.252.5.15 tristate coco::coco::semantics::separable

The seperability information for any node.

Definition at line 100 of file search_graph.cc.

10.252.5.16 int coco::coco::semantics::stage

This describes the stage to which this node belongs in a multistage problem.

Definition at line 134 of file search_graph.cc.

10.252.5.17 type_annotation coco::coco::semantics::type

a exists, forall, free, stochastic type

Definition at line 109 of file search_graph.cc.

The documentation for this class was generated from the following files:

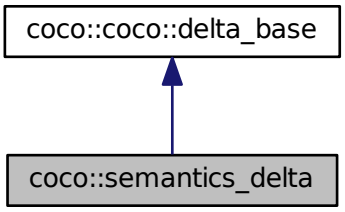
- semantics.h
- semantics.cc

10.253 coco::semantics_delta Class Reference

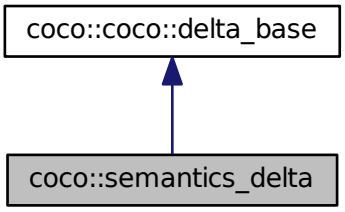
The semantics delta class for changing the node semantics within a model.

`#include <semantics_delta.h>`

Inheritance diagram for coco::semantics_delta:



Collaboration diagram for coco::semantics_delta:



Public Member Functions

- uint32_t encode_convex (uint32_t e, const convex_e &c) const
- uint32_t encode_activity (uint32_t e, const activity_descr &a) const
- uint32_t encode_is_at_either_bound (uint32_t e, bool b) const
- uint32_t encode_integer (uint32_t e, bool b) const
- uint32_t encode_hard (uint32_t e, bool b) const
- uint32_t encode_separable (uint32_t e, const tristate &t) const
- uint32_t encode_type (uint32_t e, const type_annotation &a) const
- uint32_t encode (const semantics &s) const
- void decode_convex (uint32_t e, convex_e &c) const
- void decode_activity (uint32_t e, activity_descr &a) const
- void decode_is_at_either_bound (uint32_t e, bool &b) const
- void decode_integer (uint32_t e, bool &b) const
- void decode_hard (uint32_t e, bool &b) const
- void decode_separable (uint32_t e, tristate &t) const
- void decode_type (uint32_t e, type_annotation &a) const
- void decode (uint32_t e, semantics &s) const
- semantics_delta ()
- semantics_delta (const std::vector< unsigned int > &__i, const std::vector< uint32_t > &__b)
- semantics_delta (unsigned int __i, uint32_t __b)
- semantics_delta (const semantics_delta &__d)
- semantics_delta * new_copy () const
- void destroy_copy (delta_base *__d) const
- void set (const std::vector< unsigned int > &_i, const std::vector< uint32_t > &_s)
- void set (const std::vector< unsigned int > &_i, const std::vector< semantics > &_s)
- void set (unsigned int _i, const semantics &_s)
- void set_convex (const std::vector< unsigned int > &_i, const std::vector< convex_e > &c)
- void set_convex (unsigned int _i, const convex_e &c)
- void set_activity (const std::vector< unsigned int > &_i, const std::vector< activity_descr > &a)
- void set_activity (unsigned int _i, const activity_descr &a)
- void set_separable (const std::vector< unsigned int > &_i, const std::vector< tristate > &t)
- void set_separable (unsigned int _i, const tristate &t)
- void set_is_at_either_bound (const std::vector< unsigned int > &_i, const std::vector< bool > &b)
- void set_is_at_either_bound (unsigned int _i, bool b)
- void set_integer (const std::vector< unsigned int > &_i, const std::vector< bool > &b)
- void set_integer (unsigned int _i, bool b)
- void set_hard (const std::vector< unsigned int > &_i, const std::vector< bool > &b)
- void set_hard (unsigned int _i, bool b)
- void set_type (const std::vector< unsigned int > &_i, const std::vector< type_annotation > &a)
- void set_type (unsigned int _i, const type_annotation &a)
- bool apply (work_node &_x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const semantics_delta &_c) const
- bool operator!= (const semantics_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- const std::string & get_action () const

- const std::string & get_action () const
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual void unkeep ()
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Protected Attributes

- std::string _action

Friends

- class semantics_undelta

10.253.1 Detailed Description

This class is used to specify information for changing the node semantics within a model.

10.253.2 Constructor & Destructor Documentation

10.253.2.1 coco::semantics_delta::semantics_delta ( ) [inline]

Standard Constructor

Definition at line 272 of file semantics_delta.h.

10.253.2.2 coco::semantics_delta::semantics_delta (const std::vector< unsigned int > & __i, const std::vector< uint32_t > & __b)

Constructor which explicitly assigns the indices __i and the vector of already encoded semantics structures __b.

Definition at line 35 of file semantics_delta.cc.

10.253.2.3 coco::semantics_delta::semantics_delta (unsigned int __i, uint32_t __b) [inline]

Constructor which explicitly assigns one index __i and the corresponding already encoded semantics structure __b.

Definition at line 280 of file semantics_delta.h.

10.253.2.4 coco::semantics_delta::semantics_delta (const semantics_delta & __d) [inline]

Standard Copy Constructor

Definition at line 285 of file semantics_delta.h.

10.253.3 Member Function Documentation

10.253.3.1 virtual bool coco::coco::delta_base::apply (work_node & _x, undelta_base *& _u, const delta_id & _di, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.253.3.2 bool coco::semantics_delta::apply (work_node & _x, undelta_base *& _u, const delta_id & _did, size_t & delta_size) const

Apply the delta with delta_id _d to work node _x, hereby changing the bounds as requested.

Definition at line 223 of file semantics_delta.cc.

10.253.3.3 bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.253.3.4 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.253.3.5 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.253.3.6 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.253.3.7 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.253.3.8 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.253.3.9 void coco::semantics_delta::decode (uint32_t e, semantics & s) const [inline]

This method decodes a whole semantics structure from the encoded structure e.

Definition at line 259 of file semantics_delta.h.

10.253.3.10 void coco::semantics_delta::decode_activity (uint32_t e, activity_descr & a) const [inline]

This method decodes the activity_descr part from the encoded semantics structure e.

Definition at line 235 of file semantics_delta.h.

10.253.3.11 void coco::semantics_delta::decode_convex (uint32_t e, convex_e & c) const [inline]

This method decodes the convex_e part from the encoded semantics structure e.

Definition at line 231 of file semantics_delta.h.

10.253.3.12 void coco::semantics_delta::decode_hard (uint32_t e, bool & b) const [inline]

This method decodes the hard bool from the encoded semantics structure e.

Definition at line 247 of file semantics_delta.h.

10.253.3.13 void coco::semantics_delta::decode_integer (uint32_t e, bool & b) const [inline]

This method decodes the integer bool from the encoded semantics structure e.

Definition at line 243 of file semantics_delta.h.

10.253.3.14 void coco::semantics_delta::decode_is_at_either_bound (uint32_t *e*, bool & *b*) const [inline]

This method decodes the is_at_either_bound bool from the encoded semantics structure `e`.

Definition at line 239 of file semantics_delta.h.

10.253.3.15 void coco::semantics_delta::decode_separable (uint32_t *e*, tristate & *t*) const [inline]

This method decodes the separable tristate from the encoded semantics structure `e`.

Definition at line 251 of file semantics_delta.h.

10.253.3.16 void coco::semantics_delta::decode_type (uint32_t *e*, type_annotation & *a*) const [inline]

This method decodes the type_annotation part from the encoded semantics structure `e`.

Definition at line 255 of file semantics_delta.h.

10.253.3.17 void coco::semantics_delta::destroy_copy (delta_base * *__d*) const [inline]

Clone Destructor

Definition at line 297 of file semantics_delta.h.

10.253.3.18 uint32_t coco::semantics_delta::encode (const semantics & *s*) const [inline]

This method encodes a whole semantics structure.

Definition at line 217 of file semantics_delta.h.

10.253.3.19 uint32_t coco::semantics_delta::encode_activity (uint32_t *e*, const activity_descr & *a*) const [inline]

This method adds to an already existing encoding of a semantics change the encoding of a new activity_descr `a`.

Definition at line 194 of file semantics_delta.h.

10.253.3.20 uint32_t coco::semantics_delta::encode_convex (uint32_t *e*, const convex_e & *c*) const [inline]

This method adds to an already existing encoding of a semantics change the encoding of a new convex_e `c`.

Definition at line 190 of file semantics_delta.h.

10.253.3.21 uint32_t coco::semantics_delta::encode_hard (uint32_t *e*, bool *b*) const [inline]

This method adds to an already existing encoding of a semantics change the encoding of the boolean `hard`.

Definition at line 206 of file semantics_delta.h.

10.253.3.22 uint32_t coco::semantics_delta::encode_integer (uint32_t *e*, bool *b*) const [inline]

This method adds to an already existing encoding of a semantics change the encoding of the boolean `integer`.

Definition at line 202 of file semantics_delta.h.

10.253.3.23 uint32_t coco::semantics_delta::encode_is_at_either_bound (uint32_t *e*, bool *b*) const [inline]

This method adds to an already existing encoding of a semantics change the encoding of the boolean `is_at_either_bound`.

Definition at line 198 of file semantics_delta.h.

10.253.3.24 uint32_t coco::semantics_delta::encode_separable (uint32_t *e*, const tristate & *t*) const [inline]

This method adds to an already existing encoding of a semantics change the encoding of the tristate `separable`.

Definition at line 210 of file semantics_delta.h.

10.253.3.25 uint32_t coco::semantics_delta::encode_type (uint32_t *e*, const type_annotation & *a*) const [inline]

This method adds to an already existing encoding of a semantics change the encoding of the type_-annotation.

Definition at line 214 of file semantics_delta.h.

10.253.3.26 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.253.3.27 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.253.3.28 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.253.3.29 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.253.3.30 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.253.3.31 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action a.

Definition at line 175 of file search_graph.cc.

10.253.3.32 semantics_delta* coco::semantics_delta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::delta_base.

Definition at line 295 of file semantics_delta.h.

10.253.3.33 bool coco::semantics_delta::operator!= (const delta_base & *_c*) const [inline]

Definition at line 398 of file semantics_delta.h.

10.253.3.34 bool coco::semantics_delta::operator!= (const semantics_delta & *_c*) const [inline]

Definition at line 406 of file semantics_delta.h.

10.253.3.35 bool coco::semantics_delta::operator== (const delta_base & *_c*) const [inline]

Comparison operators

Definition at line 395 of file semantics_delta.h.

10.253.3.36 bool coco::semantics_delta::operator== (const semantics_delta & *_c*) const [inline]

Comparison operators

Definition at line 403 of file semantics_delta.h.

10.253.3.37 void coco::semantics_delta::set (const std::vector< unsigned int > & *_i*, const std::vector< uint32_t > & *_s*)

This method sets the vector of indices to _i and the vector of **encoded** semantics to _s.

Definition at line 58 of file semantics_delta.cc.

10.253.3.38 void coco::semantics_delta::set (const std::vector< unsigned int > & *_i*, const std::vector< semantics > & *_s*)

This set method sets the semantics structure for all nodes whose node numbers are in _i to the semantics information stored in the corresponding entry of vector _s.

Definition at line 48 of file semantics_delta.cc.

10.253.3.39 void coco::semantics_delta::set (unsigned int _i, const semantics & _s)

This set method sets the semantics structure for the nodes with node number _i to the semantics information stored in _s.

Definition at line 127 of file semantics_delta.cc.

10.253.3.40 void coco::semantics_delta::set_activity (const std::vector< unsigned int > & _i, const std::vector< activity_descr > & a)

This set_activity method sets the activity_descr part of the semantics structure for all nodes whose node numbers are in _i to the activity_descr information stored in the corresponding entry of vector a. All other semantics information remains unchanged.

Definition at line 77 of file semantics_delta.cc.

10.253.3.41 void coco::semantics_delta::set_activity (unsigned int _i, const activity_descr & a)

This set_activity method sets the activity_descr part of the semantics structure for the node with node number _i to the activity_descr information stored in a. All other semantics information remains unchanged.

Definition at line 153 of file semantics_delta.cc.

10.253.3.42 void coco::semantics_delta::set_convex (const std::vector< unsigned int > & _i, const std::vector< convex_e > & c)

This set_convex method sets the convex_e part of the semantics structure for all nodes whose node numbers are in _i to the convex_e information stored in the corresponding entry of vector c. All other semantics information remains unchanged.

Definition at line 67 of file semantics_delta.cc.

10.253.3.43 void coco::semantics_delta::set_convex (unsigned int _i, const convex_e & c)

This set_convex method sets the convex_e part of the semantics structure for the node with node number _i to the convex_e information stored in c. All other semantics information remains unchanged.

Definition at line 139 of file semantics_delta.cc.

10.253.3.44 void coco::semantics_delta::set_hard (const std::vector< unsigned int > & _i, const std::vector< bool > & b)

This set_hard method sets the hard entry of the semantics structure for all nodes whose node numbers are in _i to the bool information stored in the corresponding entry of vector b. All other semantics information remains unchanged.

Definition at line 107 of file semantics_delta.cc.

10.253.3.45 void coco::semantics_delta::set_hard (unsigned int _i, bool b)

This set_hard method sets the hard entry of the semantics structure for the node with node number _i to the bool information stored in b. All other semantics information remains unchanged.

Definition at line 195 of file semantics_delta.cc.

10.253.3.46 void coco::semantics_delta::set_integer (const std::vector< unsigned int > & _i, const std::vector< bool > & b)

This set_integer method sets the integer entry of the semantics structure for all nodes whose node numbers are in _i to the bool information stored in the corresponding entry of vector b. All other semantics information remains unchanged.

Definition at line 97 of file semantics_delta.cc.

10.253.3.47 void coco::semantics_delta::set_integer (unsigned int _i, bool b)

This set_integer method sets the integer entry of the semantics structure for the node with node number _i to the bool information stored in b. All other semantics information remains unchanged.

Definition at line 181 of file semantics_delta.cc.

10.253.3.48 void coco::semantics_delta::set_is_at_either_bound (const std::vector< unsigned int > & _i, const std::vector< bool > & b)

This set_is_at_either_bound method sets the is_at_either_bound entry of the semantics structure for all nodes whose node numbers are in _i to the bool information stored in the corresponding entry of vector b. All other semantics information remains unchanged.

10.253.3.49 void coco::semantics_delta::set_is_at_either_bound (unsigned int _i, bool b)

This set_is_at_either_bound method sets the is_at_either_bound entry of the semantics structure for the node with node number _i to the bool information stored in b. All other semantics information remains unchanged.

10.253.3.50 void coco::semantics_delta::set_separable (const std::vector< unsigned int > & _i, const std::vector< tristate > & t)

This set_separable method sets the separable entry of the semantics structure for all nodes whose node numbers are in _i to the tristate information stored in the corresponding entry of vector t. All other semantics information remains unchanged.

Definition at line 87 of file semantics_delta.cc.

10.253.3.51 void coco::semantics_delta::set_separable (unsigned int _i, const tristate & t)

This set_separable method sets the separable entry of the semantics structure for the node with node number _i to the tristate information stored in t. All other semantics information remains unchanged.

Definition at line 167 of file semantics_delta.cc.

10.253.3.52 void coco::semantics_delta::set_type (const std::vector< unsigned int > & _i, const std::vector< type_annotation > & a)

This set_type method sets the type_annotation part of the semantics structure for all nodes whose node numbers are in _i to the type_annotation information stored in the corresponding entry of vector a. All other semantics information remains unchanged.

Definition at line 117 of file semantics_delta.cc.

10.253.3.53 void coco::semantics_delta::set_type (unsigned int *_i*, const type_annotation & *a*)

This set_type method sets the type_annotation part of the semantics structure for the node with node number `_i` to the type_annotation information stored in `a`. All other semantics information remains unchanged.

Definition at line 209 of file semantics_delta.cc.

10.253.3.54 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.253.3.55 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.253.3.56 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.253.4 Friends And Related Function Documentation

10.253.4.1 friend class semantics_undelta [friend]

Definition at line 410 of file semantics_delta.h.

10.253.5 Member Data Documentation

10.253.5.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

The documentation for this class was generated from the following files:

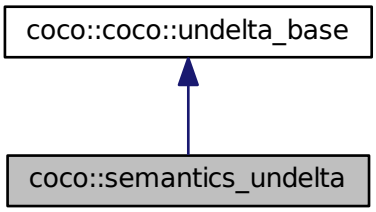
- semantics_delta.h
- semantics_delta.cc

10.254 coco::semantics_undelta Class Reference

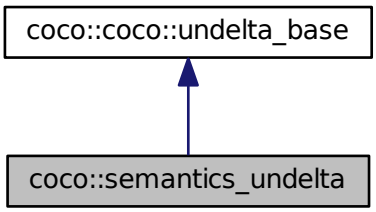
The semantics undelta class for undoing changes to the node semantics in a model.

`#include <semantics_delta.h>`

Inheritance diagram for coco::semantics_undelta:



Collaboration diagram for coco::semantics_undelta:



Public Member Functions

- semantics_undelta (const std::vector< unsigned int > &__i, const std::vector< uint32_t > &__s)
- semantics_undelta (const std::vector< unsigned int > &__i)
- semantics_undelta (const semantics_undelta &__d)
- semantics_undelta * new_copy () const
- void destroy_copy (undelta_base *__d) const
- bool unapply (work_node &_x, const delta_id &_did) const
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)

- undelta make_undelta (size_t s)
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

Friends

- class semantics_delta

10.254.1 Detailed Description

This class is used to specify undo information for changes to the node semantics within a model. This class undoes the apply of a semantics_delta.

10.254.2 Constructor & Destructor Documentation

10.254.2.1 coco::semantics_undelta::semantics_undelta (const std::vector< unsigned int > & __i, const std::vector< uint32_t > & __s) [inline]

Constructor, setting `indices` to `__i`, the `old_f_sem` to `__s`.

Definition at line 55 of file semantics_delta.h.

10.254.2.2 coco::semantics_undelta::semantics_undelta (const std::vector< unsigned int > & __i) [inline]

Constructor, setting `indices` to `__i`, and leaving `old_f_sem` empty.

Definition at line 61 of file semantics_delta.h.

10.254.2.3 coco::semantics_undelta::semantics_undelta (const semantics_undelta & __d) [inline]

Standard Copy Constructor

Definition at line 65 of file semantics_delta.h.

10.254.3 Member Function Documentation

10.254.3.1 void coco::semantics_undelta::destroy_copy (undelta_base * __d) const [inline]

Clone Destructor

Definition at line 77 of file semantics_delta.h.

10.254.3.2 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.254.3.3 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.254.3.4 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.254.3.5 semantics_undelta* coco::semantics_undelta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::undelta_base.

Definition at line 75 of file semantics_delta.h.

10.254.3.6 bool coco::semantics_undelta::unapply (work_node & *_x*, const delta_id & *_did*) const [virtual]

Undo the semantics_delta with delta_id _i in work node _x

Reimplemented from coco::coco::undelta_base.

Definition at line 256 of file semantics_delta.cc.

10.254.3.7 virtual bool coco::coco::undelta_base::unapply3 (work_node & *_x*, const work_node & *_y*, const delta_id & *_d*) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.254.3.8 virtual bool coco::coco::undelta_base::unapply3 (work_node & *_x*, const work_node & *_y*, const delta_id & *_d*) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.254.3.9 bool coco::coco::undelta_base::unapply3 (work_node & *_x*, const work_node & *_y*, const delta_id & *_d*) const [inline, virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

Definition at line 94 of file api_delta.h.

10.254.4 Friends And Related Function Documentation

10.254.4.1 friend class semantics_delta [friend]

Definition at line 82 of file semantics_delta.h.

The documentation for this class was generated from the following files:

- semantics_delta.h
- semantics_delta.cc

10.255 coco::model::simplify_visitor_0 Class Reference

`#include <model.hpp>`

Public Member Functions

- simplify_visitor_0 ()
- simplify_visitor_0 (int __nv, std::vector< unsigned int > *__dn, std::vector< unsigned int > *__dg, std::multimap< unsigned int, unsigned int > *__de, std::vector< bool > *__vt, model *__m, bool restr_simp=false)
- simplify_visitor_0 (const simplify_visitor_0 &__x)
- void vinit ()
- void init ()
- void postorder_help (const expression_node &r, unsigned int n_chld)
- bool postorder (expression_node &r)
- bool preorder (expression_node &r)
- simplify_visitor_0 value ()
- simplify_visitor_0 vvalue ()
- void simple_sum_prod_update (expression_node &r, const simplify_visitor_0 &__s)
- void delete_const_node (unsigned int nnum, unsigned int pnum, unsigned int flags, int N)
- void transfer_ghost_down (unsigned int nnum, unsigned int pnum)
- void vcollect (const simplify_visitor_0 &__s)
- void collect (expression_node &r, const simplify_visitor_0 &__s)

10.255.1 Detailed Description

This class is a postorder visitor used for the phase 0 of the simplifier.

10.255.2 Constructor & Destructor Documentation

10.255.2.1 coco::model::simplify_visitor_0::simplify_visitor_0 ( ) [inline]

Definition at line 956 of file model.hpp.

10.255.2.2 coco::model::simplify_visitor_0::simplify_visitor_0 (int *__nv,* std::vector< unsigned int > * *__dn,* std::vector< unsigned int > * *__dg,* std::multimap< unsigned int, unsigned int > * *__de,* std::vector< bool > * *__vt,* model * *__m,* bool *restr_simp* = false) [inline]

Definition at line 964 of file model.hpp.

10.255.2.3 coco::model::simplify_visitor_0::simplify_visitor_0 (const simplify_visitor_0 & *__x*) [inline]

Definition at line 976 of file model.hpp.

10.255.3 Member Function Documentation

10.255.3.1 void coco::model::simplify_visitor_0::collect (expression_node & *r,* const simplify_visitor_0 & *__s*)

Definition at line 2021 of file model.cc.

10.255.3.2 void coco::model::simplify_visitor_0::delete_const_node (unsigned int *nnum,* unsigned int *pnum,* unsigned int *flags,* int *N*) [inline]

Definition at line 1087 of file model.hpp.

10.255.3.3 void coco::model::simplify_visitor_0::init ( ) [inline]

Definition at line 992 of file model.hpp.

10.255.3.4 bool coco::model::simplify_visitor_0::postorder (expression_node & *r*)

Definition at line 1815 of file model.cc.

10.255.3.5 void coco::model::simplify_visitor_0::postorder_help (const expression_node & *r,* unsigned int *n_chld*) [inline]

Definition at line 3223 of file model.hpp.

10.255.3.6 bool coco::model::simplify_visitor_0::preorder (expression_node & *r*) [inline]

Definition at line 997 of file model.hpp.

10.255.3.7 void coco::model::simplify_visitor_0::simple_sum_prod_update (expression_node & *r,* const simplify_visitor_0 & *__s*) [inline]

Definition at line 1038 of file model.hpp.

10.255.3.8 void coco::model::simplify_visitor_0::transfer_ghost_down (unsigned int *nnum,* unsigned int *pnum*) [inline]

Definition at line 1111 of file model.hpp.

10.255.3.9 simplify_visitor_0 coco::model::simplify_visitor_0::value ( ) [inline]

Definition at line 1035 of file model.hpp.

10.255.3.10 void coco::model::simplify_visitor_0::vcollect (const simplify_visitor_0 & __s)

Definition at line 1901 of file model.cc.

10.255.3.11 void coco::model::simplify_visitor_0::vinit ( ) [inline]

Definition at line 989 of file model.hpp.

10.255.3.12 simplify_visitor_0 coco::model::simplify_visitor_0::vvalue ( ) [inline]

Definition at line 1036 of file model.hpp.

The documentation for this class was generated from the following files:

- model.hpp
- model.cc

10.256 coco::model::simplify_visitor_m Class Reference

`#include <model.hpp>`

Public Member Functions

- simplify_visitor_m ()
- simplify_visitor_m (int __nv, std::vector< unsigned int > *__dn, std::multimap< unsigned int, unsigned int > *__de, std::vector< bool > *__vt, model *__m)
- simplify_visitor_m (const simplify_visitor_m &__x)
- void vinit ()
- void init ()
- void postorder_help (const expression_node &r, unsigned int n_chld)
- bool postorder (expression_node &r)
- bool preorder (expression_node &r)
- const simplify_visitor_m & value ()
- const simplify_visitor_m & vvalue ()
- void vcollect (const simplify_visitor_m &__s)
- void collect (expression_node &r, const simplify_visitor_m &__s)

10.256.1 Detailed Description

This class is a postorder visitor used for the micro-simplifier, which only initializes the most important counters and semantics structures.

10.256.2 Constructor & Destructor Documentation

10.256.2.1 coco::model::simplify_visitor_m::simplify_visitor_m ( ) [inline]

Definition at line 1178 of file model.hpp.

10.256.2.2 coco::model::simplify_visitor_m::simplify_visitor_m (int *__nv*, std::vector< unsigned int > * *__dn*, std::multimap< unsigned int, unsigned int > * *__de*, std::vector< bool > * *__vt*, model * *__m*) [inline]

Definition at line 1184 of file model.hpp.

10.256.2.3 coco::model::simplify_visitor_m::simplify_visitor_m (const simplify_visitor_m & *__x*) [inline]

Definition at line 1193 of file model.hpp.

10.256.3 Member Function Documentation

10.256.3.1 void coco::model::simplify_visitor_m::collect (expression_node & *r*, const simplify_visitor_m & *__s*)

Definition at line 3409 of file model.cc.

10.256.3.2 void coco::model::simplify_visitor_m::init ( ) [inline]

Definition at line 1206 of file model.hpp.

10.256.3.3 bool coco::model::simplify_visitor_m::postorder (expression_node & *r*)

Definition at line 3354 of file model.cc.

10.256.3.4 void coco::model::simplify_visitor_m::postorder_help (const expression_node & *r*, unsigned int *n_chld*) [inline]

Definition at line 3368 of file model.hpp.

10.256.3.5 bool coco::model::simplify_visitor_m::preorder (expression_node & *r*) [inline]

Definition at line 1211 of file model.hpp.

10.256.3.6 const simplify_visitor_m& coco::model::simplify_visitor_m::value ( ) [inline]

Definition at line 1224 of file model.hpp.

10.256.3.7 void coco::model::simplify_visitor_m::vcollect (const simplify_visitor_m & *__s*) [inline]

Definition at line 1227 of file model.hpp.

10.256.3.8 void coco::model::simplify_visitor_m::vinit ( ) [inline]

Definition at line 1203 of file model.hpp.

10.256.3.9 const simplify_visitor_m& coco::model::simplify_visitor_m::vvalue ( ) [inline]

Definition at line 1225 of file model.hpp.

The documentation for this class was generated from the following files:

- model.hpp
- model.cc

10.257 coco::model::sort_constraints Class Reference

`#include <model.hpp>`

Public Member Functions

- bool operator() (const walker &_c1, const walker &_c2) const

10.257.1 Detailed Description

This function class is used to compare constraints and objectives for constraint and objective sorting.

10.257.2 Member Function Documentation

10.257.2.1 bool coco::model::sort_constraints::operator() (const walker & *_c1*, const walker & *_c2*) const [inline]

Definition at line 882 of file model.hpp.

The documentation for this class was generated from the following file:

- model.hpp

10.258 coco::sparsity3_visitor Class Reference

Preorder visitor which calculates sparsity structures.

Public Member Functions

- sparsity3_visitor (stlist &__o, int __n, bool __b=false)
- sparsity3_visitor (const sparsity3_visitor &__c)

- bool preorder (const expression_node &r)
- const variable_indicator & vvalue ()
- const variable_indicator & value ()
- void collect (const expression_node &r, const variable_indicator &__v)

10.258.1 Detailed Description

This preorder visitor is used to calculate the sparsity structure of 3rd derivatives.

10.258.2 Constructor & Destructor Documentation

10.258.2.1 coco::sparsity3_visitor::sparsity3_visitor (stlist & __o, int __n, bool __b = false) [inline]

Constructor, which initializes the output variable holding the sparsity structure to __o, and the number of variables to __n. If __b is true, the array base is set to 1, and to 0 otherwise.

Definition at line 122 of file sparsity3.cc.

10.258.2.2 coco::sparsity3_visitor::sparsity3_visitor (const sparsity3_visitor & __c) [inline]

Standard Copy Constructor

Definition at line 127 of file sparsity3.cc.

10.258.3 Member Function Documentation

10.258.3.1 void coco::sparsity3_visitor::collect (const expression_node & r, const variable_indicator & __v) [inline]

This is a method as needed by a preorder visitor.

Definition at line 251 of file sparsity3.cc.

10.258.3.2 bool coco::sparsity3_visitor::preorder (const expression_node & r) [inline]

This is a method as needed by a preorder visitor.

Definition at line 134 of file sparsity3.cc.

10.258.3.3 const variable_indicator& coco::sparsity3_visitor::value ( ) [inline]

This is a method as needed by a preorder visitor.

Definition at line 249 of file sparsity3.cc.

10.258.3.4 const variable_indicator& coco::sparsity3_visitor::vvalue ( ) [inline]

This is a method as needed by a preorder visitor.

Definition at line 248 of file sparsity3.cc.

The documentation for this class was generated from the following file:

- sparsity3.cc

10.259 coco::sparsity_visitor Class Reference

Preorder visitor which calculates sparsity structures.

Public Member Functions

- sparsity_visitor (splist &__o, int __n, bool __b=false)
- sparsity_visitor (const sparsity_visitor &__c)

- bool preorder (const expression_node &r)
- const variable_indicator & vvalue ()
- const variable_indicator & value ()
- void collect (const expression_node &r, const variable_indicator &__v)

10.259.1 Detailed Description

This preorder visitor is used to calculate the sparsity structure of Hessians and Jacobi matrices.

10.259.2 Constructor & Destructor Documentation

10.259.2.1 coco::sparsity_visitor::sparsity_visitor (splist & __o, int __n, bool __b = false) [inline]

Constructor, which initializes the output variable holding the sparsity structure to __o, and the number of variables to __n. If __b is true, the array base is set to 1, and to 0 otherwise.

Definition at line 116 of file sparsity.cc.

10.259.2.2 coco::sparsity_visitor::sparsity_visitor (const sparsity_visitor & __c) [inline]

Standard Copy Constructor

Definition at line 121 of file sparsity.cc.

10.259.3 Member Function Documentation

10.259.3.1 void coco::sparsity_visitor::collect (const expression_node & *r*, const variable_indicator & *__v*) [inline]

This is a method as needed by a preorder visitor.

Definition at line 251 of file sparsity.cc.

10.259.3.2 bool coco::sparsity_visitor::preorder (const expression_node & *r*) [inline]

This is a method as needed by a preorder visitor.

Definition at line 128 of file sparsity.cc.

10.259.3.3 const variable_indicator& coco::sparsity_visitor::value ( ) [inline]

This is a method as needed by a preorder visitor.

Definition at line 249 of file sparsity.cc.

10.259.3.4 const variable_indicator& coco::sparsity_visitor::vvalue ( ) [inline]

This is a method as needed by a preorder visitor.

Definition at line 248 of file sparsity.cc.

The documentation for this class was generated from the following file:

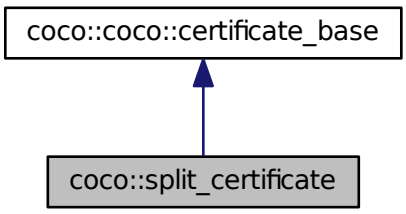
- sparsity.cc

10.260 coco::split_certificate Class Reference

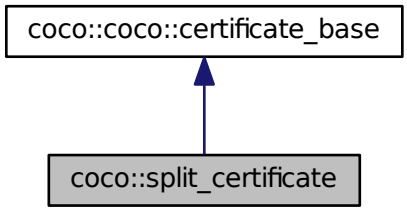
The certificate for deltas formed by splits.

`#include <api_cert.h>`

Inheritance diagram for coco::split_certificate:



Collaboration diagram for coco::split_certificate:



Public Member Functions

- split_certificate ()
- split_certificate (const std::string &mn)
- split_certificate (const split_certificate &__c)
- virtual ~split_certificate ()
- split_certificate * new_copy () const
- void destroy_copy (certificate_base *__c) const
- bool verify (const work_node &_x) const
- std::string get_contents () const
- bool operator== (const certificate_base &_c) const
- bool operator!= (const certificate_base &_c) const
- bool operator== (const split_certificate &_c) const
- bool operator!= (const split_certificate &_c) const
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)
- certificate make_certificate (const std::string &c)

Protected Attributes

- std::string _contents

10.260.1 Detailed Description

The split certificate for generated splits.

10.260.2 Constructor & Destructor Documentation

10.260.2.1 coco::split_certificate::split_certificate ( ) [inline]

Standard Constructor

Definition at line 141 of file api_cert.h.

10.260.2.2 coco::split_certificate::split_certificate (const std::string & *mn*) [inline]

Standard Constructor with explicit module name mn

Definition at line 144 of file api_cert.h.

10.260.2.3 coco::split_certificate::split_certificate (const split_certificate & *__c*) [inline]

Standard Copy Constructor

Definition at line 147 of file api_cert.h.

10.260.2.4 virtual coco::split_certificate::~split_certificate ( ) [inline, virtual]

Standard Destructor

Definition at line 156 of file api_cert.h.

10.260.3 Member Function Documentation

10.260.3.1 void coco::split_certificate::destroy_copy (certificate_base * *__c*) const [inline]

Clone Destructor

Definition at line 167 of file api_cert.h.

10.260.3.2 std::string coco::split_certificate::get_contents ( ) const [inline, virtual]

Retrieve the contents information (the certificate type) for this certificate. This method includes the module_name in the return value.

Reimplemented from coco::coco::certificate_base.

Definition at line 177 of file api_cert.h.

10.260.3.3 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.260.3.4 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents c.

Definition at line 156 of file search_graph.cc.

10.260.3.5 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents `c`.

Definition at line 156 of file search_graph.cc.

10.260.3.6 certificate coco::coco::certificate_base::make_certificate (const std::string & *c*) [inline, inherited]

Construct a certificate from this certificate_base with the contents `c`.

Definition at line 156 of file search_graph.cc.

10.260.3.7 split_certificate* coco::split_certificate::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::certificate_base.

Definition at line 159 of file api_cert.h.

10.260.3.8 bool coco::split_certificate::operator!= (const certificate_base & *_c*) const [inline]

Definition at line 184 of file api_cert.h.

10.260.3.9 bool coco::split_certificate::operator!= (const split_certificate & *_c*) const [inline]

Definition at line 191 of file api_cert.h.

10.260.3.10 bool coco::split_certificate::operator== (const certificate_base & *_c*) const [inline]

Comparison operators

Definition at line 181 of file api_cert.h.

10.260.3.11 bool coco::split_certificate::operator== (const split_certificate & *_c*) const [inline]

Comparison operators

Definition at line 189 of file api_cert.h.

10.260.3.12 bool coco::split_certificate::verify (const work_node & *_x*) const [inline, virtual]

Verification Test: Verify whether the certificate verifies the corresponding delta for work_node _x. In this class: `true`.

Reimplemented from coco::coco::certificate_base.

Definition at line 172 of file api_cert.h.

10.260.4 Member Data Documentation

10.260.4.1 std::string coco::coco::certificate_base::_contents [protected, inherited]

The contents (descriptive string, type) of this certificate

Definition at line 128 of file search_graph.cc.

The documentation for this class was generated from the following file:

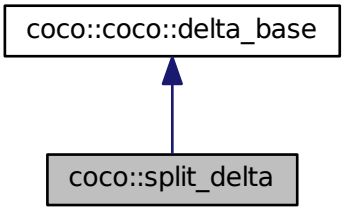
- api_cert.h

10.261 coco::split_delta Class Reference

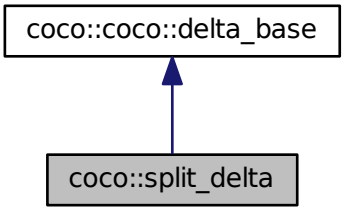
The split delta class for proposing useful splits.

`#include <split_delta.h>`

Inheritance diagram for coco::split_delta:



Collaboration diagram for coco::split_delta:



Public Member Functions

- split_delta ()
- split_delta (unsigned int _node_num, const interval &_l, const interval &_u)
- split_delta (unsigned int _node_num, const std::vector< interval > &_m)
- split_delta (const std::vector< unsigned int > &_i, const std::vector< interval > &_l, const std::vector< interval > &_u)
- split_delta (const std::list< std::vector< delta > > &__dl)
- split_delta (const std::vector< unsigned int > nnum, const std::vector< std::pair< interval, interval > > &bds)
- split_delta (const std::vector< unsigned int > nnum, const std::vector< std::vector< interval > > &bds)
- ~split_delta ()
- split_delta (const split_delta &__s)
- split_delta * new_copy () const
- void destroy_copy (delta_base *__d) const
- void add_delta (const delta &__d)
- void add_deltas (const std::vector< delta > &__d)
- void add_split (const std::vector< unsigned int > &_i, const std::vector< interval > &_b)
- void add_split (unsigned int _nnum, interval _b)
- void add_split (unsigned int _nnum, interval _l, interval _u)
- void add_split (const std::vector< unsigned int > &_i, const std::vector< interval > &_l, const std::vector< interval > &_u)
- void add_multisplit (const std::vector< unsigned int > nnum, const std::vector< std::pair< interval, interval > > &bds)
- void add_multisplit (const std::vector< unsigned int > nnum, const std::vector< std::vector< interval > > &bds)
- bool apply (work_node &x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const split_delta &_c) const
- bool operator!= (const split_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- const std::string & get_action () const
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual void unkeep ()
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Public Attributes

- std::list< std::vector< delta > > splits

Protected Attributes

- std::string _action

10.261.1 Detailed Description

This class is used to propose useful splits and store them in the proposed_splits structure in the work_node.

10.261.2 Constructor & Destructor Documentation

10.261.2.1 coco::split_delta::split_delta ( ) [inline]

Standard Constructor

Definition at line 91 of file split_delta.h.

10.261.2.2 coco::split_delta::split_delta (unsigned int *_node_num*, const interval & *_l*, const interval & *_u*) [inline]

Constructor, which initializes a split in two subproblems. The parameter _node_num specifies which variable is split. The intervals _l and _u give the intervals into which the variable _node_num is bisected.

Definition at line 101 of file split_delta.h.

10.261.2.3 coco::split_delta::split_delta (unsigned int *_node_num*, const std::vector< interval > & *_m*) [inline]

Constructor, which initializes a split in a number of subproblems. The parameter _node_num specifies which variable is split. The vector _m gives a list of intervals into which the variable _node_num is multisected.

Definition at line 114 of file split_delta.h.

10.261.2.4 coco::split_delta::split_delta (const std::vector< unsigned int > & *_i*, const std::vector< interval > & *_l*, const std::vector< interval > & *_u*) [inline]

Constructor, which initializes a split in two subproblems. The first of the subproblems is given by a bound_delta with indices vector _i and bounds vector _l, while the second subproblem has the same index vector _i but the bounds vector _u.

Definition at line 129 of file split_delta.h.

10.261.2.5 coco::split_delta::split_delta (const std::list< std::vector< delta > > & *__dl*) [inline]

Constructor, which initializes the split member with __dl.

Definition at line 138 of file split_delta.h.

10.261.2.6 coco::split_delta::split_delta (const std::vector< unsigned int > *nnum,* const std::vector< std::pair< interval, interval > > & *bds*) [inline]

Constructor which initializes the splits member with a multisplit of 2^n subproblems, where n is the length of the nnum and bds vectors. The nnum vector specifies which nodes are to be split, and the vector contains for every node split the lower and the upper subdivision interval.

Definition at line 147 of file split_delta.h.

10.261.2.7 coco::split_delta::split_delta (const std::vector< unsigned int > *nnum,* const std::vector< std::vector< interval > > & *bds*) [inline]

Constructor which initializes the splits member with a multisplit of an exponential number of subproblems, where nnum and bds are vectors of the same length. The nnum vector specifies which nodes are to be split, and the vector contains for every node split the generated subintervals.

Definition at line 156 of file split_delta.h.

10.261.2.8 coco::split_delta::~split_delta ( ) [inline]

Standard Destructor

Definition at line 162 of file split_delta.h.

10.261.2.9 coco::split_delta::split_delta (const split_delta & *__s*) [inline]

Standard Copy Constructor

Definition at line 165 of file split_delta.h.

10.261.3 Member Function Documentation

10.261.3.1 void coco::split_delta::add_delta (const delta & *__d*) [inline]

This method adds a new subproblem in the splits determined by the single delta __d.

Definition at line 174 of file split_delta.h.

10.261.3.2 void coco::split_delta::add_deltas (const std::vector< delta > & *__d*) [inline]

This method adds a new subproblem in the splits determined by the vector of deltas __d.

Definition at line 181 of file split_delta.h.

10.261.3.3 void coco::split_delta::add_multisplit (const std::vector< unsigned int > *nnum,* const std::vector< std::pair< interval, interval > > & *bds*) [inline]

This method generates a multisplit of 2^n boxes, where n is the length of the nnum and bds vectors. The nnum vector specifies which nodes are to be split, and the vector contains for every node split the lower and the upper subdivision interval.

Definition at line 287 of file split_delta.h.

10.261.3.4 void coco::split_delta::add_multisplit (const std::vector< unsigned int > *nnum*, const std::vector< std::vector< interval > > & *bds*) [inline]

This method generates a multisplit of exponentially many boxes. The nnum vector specifies which nodes are to be split, and the vector contains for every node split the generated subdivision intervals.

Definition at line 321 of file split_delta.h.

10.261.3.5 void coco::split_delta::add_split (const std::vector< unsigned int > & *_i*, const std::vector< interval > & *_b*) [inline]

This method adds a further subproblem to the splits. It is given by a bound_delta with indices vector _i and bounds vector _b.

Definition at line 187 of file split_delta.h.

10.261.3.6 void coco::split_delta::add_split (unsigned int *_nnum*, interval *_b*) [inline]

This method adds one subproblem to the splits. The parameter _nnum specifies which variable is split. The interval _b specifies the new interval.

Definition at line 197 of file split_delta.h.

10.261.3.7 void coco::split_delta::add_split (unsigned int *_nnum*, interval *_l*, interval *_u*) [inline]

This method adds two subproblems to the splits. The parameter _nnum specifies which variable is split. The intervals _l and _u give the intervals into which the variable _node_num is bisected.

Definition at line 206 of file split_delta.h.

10.261.3.8 void coco::split_delta::add_split (const std::vector< unsigned int > & *_i*, const std::vector< interval > & *_l*, const std::vector< interval > & *_u*) [inline]

This method adds two subproblems to the splits. The first of the subproblems is given by a bound_delta with indices vector _i and bounds vector _l, while the second subproblem has the same index vector _i but the bounds vector _u.

Definition at line 216 of file split_delta.h.

10.261.3.9 virtual bool coco::coco::delta_base::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_di*, size_t & *delta_size*) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.261.3.10 bool coco::split_delta::apply (work_node & *x*, undelta_base *& *_u*, const delta_id & *_did*, size_t & *delta_size*) const [inline]

Apply the delta with delta_id _d to work node _x, hereby changing the bounds as requested.

Definition at line 272 of file split_delta.h.

10.261.3.11 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.261.3.12 bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.261.3.13 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.261.3.14 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.261.3.15 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.261.3.16 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.261.3.17 void coco::split_delta::destroy_copy (delta_base * __d) const [inline]

Clone Destructor

Definition at line 170 of file split_delta.h.

10.261.3.18 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.261.3.19 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.261.3.20 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.261.3.21 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.261.3.22 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.261.3.23 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.261.3.24 split_delta* coco::split_delta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::delta_base.

Definition at line 168 of file split_delta.h.

10.261.3.25 bool coco::split_delta::operator!= (const delta_base & *_c*) const [inline]

Definition at line 246 of file split_delta.h.

10.261.3.26 bool coco::split_delta::operator!= (const split_delta & _c) const [inline]

Definition at line 253 of file split_delta.h.

10.261.3.27 bool coco::split_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Definition at line 243 of file split_delta.h.

10.261.3.28 bool coco::split_delta::operator== (const split_delta & _c) const [inline]

Comparison operators

Definition at line 251 of file split_delta.h.

10.261.3.29 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.261.3.30 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.261.3.31 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.261.4 Member Data Documentation

10.261.4.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

10.261.4.2 std::list<std::vector<delta> > coco::split_delta::splits

splits represents a list of newly created submodels. Each of these submodels is generated from the work node by a number of deltas stored in the inner vector.

Definition at line 85 of file split_delta.h.

The documentation for this class was generated from the following file:

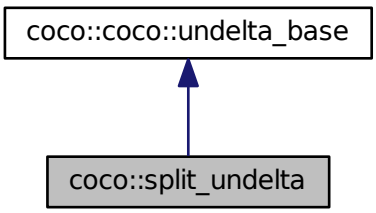
- split_delta.h

10.262 coco::split_undelta Class Reference

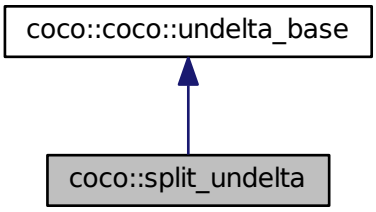
The split undelta class for removing proposed splits from a work_node.

`#include <split_delta.h>`

Inheritance diagram for coco::split_undelta:



Collaboration diagram for coco::split_undelta:



Public Member Functions

- split_undelta (const work_node::transaction_number &_se)
- split_undelta (const split_undelta &_su)
- virtual ~split_undelta ()
- split_undelta * new_copy () const
- void destroy_copy (undelta_base *__d) const

- bool unapply (work_node &x, const delta_id &_did) const
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- undelta make_undelta (size_t s)
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

10.262.1 Detailed Description

This class is used to specify information for removing proposed splits from a work node. This class undoes the apply of a split_delta.

10.262.2 Constructor & Destructor Documentation

10.262.2.1 coco::split_undelta::split_undelta (const work_node::transaction_number & _se) [inline]

Constructor, which stores the transaction number _se.

Definition at line 51 of file split_delta.h.

10.262.2.2 coco::split_undelta::split_undelta (const split_undelta & _su) [inline]

Standard Copy Constructor

Definition at line 54 of file split_delta.h.

10.262.2.3 virtual coco::split_undelta::~split_undelta ( ) [inline, virtual]

Standard Destructor

Definition at line 62 of file split_delta.h.

10.262.3 Member Function Documentation

10.262.3.1 void coco::split_undelta::destroy_copy (undelta_base * __d) const [inline]

Clone Destructor

Definition at line 67 of file split_delta.h.

10.262.3.2 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.262.3.3 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.262.3.4 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.262.3.5 split_undelta* coco::split_undelta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::undelta_base.

Definition at line 65 of file split_delta.h.

10.262.3.6 bool coco::split_undelta::unapply (work_node & *x*, const delta_id & *_did*) const [inline, virtual]

Undo the bound_delta with delta_id _i in work node _x

Reimplemented from coco::coco::undelta_base.

Definition at line 257 of file split_delta.h.

10.262.3.7 virtual bool coco::coco::undelta_base::unapply3 (work_node & *_x*, const work_node & *_y*, const delta_id & *_d*) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.262.3.8 virtual bool coco::coco::undelta_base::unapply3 (work_node & *_x*, const work_node & *_y*, const delta_id & *_d*) const [virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.262.3.9 bool coco::coco::undelta_base::unapply3 (work_node & *_x*, const work_node & *_y*, const delta_id & *_d*) const [inline, virtual, inherited]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

Definition at line 94 of file api_delta.h.

The documentation for this class was generated from the following file:

- split_delta.h

10.263 coco::statistic_info Class Reference

Base class for all inference engine statistics classes.

`#include <ie_statistic.h>`

Public Member Functions

- statistic_info ()
- virtual ~statistic_info ()
- virtual void calc_effectiveness (int n)

Public Attributes

- double effectiveness
- int number_of_infers

10.263.1 Detailed Description

This is the base class of all statistcs classes used to keep the statistics information of all the inference engines.

10.263.2 Constructor & Destructor Documentation

10.263.2.1 coco::statistic_info::statistic_info ( ) [inline]

Standard Constructor

Definition at line 49 of file ie_statistic.h.

10.263.2.2 virtual coco::statistic_info::~statistic_info ( ) [inline, virtual]

Standard Destructor

Definition at line 51 of file ie_statistic.h.

10.263.3 Member Function Documentation

10.263.3.1 virtual void coco::statistic_info::calc_effectiveness (int *n*) [inline, virtual]

A call to this method calculates the effectiveness of the inference engine and stores the result in the `effectiveness` variable.

Definition at line 55 of file ie_statistic.h.

10.263.4 Member Data Documentation

10.263.4.1 double coco::statistic_info::effectiveness

This is the effectiveness of the engine

Definition at line 43 of file ie_statistic.h.

10.263.4.2 int coco::statistic_info::number_of_infers

The number_of_infers variable keeps the number of calls to the `infer` method of the inference engine.

Definition at line 46 of file ie_statistic.h.

The documentation for this class was generated from the following file:

- ie_statistic.h

10.264 sum_deltas Class Reference

Pre-post visitor for summing up all the deltas during work node extraction.

`#include <sum_deltas.h>`

Public Member Functions

- sum_deltas (std::list< delta_id > &_dels, std::set< search_node_id > &_nds)
- sum_deltas (std::list< delta_id > &_dels, std::set< search_node_id > &_nds, const std::set< search_node_id > &_c_nds)
- sum_deltas (std::list< delta_id > &_dels, std::set< search_node_id > &_nds, const search_node_id &_psnid)

- void vinit ()
- void vcollect (return_value const &)
- return_value vvalue ()
- bool preorder (search_node ∗const &r)
- void collect (search_node ∗const &, return_value const &ancestor)
- bool postorder (search_node ∗const &r)
- return_value value ()

Public Attributes

- std::vector< return_value > ancestors
- return_value result
- std::list< delta_id > ∗ dels
- std::set< search_node_id > ∗ nds
- const std::set< search_node_id > ∗ c_nds
- const search_node_id ∗ psnid

10.264.1 Detailed Description

This pre-post visitor is used to sum up all the deltas when a new work_node is extracted from the search_-node the focus points at.

10.264.2 Constructor & Destructor Documentation

10.264.2.1 sum_deltas::sum_deltas (std::list< delta_id > & *_dels,* std::set< search_node_id > & *_nds*) [inline]

Constructor for uncompared search

Definition at line 77 of file sum_deltas.h.

10.264.2.2 sum_deltas::sum_deltas (std::list< delta_id > & *_dels,* std::set< search_node_id > & *_nds,* const std::set< search_node_id > & *_c_nds*) [inline]

Constructor for compared search with set of parent ids

Definition at line 86 of file sum_deltas.h.

10.264.2.3 sum_deltas::sum_deltas (std::list< delta_id > & *_dels,* std::set< search_node_id > & *_nds,* const search_node_id & *_psnid*) [inline]

Constructor for compared search with single parent id

Definition at line 96 of file sum_deltas.h.

10.264.3 Member Function Documentation

10.264.3.1 void sum_deltas::collect (search_node *const & , return_value const & *ancestor*) [inline]

This method is required by a prepost_visitor.

Definition at line 141 of file sum_deltas.h.

10.264.3.2 bool sum_deltas::postorder (search_node *const & *r*) [inline]

This method is required by a prepost_visitor.

Definition at line 147 of file sum_deltas.h.

10.264.3.3 bool sum_deltas::preorder (search_node *const & *r*) [inline]

This method is required by a prepost_visitor.

Definition at line 125 of file sum_deltas.h.

10.264.3.4 return_value sum_deltas::value ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 170 of file sum_deltas.h.

10.264.3.5 void sum_deltas::vcollect (return_value const &) [inline]

This method is required by a prepost_visitor.

Definition at line 112 of file sum_deltas.h.

10.264.3.6 void sum_deltas::vinit ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 107 of file sum_deltas.h.

10.264.3.7 return_value sum_deltas::vvalue ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 119 of file sum_deltas.h.

10.264.4 Member Data Documentation

10.264.4.1 std::vector<return_value> sum_deltas::ancestors

The return values of the ancestors

Definition at line 64 of file sum_deltas.h.

10.264.4.2 const std::set<search_node_id>* sum_deltas::c_nds

The list of search_node parents of the compared node

Definition at line 72 of file sum_deltas.h.

10.264.4.3 std::list<delta_id>* sum_deltas::dels

The list of deltas to be summed

Definition at line 68 of file sum_deltas.h.

10.264.4.4 std::set<search_node_id>* sum_deltas::nds

The list of search_node parents

Definition at line 70 of file sum_deltas.h.

10.264.4.5 const search_node_id* sum_deltas::psnid

The search_node parent until which we sum

Definition at line 74 of file sum_deltas.h.

10.264.4.6 return_value sum_deltas::result

The return value of this visit

Definition at line 66 of file sum_deltas.h.

The documentation for this class was generated from the following file:

- sum_deltas.h

10.265 coco::sum_deltas Class Reference

Pre-post visitor for summing up all the deltas during work node extraction.

Public Member Functions

- sum_deltas (std::list< delta_id > &_dels, std::set< search_node_id > &_nds)
- sum_deltas (std::list< delta_id > &_dels, std::set< search_node_id > &_nds, const std::set< search_node_id > &_c_nds)
- sum_deltas (std::list< delta_id > &_dels, std::set< search_node_id > &_nds, const search_node_id &_psnid)

- void vinit ()
- void vcollect (return_value const &)
- return_value vvalue ()
- bool preorder (search_node *const &r)
- void collect (search_node *const &, return_value const &ancestor)
- bool postorder (search_node *const &r)
- return_value value ()

Public Attributes

- std::vector< return_value > ancestors
- return_value result
- std::list< delta_id > * dels
- std::set< search_node_id > * nds
- const std::set< search_node_id > * c_nds
- const search_node_id * psnid

10.265.1 Detailed Description

This pre-post visitor is used to sum up all the deltas when a new work_node is extracted from the search_node the focus points at.

10.265.2 Constructor & Destructor Documentation

10.265.2.1 coco::sum_deltas::sum_deltas (std::list< delta_id > & *_dels*, std::set< search_node_id > & *_nds*) [inline]

Constructor for uncompared search

Definition at line 77 of file search_graph.cc.

10.265.2.2 coco::sum_deltas::sum_deltas (std::list< delta_id > & *_dels*, std::set< search_node_id > & *_nds*, const std::set< search_node_id > & *_c_nds*) [inline]

Constructor for compared search with set of parent ids

Definition at line 86 of file search_graph.cc.

10.265.2.3 coco::sum_deltas::sum_deltas (std::list< delta_id > & *_dels*, std::set< search_node_id > & *_nds*, const search_node_id & *_psnid*) [inline]

Constructor for compared search with single parent id

Definition at line 96 of file search_graph.cc.

10.265.3 Member Function Documentation

10.265.3.1 void coco::sum_deltas::collect (search_node *const & , return_value const & *ancestor*) [inline]

This method is required by a prepost_visitor.

Definition at line 141 of file search_graph.cc.

10.265.3.2 bool coco::sum_deltas::postorder (search_node *const & *r*) [inline]

This method is required by a prepost_visitor.

Definition at line 147 of file search_graph.cc.

10.265.3.3 bool coco::sum_deltas::preorder (search_node *const & *r*) [inline]

This method is required by a prepost_visitor.

Definition at line 125 of file search_graph.cc.

10.265.3.4 return_value coco::sum_deltas::value ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 170 of file search_graph.cc.

10.265.3.5 void coco::sum_deltas::vcollect (return_value const &) [inline]

This method is required by a prepost_visitor.

Definition at line 112 of file search_graph.cc.

10.265.3.6 void coco::sum_deltas::vinit ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 107 of file search_graph.cc.

10.265.3.7 return_value coco::sum_deltas::vvalue ( ) [inline]

This method is required by a prepost_visitor.

Definition at line 119 of file search_graph.cc.

10.265.4 Member Data Documentation

10.265.4.1 std::vector<return_value> coco::sum_deltas::ancestors

The return values of the ancestors

Definition at line 64 of file search_graph.cc.

10.265.4.2 const std::set<search_node_id>* coco::sum_deltas::c_nds

The list of search_node parents of the compared node

Definition at line 72 of file search_graph.cc.

10.265.4.3 std::list<delta_id>* coco::sum_deltas::dels

The list of deltas to be summed

Definition at line 68 of file search_graph.cc.

10.265.4.4 std::set<search_node_id>* coco::sum_deltas::nds

The list of search_node parents

Definition at line 70 of file search_graph.cc.

10.265.4.5 const search_node_id* coco::sum_deltas::psnid

The search_node parent until which we sum

Definition at line 74 of file search_graph.cc.

10.265.4.6 return_value coco::sum_deltas::result

The return value of this visit

Definition at line 66 of file search_graph.cc.

The documentation for this class was generated from the following file:

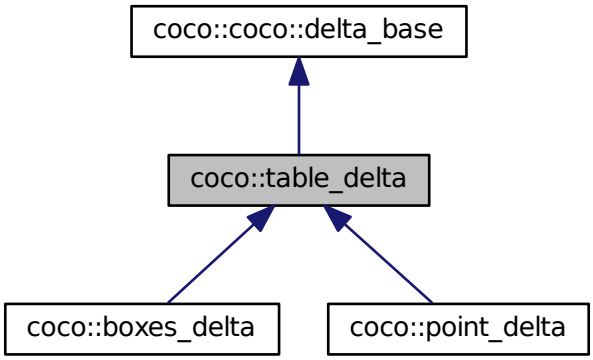
- sum_deltas.h

10.266 coco::table_delta Class Reference

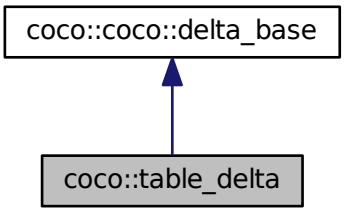
The base class for all deltas adding information to the search database.

`#include <table_delta.h>`

Inheritance diagram for coco::table_delta:



Collaboration diagram for coco::table_delta:



Public Types

- typedef std::pair< std::string, dbt_row > t_line
- typedef std::vector< t_line > t_ctr

Public Member Functions

- table_delta (const std::string &a)
- table_delta (const std::string &a, const t_ctr &_n, const std::vector< annotation > &_r)
- table_delta (const table_delta &__d)

- table_delta (const std::string &a, const std::string &__tn, const dbt_row &__t)
- table_delta (const std::string &a, const std::string &__tn, const std::vector< dbt_row > &__t)
- void add (const t_line &_tl)
- void add (const std::string &_tn, const dbt_row &_r)
- void add (const std::vector< t_line > &_tlv)
- void rm (const annotation &_tr)
- void rm (const std::vector< annotation > &_trv)
- virtual table_delta * new_copy () const
- virtual void destroy_copy (delta_base *__d) const
- virtual void create_table (work_node &_x, vdbl::standard_table *&ptb, const std::string &__t) const
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_did, size_t &delta_size) const
- void convert (work_node &_x, delta_base *&_u)
- bool operator== (const delta_base &_c) const
- bool operator!= (const delta_base &_c) const
- bool operator== (const table_delta &_c) const
- bool operator!= (const table_delta &_c) const
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- const std::string & get_action () const
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual void unkeep ()
- virtual void unkeep ()
- virtual bool apply (work_node &_x, undelta_base *&_u, const delta_id &_di, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d, size_t &delta_size) const

Protected Attributes

- std::string _action

10.266.1 Detailed Description

A base delta class which provides all functionality to add information to the search database. It is, after storing the information in the search database, converted to an annotation_delta. Therefore, no `table_undelta` class is needed.

10.266.2 Member Typedef Documentation

10.266.2.1 typedef std::vector<t_line> coco::table_delta::t_ctr

A variable of this type holds the new table entries.

Definition at line 49 of file table_delta.h.

10.266.2.2 typedef std::pair<std::string,dbt_row> coco::table_delta::t_line

This type specifies one row in one table.

Definition at line 47 of file table_delta.h.

10.266.3 Constructor & Destructor Documentation

10.266.3.1 coco::table_delta::table_delta (const std::string & *a*) [inline]

Standard Constructor, which sets the action descriptor `a`

Definition at line 59 of file table_delta.h.

10.266.3.2 coco::table_delta::table_delta (const std::string & *a*, const t_ctr & *_n*, const std::vector< annotation > & *_r*) [inline]

Full Constructor, setting the action descriptor to `a`, the new table rows to `_n`, and the annotations that have to be removed to `_r`.

Definition at line 64 of file table_delta.h.

10.266.3.3 coco::table_delta::table_delta (const table_delta & *__d*) [inline]

Standard Copy Constructor

Definition at line 69 of file table_delta.h.

10.266.3.4 coco::table_delta::table_delta (const std::string & *a*, const std::string & *__tn*, const dbt_row & *__t*) [inline]

Constructor, setting the action descriptor to `a`, and adding one row `___t` to the table `___tn`.

Definition at line 79 of file table_delta.h.

10.266.3.5 coco::table_delta::table_delta (const std::string & *a*, const std::string & *__tn*, const std::vector< dbt_row > & *__t*) [inline]

Constructor, setting the action descriptor to `a`, and adding a list of rows `___t` to the table `___tn`.

Definition at line 85 of file table_delta.h.

10.266.4 Member Function Documentation

10.266.4.1 void coco::table_delta::add (const t_line & _tl) [inline]

This method adds one table row.

Definition at line 95 of file table_delta.h.

10.266.4.2 void coco::table_delta::add (const std::string & _tn, const dbt_row & _r) [inline]

This method adds one row _r to the table _tn.

Definition at line 97 of file table_delta.h.

10.266.4.3 void coco::table_delta::add (const std::vector< t_line > & _tlv) [inline]

This method adds a list of table rows.

Definition at line 100 of file table_delta.h.

10.266.4.4 bool coco::table_delta::apply (work_node & _x, undelta_base *& _u, const delta_id & _did, size_t & delta_size) const [virtual]

Apply the delta with delta_id _d to work node _x. This will never be used, because the point_delta is converted to annotation_delta before the apply is performed.

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 35 of file table_delta.cc.

10.266.4.5 virtual bool coco::coco::delta_base::apply (work_node & _x, undelta_base *& _u, const delta_id & _di, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x. In this process the undo information for this delta is stored in _u.

Definition at line 198 of file search_graph.cc.

10.266.4.6 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.266.4.7 bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [inline, virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

Definition at line 88 of file api_delta.h.

10.266.4.8 virtual bool coco::coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d, size_t & delta_size) const [virtual, inherited]

Apply the delta with delta_id _d to work node _x, constructing in the process work_node _y, without changing _x. In this process the undo information for this delta is stored in _u. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the apply method.

10.266.4.9 void coco::table_delta::convert (work_node & _x, delta_base *& _u)

This method converts the table delta to an annotation_delta after the information is stored in the search database. The conversion is based on work_node _x, and the generated annotation_delta is returned via _u.

Definition at line 43 of file table_delta.cc.

10.266.4.10 virtual void coco::coco::delta_base::convert (work_node & _x, delta_base *& _d) [inline, virtual, inherited]

Convert this delta to a delta which can be stored in _x, this is e.g. used for all delta version, which are actually stored as annotation changes.

Definition at line 189 of file search_graph.cc.

10.266.4.11 virtual void coco::table_delta::create_table (work_node & _x, vdbl::standard_table *& ptb, const std::string & __t) const [inline, virtual]

This method creates the table with name ___t in the search database for the work_node _x. A pointer ptb to the created table is set. This method must be overloaded by the subclasses.

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 119 of file table_delta.h.

10.266.4.12 virtual void coco::table_delta::destroy_copy (delta_base * __d) const [inline, virtual]

Clone Destructor

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 114 of file table_delta.h.

10.266.4.13 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.266.4.14 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.266.4.15 const std::string& coco::coco::delta_base::get_action ( ) const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file search_graph.cc.

10.266.4.16 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.266.4.17 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.266.4.18 delta coco::coco::delta_base::make_delta (const std::string & *a*) [inline, inherited]

Construct a delta from this delta_base with the action `a`.

Definition at line 175 of file search_graph.cc.

10.266.4.19 virtual table_delta* coco::table_delta::new_copy ( ) const [inline, virtual]

Clone Operation

Reimplemented from coco::coco::delta_base.

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 111 of file table_delta.h.

10.266.4.20 bool coco::table_delta::operator!= (const delta_base & *_c*) const [inline]

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 138 of file table_delta.h.

10.266.4.21 bool coco::table_delta::operator!= (const table_delta & *_c*) const [inline]

Definition at line 144 of file table_delta.h.

10.266.4.22 bool coco::table_delta::operator== (const delta_base & *_c*) const [inline]

Comparison operators

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 135 of file table_delta.h.

10.266.4.23 bool coco::table_delta::operator== (const table_delta & _c) const [inline]

Comparison operators

Definition at line 143 of file table_delta.h.

10.266.4.24 void coco::table_delta::rm (const annotation & _tr) [inline]

This method adds the annotation `_tr` to the list of removed annotations.

Definition at line 104 of file table_delta.h.

10.266.4.25 void coco::table_delta::rm (const std::vector< annotation > & _trv) [inline]

This method adds the annotations in `_trv` to the list of removed annotations.

Definition at line 107 of file table_delta.h.

10.266.4.26 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.266.4.27 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.266.4.28 virtual void coco::coco::delta_base::unkeep ( ) [inline, virtual, inherited]

Perform this operation when the delta is released (unkept) from a work_node.

Reimplemented in coco::dag_delta.

Definition at line 194 of file search_graph.cc.

10.266.5 Member Data Documentation

10.266.5.1 std::string coco::coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file search_graph.cc.

The documentation for this class was generated from the following files:

- table_delta.h
- table_delta.cc

10.267 coco::termination_reason Class Reference

This class holds the reason of termination of inference and management modules.

`#include <termreason.h>`

Public Member Functions

- termination_reason ()
- termination_reason (int termr_r, const std::string &termr_ref)
- termination_reason (const termination_reason &__t)
- termination_reason & operator= (const termination_reason &__t)
- const std::string & get_message () const
- int get_code () const

Friends

- std::ostream & operator<< (std::ostream &o, const termination_reason &__x)

  C++ stream output operator for the termination_reason.

10.267.1 Detailed Description

This class is used to store the reason of termination of inference and management modules.

10.267.2 Constructor & Destructor Documentation

10.267.2.1 coco::termination_reason::termination_reason ( ) [inline]

This is a map for message translation. Standard Constructor

Definition at line 58 of file termreason.h.

10.267.2.2 coco::termination_reason::termination_reason (int *termr_r*, const std::string & *termr_ref*) [inline]

Constructor, setting the return code to `termr_r` and the message string to `termr_ref`.

Definition at line 62 of file termreason.h.

10.267.2.3 coco::termination_reason::termination_reason (const termination_reason & *__t*) [inline]

Standard Copy Constructor

Definition at line 66 of file termreason.h.

10.267.3 Member Function Documentation

10.267.3.1 int coco::termination_reason::get_code ( ) const [inline]

This method return the termination code.

Definition at line 81 of file termreason.h.

10.267.3.2 const std::string& coco::termination_reason::get_message ( ) const [inline]

This method return the termination message.

Definition at line 78 of file termreason.h.

10.267.3.3 termination_reason& coco::termination_reason::operator= (const termination_reason & __t) [inline]

Standard Assignment Operator

Definition at line 70 of file termreason.h.

10.267.4 Friends And Related Function Documentation

10.267.4.1 std::ostream& operator<< (std::ostream & *o*, const termination_reason & __x) [friend]

This operator writes a termination_reason (the message) to an `ostream`.

Definition at line 90 of file termreason.h.

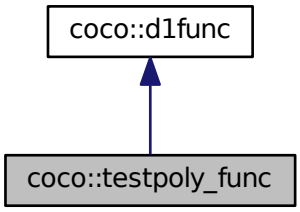
The documentation for this class was generated from the following file:

- termreason.h

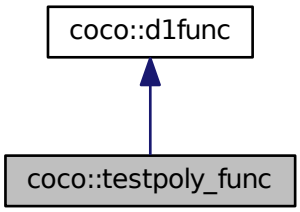
10.268 coco::testpoly_func Class Reference

`#include <d1testf.h>`

Inheritance diagram for coco::testpoly_func:



Collaboration diagram for coco::testpoly_func:



Public Types

- enum d1func_point_t { d1fp_unknown = -2, d1fp_regular = -1, d1fp_extremum = 0, d1fp_evenpoleup = 1, d1fp_evenpoledn = 2, d1fp_oddpoleup = 3, d1fp_oddpoledn = 4, d1fp_jumpldef = 5, d1fp_jumpludef = 6, d1fp_point = 7, d1fp_undefined = 8, d1fp_wild = 9, d1fp_jumprdef = 10, d1fp_jumprudef = 11 }
- enum d1solve_t { d1solve_0, d1solve_1, d1solve_2 }
- typedef std::triple< interval, interval, d1func_point_t > point_info
- typedef std::vector< point_info > point_list

Public Member Functions

- testpoly_func ()
- virtual interval imperfect_eval (const interval &x, unsigned int k) const =0
- virtual double eval (double x, unsigned int k) const =0

- virtual diffNumber eval (const diffNumber &x, unsigned int k) const
- virtual hessNumber eval (const hessNumber &x, unsigned int k) const
- virtual interval eval (const interval &x, unsigned int k) const
- virtual diffI eval (const diffI &x, unsigned int k) const
- virtual ihessNumber eval (const ihessNumber &x, unsigned int k) const
- virtual Islope eval (const Islope &x, unsigned int k) const
- virtual interval_set eval (const interval_set &x, unsigned int k) const
- virtual const char * func_name () const =0
- virtual uint64_t thash ()=0
- virtual uint64_t chash ()=0
- virtual std::string dag_out () const
- virtual std::ostream & print_add (std::ostream &o) const
- virtual std::string print_params () const
- virtual interval eval_slp (const interval &z, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &y, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &x, unsigned int k) const
- virtual void internal_eval_d1fp (const diffI &x, diffI &res, const std::vector< d1func_point_t > &ptin, std::vector< d1func_point_t > &ptres, unsigned int order) const
- virtual interval_set intersect_inv (const interval &x, const interval &r, unsigned int k) const
- virtual interval_set intersect_inv (const interval_set &x, const interval_set &r, unsigned int k) const
- virtual std::pair< double, double > evald (double x, unsigned int k) const
- virtual std::triple< double, double, double > evalt (double x, unsigned int k) const
- virtual void evalf (double x, double *r, unsigned int k) const
- virtual convex_info convexity (const interval &i, unsigned int k) const
- virtual d1func_monotonicity_t monotonicity (const interval &i, unsigned int k) const
- virtual int differentiability (const interval &i, unsigned int k) const
- virtual int degree_update (int in_degree) const
- virtual bool order_is_supported (unsigned int k) const
- virtual unsigned int supported_eval_order () const
- virtual const func_info & ff (unsigned int k) const
- virtual bool operator== (const d1func &b) const
- virtual bool operator!= (const d1func &b) const
- const char * toString (const d1func::d1func_point_t &t) const

Public Attributes

- volatile int MAXLEN

Protected Member Functions

- virtual interval imperfect_eval_f0 (const interval &x) const
- virtual interval imperfect_eval_f1 (const interval &x) const
- virtual interval imperfect_eval_f2 (const interval &x) const
- virtual void raise_order (unsigned int neworder)

Protected Attributes

- double dod_left
- double dod_right
- int basic_diff
- std::vector< func_info > _f

10.268.1 Member Typedef Documentation

10.268.1.1 typedef std::triple<interval, interval, d1func_point_t> coco::d1func::point_info [inherited]

The information on one point .first is the point, .second the value

Definition at line 110 of file d1func.h.

10.268.1.2 typedef std::vector<point_info> coco::d1func::point_list [inherited]

a list of points for function value, first and second derivatives

Definition at line 113 of file d1func.h.

10.268.2 Member Enumeration Documentation

10.268.2.1 enum coco::d1func::d1func_point_t [inherited]

The possible classes for special points in the lists: d1fp_regular: At this point the function is regular (for internal use only!) d1fp_extremum: The point is a local extremum d1fp_evenpoleup: At this point the function has a positive even pole d1fp_evenpoledn: At this point the function has a negative even pole d1fp_oddpoleup: At this point the function has an odd pole like 1/x at 0 d1fp_oddpoledn: At this point the function has an odd pole like -1/x at 0 d1fp_jumpldef: At this point the function has a jump (left side defined) d1fp_jumpludef: At this point the function has a jump (left side not defined) d1fp_point-: At this point the function has the specified value d1fp_undefined: In the given interval the function is undefined d1fp_wild: In the given interval the function has "wild" behaviour d1fp_jumprdef: At this point the function has a jump (right side defined) d1fp_jumprudef: At this point the function has a jump (right side not defined) d1fp_unknown: At this point the function has a unknown behaviour (for internal use only!)

Enumerator:

d1fp_unknown

d1fp_regular

d1fp_extremum

d1fp_evenpoleup

d1fp_evenpoledn

d1fp_oddpoleup

d1fp_oddpoledn

d1fp_jumpldef

d1fp_jumpludef

d1fp_point
d1fp_undefined
d1fp_wild
d1fp_jumprdef
d1fp_jumprudef

Definition at line 95 of file d1func.h.

10.268.2.2 enum coco::d1func::d1solve_t [inherited]

The possible functions for 1D Newton solving: d1solve_0: f(x) = c d1solve_1: f(x) - f(z) - f'(x)(x-z) = 0 d1solve_2: 1/(x-z)(f(x)-f(z))(1+(x-w)(x-z)) - (f(w)-f(z))/(w-z) - (x-w)/(x-z) f'(x) = 0

Enumerator:

d1solve_0
d1solve_1
d1solve_2

Definition at line 106 of file d1func.h.

10.268.3 Constructor & Destructor Documentation

10.268.3.1 coco::testpoly_func::testpoly_func ( ) [inline]

Definition at line 57 of file d1testf.h.

10.268.4 Member Function Documentation

10.268.4.1 virtual uint64_t coco::d1func::chash ( ) [pure virtual, inherited]

the const hash value for hash calculation in the DAG

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.268.4.2 convex_info coco::d1func::convexity (const interval & *i*, unsigned int *k*) const [virtual, inherited]

return convexity information on `i`.

Definition at line 2308 of file d1func.cc.

10.268.4.3 virtual std::string coco::d1func::dag_out ( ) const [inline, virtual, inherited]

the parameters written in DAG format (it must contain necessary ':'s)

Reimplemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

Definition at line 205 of file d1func.h.

10.268.4.4 virtual int coco::d1func::degree_update (int *in_degree*) const [inline, virtual, inherited]

return degree information for in-argument degree `in_degree`.

Definition at line 457 of file d1func.h.

10.268.4.5 int coco::d1func::differentiability (const interval & *x*, unsigned int *k*) const [virtual, inherited]

return differentiability information on `i`.

Reimplemented in coco::hsf_func, and coco::xexp_func.

Definition at line 2351 of file d1func.cc.

10.268.4.6 virtual double coco::d1func::eval (double *x*, unsigned int *k*) const [pure virtual, inherited]

the point evaluation

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.268.4.7 diffNumber coco::d1func::eval (const diffNumber & *x*, unsigned int *k*) const [virtual, inherited]

the diffNumber evaluation

Reimplemented in coco::dag_d1func.

Definition at line 349 of file d1func.cc.

10.268.4.8 hessNumber coco::d1func::eval (const hessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the hessNumber evaluation

Definition at line 334 of file d1func.cc.

10.268.4.9 interval coco::d1func::eval (const interval & *x_in*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 390 of file d1func.cc.

10.268.4.10 diffI coco::d1func::eval (const diffI & *x*, unsigned int *k*) const [virtual, inherited]

the interval diffNumber evaluation

Definition at line 715 of file d1func.cc.

10.268.4.11 ihessNumber coco::d1func::eval (const ihessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the interval hessNumber evaluation

the hessNumber evaluation

Definition at line 411 of file d1func.cc.

10.268.4.12 Islope coco::d1func::eval (const Islope & *x*, unsigned int *k*) const [virtual, inherited]

the interval slope evaluation

the interval diffNumber evaluation

Definition at line 429 of file d1func.cc.

10.268.4.13 interval_set coco::d1func::eval (const interval_set & *x*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 794 of file d1func.cc.

10.268.4.14 interval coco::d1func::eval_slp (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval first order slope evaluation at center z

Definition at line 1403 of file d1func.cc.

10.268.4.15 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *y*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at centers z and y

Definition at line 1984 of file d1func.cc.

10.268.4.16 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at double center z

Definition at line 1991 of file d1func.cc.

10.268.4.17 virtual std::pair<double,double> coco::d1func::evald (double *x*, unsigned int *k*) const [inline, virtual, inherited]

the point and derivative evaluation

Reimplemented in coco::xexp_func, and coco::dag_d1func.

Definition at line 437 of file d1func.h.

10.268.4.18 virtual void coco::d1func::evalf (double *x*, double * *r*, unsigned int *k*) const [inline, virtual, inherited]

the point, first to third derivative evaluation

Reimplemented in coco::xexp_func.

Definition at line 443 of file d1func.h.

10.268.4.19 virtual std::triple<double,double,double> coco::d1func::evalt (double *x*, unsigned int *k*) const [inline, virtual, inherited]

the point and first and second derivative evaluation

Reimplemented in coco::xexp_func, and coco::dag_d1func.

Definition at line 440 of file d1func.h.

10.268.4.20 virtual const func_info& coco::d1func::ff (unsigned int *k*) const [inline, virtual, inherited]

retrieve the k th special points list

Definition at line 469 of file d1func.h.

10.268.4.21 virtual const char* coco::d1func::func_name ( ) const [pure virtual, inherited]

the name of the d1func.

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.268.4.22 virtual interval coco::d1func::imperfect_eval (const interval & *x*, unsigned int *k*) const [pure virtual, inherited]

the imperfect evaluation of k th derivative.

Implemented in coco::xexp_func, coco::hsf_func, and coco::dag_d1func.

10.268.4.23 virtual interval coco::testpoly_func::imperfect_eval_f0 (const interval & *x*) const [inline, protected, virtual]

the imperfect function evaluation

Definition at line 41 of file d1testf.h.

10.268.4.24 virtual interval coco::testpoly_func::imperfect_eval_f1 (const interval & *x*) const [inline, protected, virtual]

the imperfect first derivative evaluation

Definition at line 46 of file d1testf.h.

10.268.4.25 virtual interval coco::testpoly_func::imperfect_eval_f2 (const interval & *x*) const [inline, protected, virtual]

the imperfect second derivative evaluation

Definition at line 51 of file d1testf.h.

10.268.4.26 void coco::d1func::internal_eval_d1fp (const diffI & *x*, diffI & *res*, const std::vector< d1func_point_t > & *ptin*, std::vector< d1func_point_t > & *ptres*, unsigned int *k*) const [virtual, inherited]

perform internal evaluations to update eventual point lists of "higher" functions.

Definition at line 168 of file d1func.cc.

10.268.4.27 interval_set coco::d1func::intersect_inv (const interval & *x*, const interval & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1708 of file d1func.cc.

10.268.4.28 interval_set coco::d1func::intersect_inv (const interval_set & *x*, const interval_set & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1778 of file d1func.cc.

10.268.4.29 d1func_monotonicity_t coco::d1func::monotonicity (const interval & *x*, unsigned int *k*) const [virtual, inherited]

return monotonicity information on `i`.

Definition at line 2185 of file d1func.cc.

10.268.4.30 virtual bool coco::d1func::operator!= (const d1func & *b*) const [inline, virtual, inherited]

the other comparison operator

Definition at line 476 of file d1func.h.

10.268.4.31 virtual bool coco::d1func::operator== (const d1func & *b*) const [inline, virtual, inherited]

the comparison operator

Reimplemented in coco::hsf_func, coco::xexp_func, and coco::dag_d1func.

Definition at line 473 of file d1func.h.

10.268.4.32 virtual bool coco::d1func::order_is_supported (unsigned int *k*) const [inline, virtual, inherited]

check whether evaluation of this order is supported.

Definition at line 460 of file d1func.h.

10.268.4.33 virtual std::ostream& coco::d1func::print_add (std::ostream & *o*) const [inline, virtual, inherited]

additional info written in DAG format (it must contain dag lines)

Reimplemented in coco::dag_d1func.

Definition at line 208 of file d1func.h.

10.268.4.34 virtual std::string coco::d1func::print_params ( ) const [inline, virtual, inherited]

the parameters written in printable form

Reimplemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

Definition at line 211 of file d1func.h.

10.268.4.35 virtual void coco::d1func::raise_order (unsigned int *neworder*) [inline, protected, virtual, inherited]

raise the order of the supported function evaluation to `neworder`

Reimplemented in coco::hsf_func, and coco::xexp_func.

Definition at line 215 of file d1func.h.

10.268.4.36 virtual unsigned int coco::d1func::supported_eval_order ( ) const [inline, virtual, inherited]

return the maximal evaluation order supported.

Definition at line 466 of file d1func.h.

10.268.4.37 virtual uint64_t coco::d1func::thash ( ) [pure virtual, inherited]

the pure hash value for hash calculation in the DAG

Implemented in coco::dag_d1func, coco::hsf_func, and coco::xexp_func.

10.268.4.38 const char * coco::d1func::toString (const d1func::d1func_point_t & *t*) const [inherited]

Definition at line 2415 of file d1func.cc.

10.268.5 Member Data Documentation

10.268.5.1 std::vector<func_info> coco::d1func::_f [protected, inherited]

the function info tables for function values and derivatives.

Definition at line 163 of file d1func.h.

10.268.5.2 int coco::d1func::basic_diff [protected, inherited]

basic differentiability

Definition at line 160 of file d1func.h.

10.268.5.3 double coco::d1func::dod_left [protected, inherited]

the left limit of the DOD

Definition at line 156 of file d1func.h.

10.268.5.4 double coco::d1func::dod_right [protected, inherited]

the right limit of the DOD

Definition at line 158 of file d1func.h.

10.268.5.5 volatile int coco::d1func::MAXLEN [inherited]

maximal lenght of the interval set returned

Definition at line 167 of file d1func.h.

The documentation for this class was generated from the following file:

- d1testf.h

10.269 coco::thirdderBackwardEvaluator Class Reference

`#include <thirdder_evaluator.h>`

Inheritance diagram for coco::thirdderBackwardEvaluator:

Collaboration diagram for coco::thirdderBackwardEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- thirdderBackwardEvaluator (std::vector< std::vector< double > > *__der_data, std::vector< std::vector< double > > *__hessv_data, std::vector< std::vector< double > > *__hessw_data, std::vector< std::vector< double > > *__thirdder_data, variable_indicator &__v, const model &__m, std::vector< std::vector< double > > *__d, std::vector< std::vector< double > > *__hv, std::vector< std::vector< double > > *__hw, std::vector< std::vector< double > > *__t, std::vector<

double > *__grad, std::vector< double > *__hessv, std::vector< double > *__hessw, std::vector< double > *__thirdder)
- thirdderBackwardEvaluator (const thirdderBackwardEvaluator &__he)
- ~thirdderBackwardEvaluator ()
- void newPoint (std::vector< std::vector< double > > *__der_data, std::vector< std::vector< double > > *__hessv_data, std::vector< std::vector< double > > *__hessw_data, std::vector< std::vector< double > > *__thirdder_data, const variable_indicator &__v)
- void newResult (std::vector< double > *__grad, std::vector< double > *__hessv, std::vector< double > *__hessw, std::vector< double > *__thirdder)
- void set_mult (double scal)
- expression_const_walker short_cut_to (const expression_node &__data)
- void initialize ()
- int calculate (const expression_node &__data)
- void cleanup (const expression_node &__data)
- int update (const bool &__rval)
- int update (const expression_node &__data, const bool &__rval)
- bool calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int calculate (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual int update (const return_value &__rval)
- virtual int update (const node_data_type &__data, const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.269.1 Member Typedef Documentation

10.269.1.1 typedef _Base::const_walker coco::coco::cached_backward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 854 of file search_graph.cc.

10.269.1.2 typedef _Base::node_data_type coco::coco::cached_backward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 850 of file search_graph.cc.

10.269.1.3 typedef _Base::return_value coco::coco::cached_backward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 852 of file search_graph.cc.

10.269.2 Constructor & Destructor Documentation

10.269.2.1 coco::thirdderBackwardEvaluator::thirdderBackwardEvaluator (std::vector< std::vector< double > > * __der_data, std::vector< std::vector< double > > * __hessv_data, std::vector< std::vector< double > > * __hessw_data, std::vector< std::vector< double > > * __thirdder_data, variable_indicator & __v, const model & __m, std::vector< std::vector< double > > * __d, std::vector< std::vector< double > > * __hv, std::vector< std::vector< double > > * __hw, std::vector< std::vector< double > > * __t, std::vector< double > * __grad, std::vector< double > * __hessv, std::vector< double > * __hessw, std::vector< double > * __thirdder) [inline]

Definition at line 1327 of file thirdder_evaluator.h.

10.269.2.2 coco::thirdderBackwardEvaluator::thirdderBackwardEvaluator (const thirdderBackwardEvaluator & __he) [inline]

Definition at line 1375 of file thirdder_evaluator.h.

10.269.2.3 coco::thirdderBackwardEvaluator::~thirdderBackwardEvaluator ( ) [inline]

Definition at line 1377 of file thirdder_evaluator.h.

10.269.3 Member Function Documentation

10.269.3.1 virtual int coco::coco::cached_backward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right before all children of a node are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 922 of file search_graph.cc.

10.269.3.2 int coco::thirdderBackwardEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 1428 of file thirdder_evaluator.h.

10.269.3.3 bool coco::thirdderBackwardEvaluator::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter `eval_all` is `true` whether the node is a virtual node.

Reimplemented from coco::coco::cached_backward_evaluator_base< thirdderBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1669 of file thirdder_evaluator.h.

10.269.3.4 virtual void coco::coco::cached_backward_evaluator_base::cleanup (const node_data_type & *__data*) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The `__data` parameter contains the node data of the graph node being visited.

Definition at line 926 of file search_graph.cc.

10.269.3.5 void coco::thirdderBackwardEvaluator::cleanup (const expression_node & *__data*) [inline]

Definition at line 1557 of file thirdder_evaluator.h.

10.269.3.6 int coco::coco::cached_backward_evaluator_base::collect (const node_data_type & *__data*, const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 880 of file search_graph.cc.

10.269.3.7 void coco::thirdderBackwardEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_backward_evaluator_base< thirdderBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1417 of file thirdder_evaluator.h.

10.269.3.8 bool coco::thirdderBackwardEvaluator::is_cached (const node_data_type & *__data*) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_backward_evaluator_base< thirdderBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1314 of file thirdder_evaluator.h.

10.269.3.9 void coco::thirdderBackwardEvaluator::newPoint (std::vector< std::vector< double > > * __der_data, std::vector< std::vector< double > > * __hessv_data, std::vector< std::vector< double > > * __hessw_data, std::vector< std::vector< double > > * __thirdder_data, const variable_indicator & __v) [inline]

Definition at line 1379 of file thirdder_evaluator.h.

10.269.3.10 void coco::thirdderBackwardEvaluator::newResult (std::vector< double > * __grad, std::vector< double > * __hessv, std::vector< double > * __hessw, std::vector< double > * __thirdder) [inline]

Definition at line 1395 of file thirdder_evaluator.h.

10.269.3.11 void coco::coco::cached_backward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into a call to cleanup.

Definition at line 875 of file search_graph.cc.

10.269.3.12 int coco::coco::cached_backward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Definition at line 863 of file search_graph.cc.

10.269.3.13 virtual void coco::coco::cached_backward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 930 of file search_graph.cc.

10.269.3.14 void coco::thirdderBackwardEvaluator::set_mult (double scal) [inline]

Definition at line 1404 of file thirdder_evaluator.h.

10.269.3.15 expression_const_walker coco::thirdderBackwardEvaluator::short_cut_to (const expression_node & __data) [inline]

Definition at line 1413 of file thirdder_evaluator.h.

10.269.3.16 virtual int coco::coco::cached_backward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 941 of file search_graph.cc.

10.269.3.17 virtual int coco::coco::cached_backward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 952 of file search_graph.cc.

10.269.3.18 int coco::thirdderBackwardEvaluator::update (const bool & __rval) [inline]

Definition at line 1595 of file thirdder_evaluator.h.

10.269.3.19 int coco::thirdderBackwardEvaluator::update (const expression_node & __data, const bool & __rval) [inline]

Definition at line 1605 of file thirdder_evaluator.h.

10.269.3.20 return_value coco::coco::cached_backward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 892 of file search_graph.cc.

10.269.3.21 int coco::coco::cached_backward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 886 of file search_graph.cc.

10.269.3.22 void coco::coco::cached_backward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 901 of file search_graph.cc.

10.269.3.23 return_value coco::coco::cached_backward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 897 of file search_graph.cc.

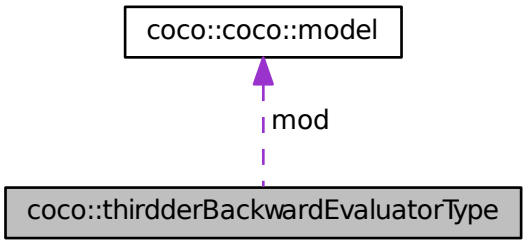
The documentation for this class was generated from the following file:

- thirdder_evaluator.h

10.270 coco::thirdderBackwardEvaluatorType Struct Reference

`#include <thirdder_evaluator.h>`

Collaboration diagram for coco::thirdderBackwardEvaluatorType:



Public Attributes

- std::vector< std::vector < double > > * d_data
- std::vector< std::vector < double > > * hv_data
- std::vector< std::vector < double > > * hw_data
- std::vector< std::vector < double > > * t_data
- std::vector< std::vector < double > > * d_cache
- std::vector< std::vector < double > > * hv_cache
- std::vector< std::vector < double > > * hw_cache
- std::vector< std::vector < double > > * t_cache
- std::vector< double > * grad_vec
- std::vector< double > * hessv_vec
- std::vector< double > * hessw_vec
- std::vector< double > * thirdder_vec
- bool calc_grad

- bool calc_hessv
- bool calc_hessw
- const model * mod
- double d_mult
- double d_mult_trans
- double hv_mult
- double hv_mult_trans
- double hw_mult
- double hw_mult_trans
- double t_mult
- double t_mult_trans
- bool is_linear
- unsigned int child_n

10.270.1 Member Data Documentation

10.270.1.1 bool coco::thirdderBackwardEvaluatorType::calc_grad

Definition at line 1289 of file thirdder_evaluator.h.

10.270.1.2 bool coco::thirdderBackwardEvaluatorType::calc_hessv

Definition at line 1290 of file thirdder_evaluator.h.

10.270.1.3 bool coco::thirdderBackwardEvaluatorType::calc_hessw

Definition at line 1291 of file thirdder_evaluator.h.

10.270.1.4 unsigned int coco::thirdderBackwardEvaluatorType::child_n

Definition at line 1302 of file thirdder_evaluator.h.

10.270.1.5 std::vector<std::vector<double> >* coco::thirdderBackwardEvaluatorType::d_cache

Definition at line 1281 of file thirdder_evaluator.h.

10.270.1.6 std::vector<std::vector<double> >* coco::thirdderBackwardEvaluatorType::d_data

Definition at line 1277 of file thirdder_evaluator.h.

10.270.1.7 double coco::thirdderBackwardEvaluatorType::d_mult

Definition at line 1293 of file thirdder_evaluator.h.

10.270.1.8 double coco::thirdderBackwardEvaluatorType::d_mult_trans

Definition at line 1294 of file thirdder_evaluator.h.

10.270.1.9 std::vector<double>* coco::thirdderBackwardEvaluatorType::grad_vec

Definition at line 1285 of file thirdder_evaluator.h.

10.270.1.10 std::vector<double>* coco::thirdderBackwardEvaluatorType::hessv_vec

Definition at line 1286 of file thirdder_evaluator.h.

10.270.1.11 std::vector<double>* coco::thirdderBackwardEvaluatorType::hessw_vec

Definition at line 1287 of file thirdder_evaluator.h.

10.270.1.12 std::vector<std::vector<double> >* coco::thirdderBackwardEvaluatorType::hv_cache

Definition at line 1282 of file thirdder_evaluator.h.

10.270.1.13 std::vector<std::vector<double> >* coco::thirdderBackwardEvaluatorType::hv_data

Definition at line 1278 of file thirdder_evaluator.h.

10.270.1.14 double coco::thirdderBackwardEvaluatorType::hv_mult

Definition at line 1295 of file thirdder_evaluator.h.

10.270.1.15 double coco::thirdderBackwardEvaluatorType::hv_mult_trans

Definition at line 1296 of file thirdder_evaluator.h.

10.270.1.16 std::vector<std::vector<double> >* coco::thirdderBackwardEvaluatorType::hw_cache

Definition at line 1283 of file thirdder_evaluator.h.

10.270.1.17 std::vector<std::vector<double> >* coco::thirdderBackwardEvaluatorType::hw_data

Definition at line 1279 of file thirdder_evaluator.h.

10.270.1.18 double coco::thirdderBackwardEvaluatorType::hw_mult

Definition at line 1297 of file thirdder_evaluator.h.

10.270.1.19 double coco::thirdderBackwardEvaluatorType::hw_mult_trans

Definition at line 1298 of file thirdder_evaluator.h.

10.270.1.20 bool coco::thirdderBackwardEvaluatorType::is_linear

Definition at line 1301 of file thirdder_evaluator.h.

10.270.1.21 const model* coco::thirdderBackwardEvaluatorType::mod

Definition at line 1292 of file thirdder_evaluator.h.

10.270.1.22 std::vector<std::vector<double> >* coco::thirdderBackwardEvaluatorType::t_cache

Definition at line 1284 of file thirdder_evaluator.h.

10.270.1.23 std::vector<std::vector<double> >* coco::thirdderBackwardEvaluatorType::t_data

Definition at line 1280 of file thirdder_evaluator.h.

10.270.1.24 double coco::thirdderBackwardEvaluatorType::t_mult

Definition at line 1299 of file thirdder_evaluator.h.

10.270.1.25 double coco::thirdderBackwardEvaluatorType::t_mult_trans

Definition at line 1300 of file thirdder_evaluator.h.

10.270.1.26 std::vector<double>* coco::thirdderBackwardEvaluatorType::thirdder_vec

Definition at line 1288 of file thirdder_evaluator.h.

The documentation for this struct was generated from the following file:

- thirdder_evaluator.h

10.271 coco::thirdderForwardEvaluator Class Reference

`#include <thirdder_evaluator.h>`

Inheritance diagram for coco::thirdderForwardEvaluator:

Collaboration diagram for coco::thirdderForwardEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- thirdderForwardEvaluator (const std::vector< hessNumber > &__x, const variable_indicator &__v, const model &__m, std::vector< std::vector< double > > &__d, std::vector< std::vector< double > > &__hv, std::vector< std::vector< double > > &__hw, std::vector< std::vector< double > > &__t, std::vector< hessNumber > *__c)
- thirdderForwardEvaluator (const thirdderForwardEvaluator &__x)

- ~thirdderForwardEvaluator ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void newPoint (const std::vector< hessNumber > &__x, const variable_indicator &__v)
- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const hessNumber &__rval)
- int update (const expression_node &__data, const thirdderForwardEvaluatorReturnValue &__rval)
- thirdderForwardEvaluatorReturnValue calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.271.1 Member Typedef Documentation

10.271.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.271.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.271.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.271.2 Constructor & Destructor Documentation

10.271.2.1 coco::thirdderForwardEvaluator::thirdderForwardEvaluator (const std::vector< hessNumber > & __x, const variable_indicator & __v, const model & __m, std::vector< std::vector< double > > & __d, std::vector< std::vector< double > > & __hv, std::vector< std::vector< double > > & __hw, std::vector< hessNumber > * __c) [inline]

Definition at line 257 of file thirdder_evaluator.h.

10.271.2.2 coco::thirdderForwardEvaluator::thirdderForwardEvaluator (const thirdderForwardEvaluator & __x) [inline]

Definition at line 283 of file thirdder_evaluator.h.

10.271.2.3 coco::thirdderForwardEvaluator::~thirdderForwardEvaluator ( ) [inline]

Definition at line 287 of file thirdder_evaluator.h.

10.271.3 Member Function Documentation

10.271.3.1 void coco::thirdderForwardEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 391 of file thirdder_evaluator.h.

10.271.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.271.3.3 thirdderForwardEvaluatorReturnValue coco::thirdderForwardEvaluator::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< thirdderForwardEvaluatorType, expression_node, thirdderForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 1266 of file thirdder_evaluator.h.

10.271.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.271.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.271.3.6 void coco::thirdderForwardEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< thirdderForwardEvaluatorType, expression_node, thirdderForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 305 of file thirdder_evaluator.h.

10.271.3.7 int coco::thirdderForwardEvaluator::initialize (const expression_node & __data) [inline]

Definition at line 313 of file thirdder_evaluator.h.

10.271.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.271.3.9 bool coco::thirdderForwardEvaluator::is_cached (const node_data_type & __data) [inline, protected, virtual]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Reimplemented from coco::coco::cached_forward_evaluator_base< thirdderForwardEvaluatorType, expression_node, thirdderForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 191 of file thirdder_evaluator.h.

10.271.3.10 void coco::thirdderForwardEvaluator::newPoint (const std::vector< hessNumber > & __x, const variable_indicator & __v) [inline]

Definition at line 292 of file thirdder_evaluator.h.

10.271.3.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.271.3.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.271.3.13 void coco::thirdderForwardEvaluator::retrieve_from_cache (const expression_node & __data) [inline]

Definition at line 400 of file thirdder_evaluator.h.

10.271.3.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.271.3.15 expression_const_walker coco::thirdderForwardEvaluator::short_cut_to (const expression_node & __data) [inline]

Definition at line 289 of file thirdder_evaluator.h.

10.271.3.16 int coco::thirdderForwardEvaluator::update (const hessNumber & __rval) [inline]

Definition at line 407 of file thirdder_evaluator.h.

10.271.3.17 int coco::thirdderForwardEvaluator::update (const expression_node & __data, const thirdderForwardEvaluatorReturnValue & __rval) [inline]

Definition at line 417 of file thirdder_evaluator.h.

10.271.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.271.3.19 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The __data parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.271.3.20 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter false.

Definition at line 763 of file search_graph.cc.

10.271.3.21 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.271.3.22 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.271.3.23 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter true.

Definition at line 768 of file search_graph.cc.

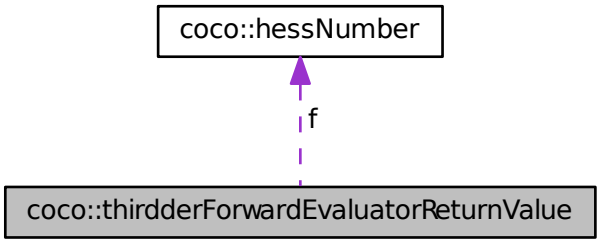
The documentation for this class was generated from the following file:

- thirdder_evaluator.h

10.272 coco::thirdderForwardEvaluatorReturnValue Struct Reference

#include <thirdder_evaluator.h>

Collaboration diagram for coco::thirdderForwardEvaluatorReturnValue:



Public Attributes

- hessNumber f
- unsigned int nn
- tristate in_chn

10.272.1 Member Data Documentation

10.272.1.1 hessNumber coco::thirdderForwardEvaluatorReturnValue::f

Definition at line 159 of file thirdder_evaluator.h.

10.272.1.2 tristate coco::thirdderForwardEvaluatorReturnValue::in_chn

Definition at line 162 of file thirdder_evaluator.h.

10.272.1.3 unsigned int coco::thirdderForwardEvaluatorReturnValue::nn

Definition at line 161 of file thirdder_evaluator.h.

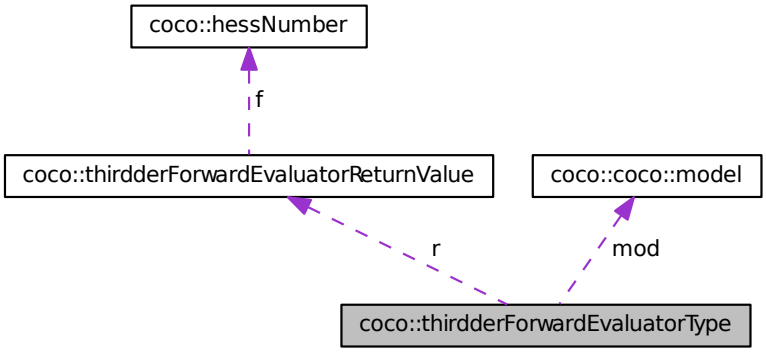
The documentation for this struct was generated from the following file:

- thirdder_evaluator.h

10.273 coco::thirdderForwardEvaluatorType Struct Reference

```
#include <thirdder_evaluator.h>
```

Collaboration diagram for coco::thirdderForwardEvaluatorType:



Public Attributes

- const std::vector< hessNumber > * x
- std::vector< hessNumber > * f_cache
- std::vector< std::vector < double > > * t_data
- std::vector< std::vector < double > > * hv_data
- std::vector< std::vector < double > > * hw_data
- std::vector< std::vector < double > > * d_data
- std::vector< unsigned int > * t
- std::vector< bool > * b
- thirdderForwardEvaluatorReturnValue r
- const model * mod
- union {
  void * p
  double d
  } u

- unsigned int n
- unsigned int info

10.273.1 Member Data Documentation

10.273.1.1 std::vector<bool>* coco::thirdderForwardEvaluatorType::b

Definition at line 174 of file thirdder_evaluator.h.

10.273.1.2 double coco::thirdderForwardEvaluatorType::d

Definition at line 177 of file thirdder_evaluator.h.

10.273.1.3 std::vector<std::vector<double> >* coco::thirdderForwardEvaluatorType::d_data

Definition at line 172 of file thirdder_evaluator.h.

10.273.1.4 std::vector<hessNumber>* coco::thirdderForwardEvaluatorType::f_cache

Definition at line 168 of file thirdder_evaluator.h.

10.273.1.5 std::vector<std::vector<double> >* coco::thirdderForwardEvaluatorType::hv_data

Definition at line 170 of file thirdder_evaluator.h.

10.273.1.6 std::vector<std::vector<double> >* coco::thirdderForwardEvaluatorType::hw_data

Definition at line 171 of file thirdder_evaluator.h.

10.273.1.7 unsigned int coco::thirdderForwardEvaluatorType::info

Definition at line 178 of file thirdder_evaluator.h.

10.273.1.8 const model* coco::thirdderForwardEvaluatorType::mod

Definition at line 176 of file thirdder_evaluator.h.

10.273.1.9 unsigned int coco::thirdderForwardEvaluatorType::n

Definition at line 178 of file thirdder_evaluator.h.

10.273.1.10 void* coco::thirdderForwardEvaluatorType::p

Definition at line 177 of file thirdder_evaluator.h.

10.273.1.11 thirdderForwardEvaluatorReturnValue coco::thirdderForwardEvaluatorType::r

Definition at line 175 of file thirdder_evaluator.h.

10.273.1.12 std::vector<unsigned int>* coco::thirdderForwardEvaluatorType::t

Definition at line 173 of file thirdder_evaluator.h.

10.273.1.13 std::vector<std::vector<double> >* coco::thirdderForwardEvaluatorType::t_data

Definition at line 169 of file thirdder_evaluator.h.

10.273.1.14 union { ... } coco::thirdderForwardEvaluatorType::u

10.273.1.15 const std::vector<hessNumber>* coco::thirdderForwardEvaluatorType::x

Definition at line 167 of file thirdder_evaluator.h.

The documentation for this struct was generated from the following file:

- thirdder_evaluator.h

10.274 coco::thirdderPreparationEvaluator Class Reference

`#include <thirdder_evaluator.h>`

Inheritance diagram for coco::thirdderPreparationEvaluator:

Collaboration diagram for coco::thirdderPreparationEvaluator:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- thirdderPreparationEvaluator (std::vector< std::vector< double > > &__d, std::vector< std::vector< double > > &__hv, std::vector< std::vector< double > > &__hw, std::vector< std::vector< double > > &__t, unsigned int _num_of_nodes)
- thirdderPreparationEvaluator (const thirdderPreparationEvaluator &__x)
- ~thirdderPreparationEvaluator ()
- void initialize ()
- bool is_cached (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- int update (bool __rval)
- int update (const expression_node &__data, bool __rval)
- bool calculate_value (bool eval_all)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()
- void vinit ()
- virtual bool is_cached (const node_data_type &__data)
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

10.274.1 Member Typedef Documentation

10.274.1.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.274.1.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.274.1.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.274.2 Constructor & Destructor Documentation

10.274.2.1 coco::thirdderPreparationEvaluator::thirdderPreparationEvaluator (std::vector< std::vector< double > > & __d, std::vector< std::vector< double > > & __hv, std::vector< std::vector< double > > & __hw, std::vector< std::vector< double > > & __t, unsigned int _num_of_nodes) [inline]

Definition at line 88 of file thirdder_evaluator.h.

10.274.2.2 coco::thirdderPreparationEvaluator::thirdderPreparationEvaluator (const thirdderPreparationEvaluator & __x) [inline]

Definition at line 108 of file thirdder_evaluator.h.

10.274.2.3 coco::thirdderPreparationEvaluator::~thirdderPreparationEvaluator ( ) [inline]

Definition at line 114 of file thirdder_evaluator.h.

10.274.3 Member Function Documentation

10.274.3.1 void coco::thirdderPreparationEvaluator::calculate (const expression_node & __data) [inline]

Definition at line 139 of file thirdder_evaluator.h.

10.274.3.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The __data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.274.3.3 bool coco::thirdderPreparationEvaluator::calculate_value (bool *eval_all*) [inline, virtual]

This method is called last for every graph node, and it produces the return value of the visitor. The parameter eval_all is true whether the node is a virtual node.

Reimplemented from coco::coco::cached_forward_evaluator_base< thirdderPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 151 of file thirdder_evaluator.h.

10.274.3.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.274.3.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.274.3.6 void coco::thirdderPreparationEvaluator::initialize ( ) [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::coco::cached_forward_evaluator_base< thirdderPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 116 of file thirdder_evaluator.h.

10.274.3.7 int coco::thirdderPreparationEvaluator::initialize (const expression_node & __data) [inline]

Definition at line 125 of file thirdder_evaluator.h.

10.274.3.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.274.3.9 bool coco::thirdderPreparationEvaluator::is_cached (const expression_node & *__data*) [inline]

Definition at line 118 of file thirdder_evaluator.h.

10.274.3.10 virtual bool coco::coco::cached_forward_evaluator_base::is_cached (const node_data_type & *__data*) [inline, virtual, inherited]

This method checks whether the return value for this method is in the cache or can be calculated without visiting the children.

Definition at line 777 of file search_graph.cc.

10.274.3.11 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.274.3.12 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & *__data*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.274.3.13 void coco::thirdderPreparationEvaluator::retrieve_from_cache (const expression_node & *__data*) [inline]

Definition at line 123 of file thirdder_evaluator.h.

10.274.3.14 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & *__data*) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.274.3.15 int coco::thirdderPreparationEvaluator::update (bool *__rval*) [inline]

Definition at line 141 of file thirdder_evaluator.h.

10.274.3.16 int coco::thirdderPreparationEvaluator::update (const expression_node & *__data*, bool *__rval*) [inline]

Definition at line 143 of file thirdder_evaluator.h.

10.274.3.17 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & *__data*, const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.274.3.18 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & *__rval*) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.274.3.19 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.274.3.20 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & *__rval*) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.274.3.21 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.274.3.22 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

- thirdder_evaluator.h

10.275 coco::thirdderPreparationEvaluatorType Struct Reference

`#include <thirdder_evaluator.h>`

Public Attributes

- std::vector< std::vector < double > > * d
- std::vector< std::vector < double > > * hv
- std::vector< std::vector < double > > * hw
- std::vector< std::vector < double > > * t

10.275.1 Member Data Documentation

10.275.1.1 std::vector<std::vector<double> >* coco::thirdderPreparationEvaluatorType::d

Definition at line 74 of file thirdder_evaluator.h.

10.275.1.2 std::vector<std::vector<double> >* coco::thirdderPreparationEvaluatorType::hv

Definition at line 75 of file thirdder_evaluator.h.

10.275.1.3 std::vector<std::vector<double> >* coco::thirdderPreparationEvaluatorType::hw

Definition at line 76 of file thirdder_evaluator.h.

10.275.1.4 std::vector<std::vector<double> >* coco::thirdderPreparationEvaluatorType::t

Definition at line 77 of file thirdder_evaluator.h.

The documentation for this struct was generated from the following file:

- thirdder_evaluator.h

10.276 std::triple Struct Reference

triple holds three objects of arbitrary type.

`#include <stlp_triple.h>`

Public Types

- typedef _T1 first_type
  type of first entry
- typedef _T2 second_type
  type of second entry
- typedef _T3 third_type
  type of third entry

Public Member Functions

- triple ()
- triple (const _T1 &__a, const _T2 &__b, const _T3 &__c)
- template<class _U1 , class _U2 , class _U3 >
  triple (const triple< _U1, _U2, _U3 > &__t)

Public Attributes

- _T1 first
  first entry
- _T2 second
  second entry
- _T3 third
  third entry

10.276.1 Detailed Description

This class is used to hold three objects of arbitrary types. It is a slight generalization of `std::pair`.

10.276.2 Member Typedef Documentation

10.276.2.1 typedef _T1 std::triple::first_type

`first_type` is the first bound type

Definition at line 44 of file stlp_triple.h.

10.276.2.2 typedef _T2 std::triple::second_type

`second_type` is the second bound type

Definition at line 46 of file stlp_triple.h.

10.276.2.3 typedef _T3 std::triple::third_type

`third_type` is the third bound type

Definition at line 48 of file stlp_triple.h.

10.276.3 Constructor & Destructor Documentation

10.276.3.1 std::triple::triple ( ) [inline]

Definition at line 64 of file stlp_triple.h.

10.276.3.2 std::triple::triple (const _T1 & __a, const _T2 & __b, const _T3 & __c) [inline]

Three objects may be passed to a `triple` constructor to be copied.

Definition at line 69 of file stlp_triple.h.

10.276.3.3 template<class _U1 , class _U2 , class _U3 > std::triple::triple (const triple< _U1, _U2, _U3 > & __t) [inline]

There is also a templated copy constructor for the `triple` class itself.

Definition at line 76 of file stlp_triple.h.

10.276.4 Member Data Documentation

10.276.4.1 _T1 std::triple::first

`first` is a copy of the first object

Definition at line 51 of file stlp_triple.h.

10.276.4.2 _T2 std::triple::second

`second` is a copy of the second object

Definition at line 53 of file stlp_triple.h.

10.276.4.3 _T3 std::triple::third

`third` is a copy of the second object

Definition at line 55 of file stlp_triple.h.

The documentation for this struct was generated from the following file:

- stlp_triple.h

10.277 coco::undelta Class Reference

The undelta class (undo of updates to work nodes)

`#include <api_deltabase.h>`

Public Member Functions

- undelta (size_t s)
- undelta (undelta_base *__d, size_t s)
- undelta (const undelta &__d)
- ~undelta ()
- const undelta_base * get_base () const
- bool unapply (work_node &_x, const delta_id &_i) const
- bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_i) const
- undelta & operator= (const undelta &_u)
- size_t delta_size () const

10.277.1 Detailed Description

The undelta class is used to store undo information in the work nodes. It is designed as a wrapper class in order to overcome the problem that copy constructors cannot be overloaded.

10.277.2 Constructor & Destructor Documentation

10.277.2.1 coco::undelta::undelta (size_t *s*) [inline]

Standard Constructor

Definition at line 234 of file api_deltabase.h.

10.277.2.2 coco::undelta::undelta (undelta_base * *__d*, size_t *s*) [inline]

Internal Constructor, which just copies the pointer to the undelta_base. This pointer must be allocated by `new`, since it will be removed with `delete` in the destructor

Definition at line 238 of file api_deltabase.h.

10.277.2.3 coco::undelta::undelta (const undelta & *__d*)

Copy Constructor, which constructs a new undelta from an existing undelta. The clone operation for the undelta_base will be called in that process, effectively overloading the copy constructor for the undelta_base.

10.277.2.4 coco::undelta::~undelta ( )

Destructor, which frees the `_d` using `delete`

10.277.3 Member Function Documentation

10.277.3.1 size_t coco::undelta::delta_size ( ) const

Returns the size of the delta stored in the database

10.277.3.2 const undelta_base* coco::undelta::get_base ( ) const

Return the undelta_base stored in this wrapper

10.277.3.3 undelta& coco::undelta::operator= (const undelta & _u)

Assignment operator, which uses the clone operation for the undelta_base, effectively overloading the assignment operator for the undelta_base.

10.277.3.4 bool coco::undelta::unapply (work_node & _x, const delta_id & _i) const

Undo the delta with delta_id _i in work node _x

10.277.3.5 bool coco::undelta::unapply3 (work_node & _x, const work_node & _y, const delta_id & _i) const

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x.

The documentation for this class was generated from the following file:

- api_deltabase.h

10.278 coco::coco::undelta Class Reference

The undelta class (undo of updates to work nodes)

Public Member Functions

- undelta (size_t s)
- undelta (undelta_base *__d, size_t s)
- undelta (const undelta &__d)
- ~undelta ()
- const undelta_base * get_base () const
- bool unapply (work_node &_x, const delta_id &_i) const
- bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_i) const
- undelta & operator= (const undelta &_u)
- size_t delta_size () const
- undelta (size_t s)
- undelta (undelta_base *__d, size_t s)
- undelta (const undelta &__d)
- ~undelta ()
- const undelta_base * get_base () const
- bool unapply (work_node &_x, const delta_id &_i) const
- bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_i) const
- undelta & operator= (const undelta &_u)
- size_t delta_size () const
- undelta (size_t s)
- undelta (undelta_base *__d, size_t s)

- undelta (const undelta &__d)
- ~undelta ()
- const undelta_base * get_base () const
- bool unapply (work_node &_x, const delta_id &_i) const
- bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_i) const
- undelta & operator= (const undelta &_u)
- size_t delta_size () const

10.278.1 Detailed Description

The undelta class is used to store undo information in the work nodes. It is designed as a wrapper class in order to overcome the problem that copy constructors cannot be overloaded.

Definition at line 224 of file search_graph.cc.

10.278.2 Constructor & Destructor Documentation

10.278.2.1 coco::coco::undelta::undelta (size_t *s*) [inline]

Standard Constructor

Definition at line 234 of file search_graph.cc.

10.278.2.2 coco::coco::undelta::undelta (undelta_base * *__d*, size_t *s*) [inline]

Internal Constructor, which just copies the pointer to the undelta_base. This pointer must be allocated by `new`, since it will be removed with `delete` in the destructor

Definition at line 238 of file search_graph.cc.

10.278.2.3 coco::coco::undelta::undelta (const undelta & *__d*) [inline]

Copy Constructor, which constructs a new undelta from an existing undelta. The clone operation for the undelta_base will be called in that process, effectively overloading the copy constructor for the undelta_base.

Definition at line 199 of file api_delta.h.

10.278.2.4 coco::coco::undelta::~undelta ( ) [inline]

Destructor, which frees the `_d` using `delete`

Definition at line 206 of file api_delta.h.

10.278.2.5 coco::coco::undelta::undelta (size_t *s*) [inline]

Standard Constructor

Definition at line 234 of file search_graph.cc.

10.278.2.6 coco::coco::undelta::undelta (undelta_base * __d, size_t s) [inline]

Internal Constructor, which just copies the pointer to the undelta_base. This pointer must be allocated by `new`, since it will be removed with `delete` in the destructor

Definition at line 238 of file search_graph.cc.

10.278.2.7 coco::coco::undelta::undelta (const undelta & __d)

Copy Constructor, which constructs a new undelta from an existing undelta. The clone operation for the undelta_base will be called in that process, effectively overloading the copy constructor for the undelta_base.

10.278.2.8 coco::coco::undelta::~undelta ( )

Destructor, which frees the `_d` using `delete`

10.278.2.9 coco::coco::undelta::undelta (size_t s) [inline]

Standard Constructor

Definition at line 234 of file search_graph.cc.

10.278.2.10 coco::coco::undelta::undelta (undelta_base * __d, size_t s) [inline]

Internal Constructor, which just copies the pointer to the undelta_base. This pointer must be allocated by `new`, since it will be removed with `delete` in the destructor

Definition at line 238 of file search_graph.cc.

10.278.2.11 coco::coco::undelta::undelta (const undelta & __d)

Copy Constructor, which constructs a new undelta from an existing undelta. The clone operation for the undelta_base will be called in that process, effectively overloading the copy constructor for the undelta_base.

10.278.2.12 coco::coco::undelta::~undelta ( )

Destructor, which frees the `_d` using `delete`

10.278.3 Member Function Documentation

10.278.3.1 size_t coco::coco::undelta::delta_size ( ) const [inline]

Returns the size of the delta stored in the database

Definition at line 230 of file api_delta.h.

10.278.3.2 size_t coco::coco::undelta::delta_size ( ) const

Returns the size of the delta stored in the database

10.278.3.3 size_t coco::coco::undelta::delta_size ( ) const

Returns the size of the delta stored in the database

10.278.3.4 const undelta_base * coco::coco::undelta::get_base ( ) const [inline]

Return the undelta_base stored in this wrapper

Definition at line 221 of file api_delta.h.

10.278.3.5 const undelta_base* coco::coco::undelta::get_base ( ) const

Return the undelta_base stored in this wrapper

10.278.3.6 const undelta_base* coco::coco::undelta::get_base ( ) const

Return the undelta_base stored in this wrapper

10.278.3.7 undelta& coco::coco::undelta::operator= (const undelta & _u)

Assignment operator, which uses the clone operation for the undelta_base, effectively overloading the assignment operator for the undelta_base.

10.278.3.8 undelta & coco::coco::undelta::operator= (const undelta & _u) [inline]

Assignment operator, which uses the clone operation for the undelta_base, effectively overloading the assignment operator for the undelta_base.

Definition at line 212 of file api_delta.h.

10.278.3.9 undelta& coco::coco::undelta::operator= (const undelta & _u)

Assignment operator, which uses the clone operation for the undelta_base, effectively overloading the assignment operator for the undelta_base.

10.278.3.10 bool coco::coco::undelta::unapply (work_node & _x, const delta_id & _i) const

Undo the delta with delta_id _i in work node _x

10.278.3.11 bool coco::coco::undelta::unapply (work_node & _x, const delta_id & _i) const [inline]

Undo the delta with delta_id _i in work node _x

Definition at line 223 of file api_delta.h.

10.278.3.12 bool coco::coco::undelta::unapply (work_node & _x, const delta_id & _i) const

Undo the delta with delta_id _i in work node _x

10.278.3.13 bool coco::coco::undelta::unapply3 (work_node & _x, const work_node & _y, const delta_id & _i) const [inline]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x.

Definition at line 226 of file api_delta.h.

10.278.3.14 bool coco::coco::undelta::unapply3 (work_node & _x, const work_node & _y, const delta_id & _i) const

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x.

10.278.3.15 bool coco::coco::undelta::unapply3 (work_node & _x, const work_node & _y, const delta_id & _i) const

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x.

The documentation for this class was generated from the following files:

- api_deltabase.h
- api_delta.h

10.279 coco::undelta_base Class Reference

Base class for the undeltas.

`#include <api_deltabase.h>`

Public Member Functions

- undelta_base ()
- undelta_base (const undelta_base &__d)
- virtual undelta_base * new_copy () const PURE_VIRTUALvirtual void destroy_copy(undelta_base *__d) const PURE_VIRTUALvirtual~undelta_base()
- undelta make_undelta (size_t s)
- virtual bool unapply (work_node &_x, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

10.279.1 Detailed Description

Base class for the undeltas, which is wrapped by the undelta class since copy constructors cannot directly be overloaded.

10.279.2 Constructor & Destructor Documentation

10.279.2.1 coco::undelta_base::undelta_base ( ) [inline]

Standard Constructor

Definition at line 276 of file api_deltabase.h.

10.279.2.2 coco::undelta_base::undelta_base (const undelta_base & __d) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 278 of file api_deltabase.h.

10.279.3 Member Function Documentation

10.279.3.1 undelta coco::undelta_base::make_undelta (size_t s) [inline]

Construct an undelta from this undelta_base.

Definition at line 289 of file api_deltabase.h.

10.279.3.2 virtual undelta_base* coco::undelta_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Definition at line 281 of file api_deltabase.h.

10.279.3.3 virtual bool coco::undelta_base::unapply (work_node & _x, const delta_id & _d) const [inline, virtual]

Undo the delta with delta_id _i in work node _x

Definition at line 296 of file api_deltabase.h.

10.279.3.4 virtual bool coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

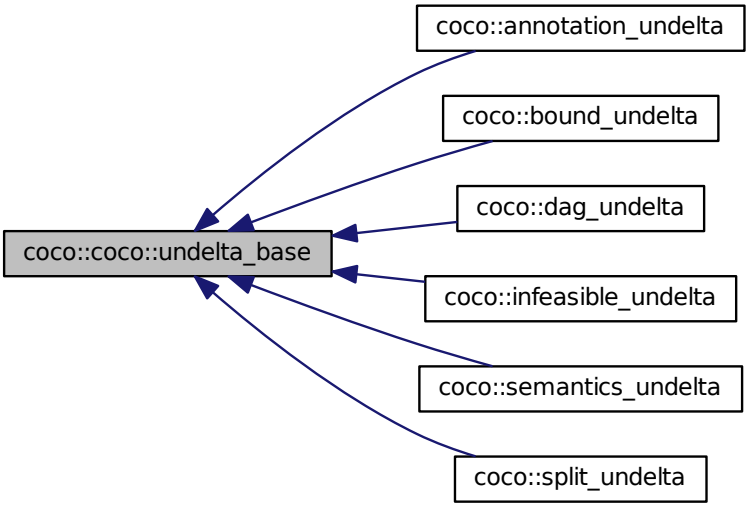
The documentation for this class was generated from the following file:

- api_deltabase.h

10.280 coco::coco::undelta_base Class Reference

Base class for the undeltas.

Inheritance diagram for coco::coco::undelta_base:



Public Member Functions

- undelta_base ()
- undelta_base (const undelta_base &__d)
- virtual undelta_base * new_copy () const PURE_VIRTUALvirtual void destroy_copy(undelta_base *__d) const PURE_VIRTUALvirtual~undelta_base()
- undelta make_undelta (size_t s)
- virtual bool unapply (work_node &_x, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- undelta_base ()
- undelta_base (const undelta_base &__d)
- virtual undelta_base * new_copy () const PURE_VIRTUALvirtual void destroy_copy(undelta_base *__d) const PURE_VIRTUALvirtual~undelta_base()
- undelta make_undelta (size_t s)
- virtual bool unapply (work_node &_x, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const
- undelta_base ()
- undelta_base (const undelta_base &__d)
- virtual undelta_base * new_copy () const PURE_VIRTUALvirtual void destroy_copy(undelta_base *__d) const PURE_VIRTUALvirtual~undelta_base()
- undelta make_undelta (size_t s)
- virtual bool unapply (work_node &_x, const delta_id &_d) const
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

10.280.1 Detailed Description

Base class for the undeltas, which is wrapped by the undelta class since copy constructors cannot directly be overloaded.

Definition at line 272 of file search_graph.cc.

10.280.2 Constructor & Destructor Documentation

10.280.2.1 coco::coco::undelta_base::undelta_base ( ) [inline]

Standard Constructor

Definition at line 276 of file search_graph.cc.

10.280.2.2 coco::coco::undelta_base::undelta_base (const undelta_base & __d) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 278 of file search_graph.cc.

10.280.2.3 coco::coco::undelta_base::undelta_base ( ) [inline]

Standard Constructor

Definition at line 276 of file search_graph.cc.

10.280.2.4 coco::coco::undelta_base::undelta_base (const undelta_base & __d) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 278 of file search_graph.cc.

10.280.2.5 coco::coco::undelta_base::undelta_base ( ) [inline]

Standard Constructor

Definition at line 276 of file search_graph.cc.

10.280.2.6 coco::coco::undelta_base::undelta_base (const undelta_base & __d) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 278 of file search_graph.cc.

10.280.3 Member Function Documentation

10.280.3.1 undelta coco::coco::undelta_base::make_undelta (size_t s) [inline]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.280.3.2 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.280.3.3 undelta coco::coco::undelta_base::make_undelta (size_t *s*) [inline]

Construct an undelta from this undelta_base.

Definition at line 289 of file search_graph.cc.

10.280.3.4 virtual undelta_base* coco::coco::undelta_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::dag_undelta, coco::bound_undelta, coco::annotation_undelta, coco::semantics_-undelta, coco::split_undelta, and coco::infeasible_undelta.

Definition at line 281 of file search_graph.cc.

10.280.3.5 virtual undelta_base* coco::coco::undelta_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::dag_undelta, coco::bound_undelta, coco::annotation_undelta, coco::semantics_-undelta, coco::split_undelta, and coco::infeasible_undelta.

Definition at line 281 of file search_graph.cc.

10.280.3.6 virtual undelta_base* coco::coco::undelta_base::new_copy ( ) const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::dag_undelta, coco::bound_undelta, coco::annotation_undelta, coco::semantics_-undelta, coco::split_undelta, and coco::infeasible_undelta.

Definition at line 281 of file search_graph.cc.

10.280.3.7 virtual bool coco::coco::undelta_base::unapply (work_node & *_x*, const delta_id & *_d*) const [inline, virtual]

Undo the delta with delta_id _i in work node _x

Reimplemented in coco::dag_undelta, coco::bound_undelta, coco::annotation_undelta, coco::semantics_-undelta, coco::infeasible_undelta, and coco::split_undelta.

Definition at line 296 of file search_graph.cc.

10.280.3.8 virtual bool coco::coco::undelta_base::unapply (work_node & *_x*, const delta_id & *_d*) const [inline, virtual]

Undo the delta with delta_id _i in work node _x

Reimplemented in coco::dag_undelta, coco::bound_undelta, coco::annotation_undelta, coco::semantics_-undelta, coco::infeasible_undelta, and coco::split_undelta.

Definition at line 296 of file search_graph.cc.

10.280.3.9 virtual bool coco::coco::undelta_base::unapply (work_node & _x, const delta_id & _d) const [inline, virtual]

Undo the delta with delta_id _i in work node _x

Reimplemented in coco::dag_undelta, coco::bound_undelta, coco::annotation_undelta, coco::semantics_-undelta, coco::infeasible_undelta, and coco::split_undelta.

Definition at line 296 of file search_graph.cc.

10.280.3.10 virtual bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

10.280.3.11 bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [inline, virtual]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

Definition at line 94 of file api_delta.h.

10.280.3.12 virtual bool coco::coco::undelta_base::unapply3 (work_node & _x, const work_node & _y, const delta_id & _d) const [virtual]

Undo the delta with delta_id _d in work node _x, constructing in the process work_node _y, without changing _x. This method should be overloaded in the specializations of this class for maximal efficiency, but it need not since a standard procedure is available, which uses the work_node copy constructur and the unapply method.

The documentation for this class was generated from the following files:

- api_deltabase.h
- api_delta.h

10.281 coco::coco::variable_indicator Class Reference

Bitmap class used to indicate variable occurrence.

Public Member Functions

- variable_indicator ()

- variable_indicator (int num_of_vars)
- variable_indicator (const std::vector< int > &__v, int num_of_vars)
- variable_indicator (const variable_indicator &__x)
- virtual ~variable_indicator ()
- void reserve (int num_of_vars)
- void set_all (const variable_indicator &_vi)
- void set (int __i)
- void set (std::vector< int > __v)
- void set (int start_idx, int end_idx)
- void clear ()
- void unset (int __i)
- void unset (std::vector< int > __v)
- void unset (int start_idx, int end_idx)
- unsigned int sum (int start_idx, int end_idx) const
- bool test (int __i) const
- bool match (const variable_indicator &__v) const
- variable_indicator & operator= (const variable_indicator &__v)
- bool operator== (const variable_indicator &__v)
- std::vector< int > encode ()
- void decode (const std::vector< int > &__e)

10.281.1 Detailed Description

This class is a bitmap class, which is used to indicate which variables share a certain property. E.g., from which variables an expression depends, which variables have been changed since the last evaluation, and the like.

10.281.2 Constructor & Destructor Documentation

10.281.2.1 coco::coco::variable_indicator::variable_indicator ( ) [inline]

Standard Constructor

Definition at line 81 of file search_graph.cc.

10.281.2.2 coco::coco::variable_indicator::variable_indicator (int *num_of_vars*) [inline]

This is a constructor preparing for `num_of_vars` variables.

Definition at line 84 of file search_graph.cc.

10.281.2.3 coco::coco::variable_indicator::variable_indicator (const std::vector< int > & *__v*, int *num_of_vars*) [inline]

This is a constructor preparing for `num_of_vars` variables, which sets all those variables, whose number appears in `__v`.

Definition at line 88 of file search_graph.cc.

10.281.2.4 coco::coco::variable_indicator::variable_indicator (const variable_indicator & __x) [inline]

Standard Copy Constructor

Definition at line 96 of file search_graph.cc.

10.281.2.5 virtual coco::coco::variable_indicator::~variable_indicator ( ) [inline, virtual]

Standard Destructor

Definition at line 99 of file search_graph.cc.

10.281.3 Member Function Documentation

10.281.3.1 void coco::coco::variable_indicator::clear ( ) [inline]

The clear method unsets all variables in the variable_indicator

Definition at line 161 of file search_graph.cc.

10.281.3.2 void coco::coco::variable_indicator::decode (const std::vector< int > & __e) [inline]

This method decodes a variable_indicator from a vector<int>. This operation is the inversion of the encode method.

Definition at line 283 of file search_graph.cc.

10.281.3.3 std::vector<int> coco::coco::variable_indicator::encode ( ) [inline]

This method encodes a variable_indicator as a vector<int> so that it can be, e.g., stored in a basic_alltype.

Definition at line 272 of file search_graph.cc.

10.281.3.4 bool coco::coco::variable_indicator::match (const variable_indicator & __v) const [inline]

The match method compares this variable_indicator with the variable_indicator ___v. It returns if the two bitsets are disjoint.

Definition at line 243 of file search_graph.cc.

10.281.3.5 variable_indicator& coco::coco::variable_indicator::operator= (const variable_indicator & __v) [inline]

Standard Assignment Operator

Definition at line 258 of file search_graph.cc.

10.281.3.6 bool coco::coco::variable_indicator::operator== (const variable_indicator & __v) [inline]

Standard Comparison Operator

Definition at line 265 of file search_graph.cc.

10.281.3.7 void coco::coco::variable_indicator::reserve (int *num_of_vars*) [inline]

This method prepares enough memory that no reallocation is necessary, when num_of_vars have to be stored.

Definition at line 103 of file search_graph.cc.

10.281.3.8 void coco::coco::variable_indicator::set (int *__i*) [inline]

This method sets (sets the bit in the bitmap) the variable with number __i.

Definition at line 127 of file search_graph.cc.

10.281.3.9 void coco::coco::variable_indicator::set (std::vector< int > *__v*) [inline]

With this set method all the variables with numbers in __v can be set.

Definition at line 133 of file search_graph.cc.

10.281.3.10 void coco::coco::variable_indicator::set (int *start_idx*, int *end_idx*) [inline]

This method sets all variables with numbers in {[start_idx,end_idx[}.

Definition at line 141 of file search_graph.cc.

10.281.3.11 void coco::coco::variable_indicator::set_all (const variable_indicator & *_vi*) [inline]

The set_all method sets bits for all those variables (within the valid set of numbers, i.e. variable number < num_of_vars) which are set in _vi.

Definition at line 113 of file search_graph.cc.

10.281.3.12 unsigned int coco::coco::variable_indicator::sum (int *start_idx*, int *end_idx*) const [inline]

The sum method counts the number of variables which are set in the number range {[start_idx,end_idx[}.

Definition at line 205 of file search_graph.cc.

10.281.3.13 bool coco::coco::variable_indicator::test (int *__i*) const [inline]

This method returns whether the variable __i is set.

Definition at line 235 of file search_graph.cc.

10.281.3.14 void coco::coco::variable_indicator::unset (int *__i*) [inline]

This method unsets (clears the bit in the bitmap) the variable with number __i.

Definition at line 169 of file search_graph.cc.

10.281.3.15 void coco::coco::variable_indicator::unset (std::vector< int > *__v*) [inline]

With this unset method all the variables with numbers in __v can be unset.

Definition at line 176 of file search_graph.cc.

10.281.3.16 void coco::coco::variable_indicator::unset (int *start_idx*, int *end_idx*) [inline]

This method unsets all variables with numbers in {[start_idx,end_idx[}.

Definition at line 184 of file search_graph.cc.

The documentation for this class was generated from the following file:

- evaluator.h

10.282 coco::variable_indicator Class Reference

Bitmap class used to indicate variable occurrence.

`#include <evaluator.h>`

Public Member Functions

- variable_indicator ()
- variable_indicator (int num_of_vars)
- variable_indicator (const std::vector< int > &__v, int num_of_vars)
- variable_indicator (const variable_indicator &__x)
- virtual ~variable_indicator ()
- void reserve (int num_of_vars)
- void set_all (const variable_indicator &_vi)
- void set (int __i)
- void set (std::vector< int > __v)
- void set (int start_idx, int end_idx)
- void clear ()
- void unset (int __i)
- void unset (std::vector< int > __v)
- void unset (int start_idx, int end_idx)
- unsigned int sum (int start_idx, int end_idx) const
- bool test (int __i) const
- bool match (const variable_indicator &__v) const
- variable_indicator & operator= (const variable_indicator &__v)
- bool operator== (const variable_indicator &__v)
- std::vector< int > encode ()
- void decode (const std::vector< int > &__e)

10.282.1 Detailed Description

This class is a bitmap class, which is used to indicate which variables share a certain property. E.g., from which variables an expression depends, which variables have been changed since the last evaluation, and the like.

10.282.2 Constructor & Destructor Documentation

10.282.2.1 coco::variable_indicator::variable_indicator ( ) [inline]

Standard Constructor

Definition at line 80 of file evaluator.h.

10.282.2.2 coco::variable_indicator::variable_indicator (int *num_of_vars*) [inline]

This is a constructor preparing for num_of_vars variables.

Definition at line 83 of file evaluator.h.

10.282.2.3 coco::variable_indicator::variable_indicator (const std::vector< int > & *__v*, int *num_of_vars*) [inline]

This is a constructor preparing for num_of_vars variables, which sets all those variables, whose number appears in __v.

Definition at line 87 of file evaluator.h.

10.282.2.4 coco::variable_indicator::variable_indicator (const variable_indicator & *__x*) [inline]

Standard Copy Constructor

Definition at line 95 of file evaluator.h.

10.282.2.5 virtual coco::variable_indicator::~variable_indicator ( ) [inline, virtual]

Standard Destructor

Definition at line 98 of file evaluator.h.

10.282.3 Member Function Documentation

10.282.3.1 void coco::variable_indicator::clear ( ) [inline]

The clear method unsets all variables in the variable_indicator

Definition at line 160 of file evaluator.h.

10.282.3.2 void coco::variable_indicator::decode (const std::vector< int > & *__e*) [inline]

This method decodes a variable_indicator from a vector<int>. This operation is the inversion of the encode method.

Definition at line 282 of file evaluator.h.

10.282.3.3 std::vector<int> coco::variable_indicator::encode ( ) [inline]

This method encodes a variable_indicator as a vector<int> so that it can be, e.g., stored in a basic_alltype.

Definition at line 271 of file evaluator.h.

10.282.3.4 bool coco::variable_indicator::match (const variable_indicator & __v) const [inline]

The match method compares this variable_indicator with the variable_indicator ___v. It returns if the two bitsets are disjoint.

Definition at line 242 of file evaluator.h.

10.282.3.5 variable_indicator& coco::variable_indicator::operator= (const variable_indicator & __v) [inline]

Standard Assignment Operator

Definition at line 257 of file evaluator.h.

10.282.3.6 bool coco::variable_indicator::operator== (const variable_indicator & __v) [inline]

Standard Comparison Operator

Definition at line 264 of file evaluator.h.

10.282.3.7 void coco::variable_indicator::reserve (int *num_of_vars*) [inline]

This method prepares enough memory that no reallocation is necessary, when num_of_vars have to be stored.

Definition at line 102 of file evaluator.h.

10.282.3.8 void coco::variable_indicator::set (int __i) [inline]

This method sets (sets the bit in the bitmap) the variable with number ___i.

Definition at line 126 of file evaluator.h.

10.282.3.9 void coco::variable_indicator::set (std::vector< int > __v) [inline]

With this set method all the variables with numbers in ___v can be set.

Definition at line 132 of file evaluator.h.

10.282.3.10 void coco::variable_indicator::set (int *start_idx*, int *end_idx*) [inline]

This method sets all variables with numbers in {[start_idx,end_idx[}.

Definition at line 140 of file evaluator.h.

10.282.3.11 void coco::variable_indicator::set_all (const variable_indicator & _vi) [inline]

The set_all method sets bits for all those variables (within the valid set of numbers, i.e. variable number < num_of_vars) which are set in _vi.

Definition at line 112 of file evaluator.h.

10.282.3.12 unsigned int coco::variable_indicator::sum (int *start_idx*, int *end_idx*) const [inline]

The sum method counts the number of variables which are set in the number range {[start_idx,end_idx[}.

Definition at line 204 of file evaluator.h.

10.282.3.13 bool coco::variable_indicator::test (int *__i*) const [inline]

This method returns whether the variable ___i is set.

Definition at line 234 of file evaluator.h.

10.282.3.14 void coco::variable_indicator::unset (int *__i*) [inline]

This method unsets (clears the bit in the bitmap) the variable with number ___i.

Definition at line 168 of file evaluator.h.

10.282.3.15 void coco::variable_indicator::unset (std::vector< int > *__v*) [inline]

With this unset method all the variables with numbers in ___v can be unset.

Definition at line 175 of file evaluator.h.

10.282.3.16 void coco::variable_indicator::unset (int *start_idx*, int *end_idx*) [inline]

This method unsets all variables with numbers in {[start_idx,end_idx[}.

Definition at line 183 of file evaluator.h.

The documentation for this class was generated from the following file:

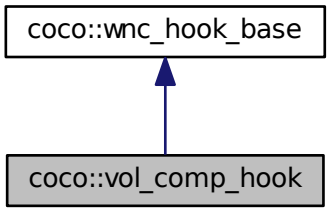
- evaluator.h

10.283 coco::vol_comp_hook Class Reference

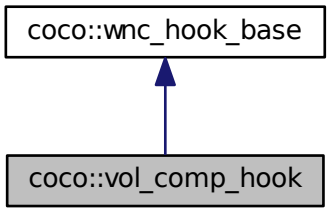
The volume computation hook (work node computation hook)

`#include <vol_hook.h>`

Inheritance diagram for coco::vol_comp_hook:



Collaboration diagram for coco::vol_comp_hook:



Public Member Functions

- vol_comp_hook ()
- virtual ~vol_comp_hook ()
- vol_comp_hook * new_copy () const
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *a=NULL, std::list< certificate > *b=NULL) const
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)
- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.283.1 Detailed Description

This class is a work node computation hook (see work_node_comp_hook), which calculates the volume of the work node (from the bound constraints) and keeps it in column "vol".

10.283.2 Constructor & Destructor Documentation

10.283.2.1 coco::vol_comp_hook::vol_comp_hook ( ) [inline]

Standard Constructor

Definition at line 46 of file vol_hook.h.

10.283.2.2 virtual coco::vol_comp_hook::~vol_comp_hook ( ) [inline, virtual]

Standard Destructor

Definition at line 49 of file vol_hook.h.

10.283.3 Member Function Documentation

10.283.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected, inherited]

Remove all columns, whose identifiers are stored in __managed_cols from table stable.

Definition at line 34 of file comp_hook.cc.

10.283.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in __managed_cols.

Definition at line 238 of file comp_hook.h.

10.283.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected, inherited]

Add a column of type _CI with name colname to the table stable. The column is stored in __managed_cols.

Definition at line 135 of file comp_hook.h.

10.283.3.4 bool coco::vol_comp_hook::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon unregistering this hook, destroy the column "vol" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 90 of file vol_hook.h.

10.283.3.5 bool coco::vol_comp_hook::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the column "vol" in the "search info" table.

Reimplemented from coco::wnc_hook_base.

Definition at line 85 of file vol_hook.h.

10.283.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline, inherited]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.283.3.7 vol_comp_hook* coco::vol_comp_hook::new_copy ( ) const [inline, virtual]

Clone Operation

Implements coco::wnc_hook_base.

Definition at line 52 of file vol_hook.h.

10.283.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_id of work_node `wn`.

Definition at line 43 of file comp_hook.cc.

10.283.3.9 void coco::vol_comp_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *a* = NULL, std::list< certificate > * *b* = NULL) const [inline, virtual]

The evaluation operator of this computation hook. It stores the volume of the work_node `wn` in dbt_row `dbr`.

Implements coco::wnc_hook_base.

Definition at line 56 of file vol_hook.h.

10.283.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected, inherited]

This method is an accessor to the search_node_relation of work_node `wn`.

Definition at line 46 of file comp_hook.cc.

The documentation for this class was generated from the following file:

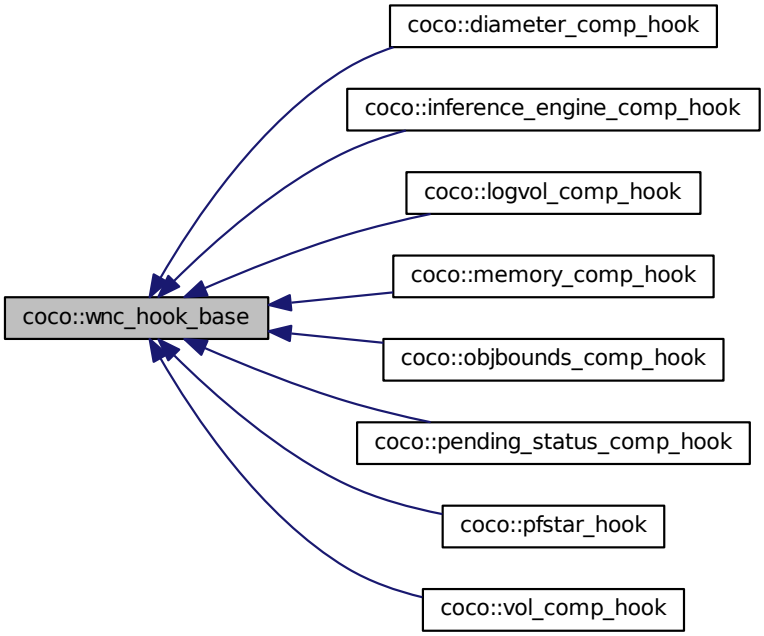
- vol_hook.h

10.284 coco::wnc_hook_base Class Reference

Base class for the work node computation hooks.

`#include <comp_hook.h>`

Inheritance diagram for coco::wnc_hook_base:



Public Member Functions

- wnc_hook_base ()
- wnc_hook_base (const std::string &name)
- wnc_hook_base (const char *name)
- wnc_hook_base (const wnc_hook_base &w)
- virtual ~wnc_hook_base ()
- virtual wnc_hook_base * new_copy () const =0
- virtual void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *add_ds, std::list< certificate > *add_cs) const =0
- virtual bool init_columns (vdbl::standard_table &stable)
- virtual bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

Protected Member Functions

- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const std::string &colname, const _CI &c)
- template<class _CI >
  bool _init_column (vdbl::standard_table &stable, const char *colname, const _CI &c)
- bool _drop_columns (vdbl::standard_table &stable)

- search_node_relation parent_relation (const work_node &wn) const
- search_node_id node_id (const work_node &wn) const

10.284.1 Detailed Description

Base class for the work node computation hooks, which is wrapped by the work_node_comp_hook class since copy constructors cannot directly be overloaded.

10.284.2 Constructor & Destructor Documentation

10.284.2.1 coco::wnc_hook_base::wnc_hook_base ( ) [inline]

Standard Constructor

Definition at line 153 of file comp_hook.h.

10.284.2.2 coco::wnc_hook_base::wnc_hook_base (const std::string & *name*) [inline]

Constructor setting the identifier string of this hook to `name`

Definition at line 155 of file comp_hook.h.

10.284.2.3 coco::wnc_hook_base::wnc_hook_base (const char * *name*) [inline]

Constructor setting the identifier string of this hook to `name`

Definition at line 157 of file comp_hook.h.

10.284.2.4 coco::wnc_hook_base::wnc_hook_base (const wnc_hook_base & *w*) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 159 of file comp_hook.h.

10.284.2.5 virtual coco::wnc_hook_base::~wnc_hook_base ( ) [inline, virtual]

Standard Destructor

Definition at line 163 of file comp_hook.h.

10.284.3 Member Function Documentation

10.284.3.1 bool coco::wnc_hook_base::_drop_columns (vdbl::standard_table & *stable*) [protected]

Remove all columns, whose identifiers are stored in `__managed_cols` from table `stable`.

Definition at line 34 of file comp_hook.cc.

10.284.3.2 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *colname*, const _CI & *c*) [protected]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 238 of file comp_hook.h.

10.284.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const char * *colname*, const _CI & *c*) [inline, protected]

Add a column of type _CI with name colname to the table stable. The column is stored in ___managed_cols.

Definition at line 135 of file comp_hook.h.

10.284.3.4 virtual bool coco::wnc_hook_base::drop_columns (vdbl::standard_table & *stable*) [inline, virtual]

Remove all columns, whose identifiers are stored in ___managed_cols from table stable. If columns are used, a call to _drop_columns is needed in an overloaded version of this method.

Reimplemented in coco::inference_engine_comp_hook, coco::objbounds_comp_hook, coco::pending_status_comp_hook, coco::vol_comp_hook, coco::diameter_comp_hook, coco::pfstar_hook, coco::memory_comp_hook, and coco::logvol_comp_hook.

Definition at line 187 of file comp_hook.h.

10.284.3.5 virtual bool coco::wnc_hook_base::init_columns (vdbl::standard_table & *stable*) [inline, virtual]

Upon registering this hook, initialize the column(s) in the "search info" table in which the result(s) of this hook are stored. It must be overloaded if columns are needed. In that case, the _init_column method should be used.

Reimplemented in coco::inference_engine_comp_hook, coco::objbounds_comp_hook, coco::pending_status_comp_hook, coco::vol_comp_hook, coco::diameter_comp_hook, coco::memory_comp_hook, coco::logvol_comp_hook, and coco::pfstar_hook.

Definition at line 182 of file comp_hook.h.

10.284.3.6 const std::string& coco::wnc_hook_base::name ( ) const [inline]

Return the identifier string of this work node computation hook.

Definition at line 190 of file comp_hook.h.

10.284.3.7 virtual wnc_hook_base* coco::wnc_hook_base::new_copy ( ) const [pure virtual]

Clone Operation, which must be overloaded

Implemented in coco::inference_engine_comp_hook, coco::objbounds_comp_hook, coco::pfstar_hook, coco::pending_status_comp_hook, coco::diameter_comp_hook, coco::logvol_comp_hook, coco::memory_comp_hook, and coco::vol_comp_hook.

10.284.3.8 search_node_id coco::wnc_hook_base::node_id (const work_node & *wn*) const [protected]

This method is an accessor to the search_node_id of work_node wn.

Definition at line 43 of file comp_hook.cc.

10.284.3.9 virtual void coco::wnc_hook_base::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *add_ds*, std::list< certificate > * *add_cs*) const [pure virtual]

The evaluation operator of this computation hook. It stores the result of the hook applied to work_node wn in dbt_row dbr, which in return is stored in the "search info" table of the search database. The hook can compute additional deltas and add them to add_ds, which are applied to the work node during the split. It must be overloaded.

Implemented in coco::inference_engine_comp_hook, coco::objbounds_comp_hook, coco::pending_status_comp_hook, coco::pfstar_hook, coco::diameter_comp_hook, coco::logvol_comp_hook, coco::memory_comp_hook, and coco::vol_comp_hook.

10.284.3.10 search_node_relation coco::wnc_hook_base::parent_relation (const work_node & *wn*) const [protected]

This method is an accessor to the search_node_relation of work_node wn.

Definition at line 46 of file comp_hook.cc.

The documentation for this class was generated from the following files:

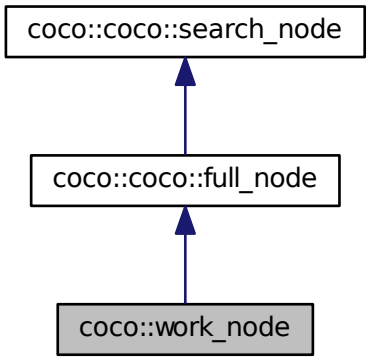
- comp_hook.h
- comp_hook.cc

10.285 coco::work_node Class Reference

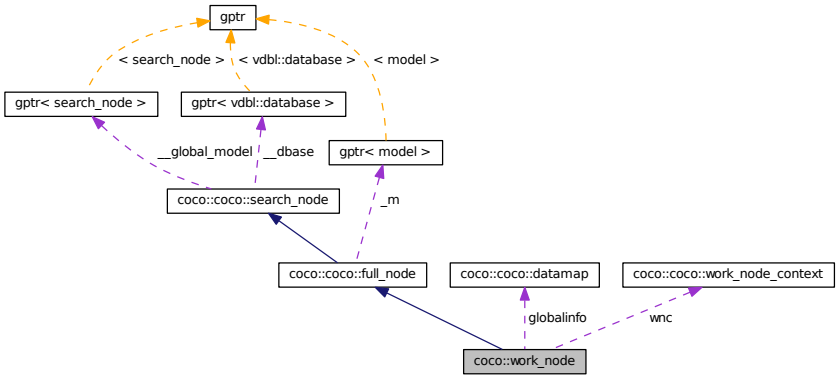
Work node, which is passed to the inference engines.

```
#include <search_node.h>
```

Inheritance diagram for coco::work_node:



Collaboration diagram for coco::work_node:



Classes

- class constraint_iterator_base

  The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

Public Types

- typedef constraint_iterator_base < expression_walker, const std::vector< expression_walker > *, const std::vector < expression_walker > &, const expression_walker *, const expression_walker

&, std::vector < expression_walker > ::const_iterator > constraint_const_iterator
- typedef constraint_iterator_base < expression_walker, std::vector< expression_walker > *, std::vector < expression_walker > &, expression_walker *, expression_walker &, std::vector < expression_walker > ::iterator > constraint_iterator
- typedef uint32_t transaction_number

Public Member Functions

- transaction_number get_transaction_number ()
- void init_cnumbers ()
- void reset_node_ranges ()
- void make_node_ranges (bool keep_old_ranges)
- void make_node_order ()
- double compute_log_volume (const std::vector< interval > &_r) const
- void set_globalinfo ()
- work_node (const work_node &__w)
- work_node (const full_node &__f, const std::list< delta_id > &__de, vdbl::viewdbase *vdbf=NULL)
- work_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__m, gptr< vdbl::database > &__d, const std::vector< annotation > &__an, const std::list< delta_id > &__de, gptr< search_node > *_gm, search_node_relation snr=snr_worknode, vdbl::viewdbase *vdbf=NULL)
- virtual ~work_node ()
- const model * get_model () const
- model * get_model ()
- work_node_context * get_work_node_context ()
- const work_node_context * get_work_node_context () const
- vdbl::viewdbase & get_viewdbase ()
- const vdbl::viewdbase & get_viewdbase () const
- vdbl::viewdbase & get_fullviewdbase ()
- const vdbl::viewdbase & get_fullviewdbase () const
- inference_module_cache & get_inf_mod_cache ()
- delta get_delta (const delta_id &_id)
- const delta & get_delta (const delta_id &_id) const
- certificate get_certificate (const delta_id &_id)
- const certificate & get_certificate (const delta_id &_id) const
- double log_volume () const
- double gain () const
- const datamap & global_info () const
- double reset_gain ()
- work_node & operator= (const work_node &__w)
- work_node & operator= (const full_node &__f)
- bool is_delta () const
- unsigned int n_annotations () const
- const annotation & get_annotation (unsigned int i) const
- const std::vector< annotation > & get_annotations () const
- const vdbl::database * get_database () const
- model * get_model_ptr () const
- vdbl::database * get_database_ptr () const
- vdbl::userid get_dbuserid () const

- gptr< search_node > * global_model () const
- gptr< vdbl::database > * database () const
- search_node_id get_id () const
- void set_id (const search_node_id &i)
- vdbl::rowid get_rowid () const
- void set_rowid (const vdbl::rowid &i)
- void keep (const annotation &_an)
- void keep (const std::vector< annotation > &_anv)
- void unkeep (const annotation &_an)
- void unkeep (const std::vector< annotation > &_anv)

Methods for accessing model parts

The methods of this section are used to extract or access parts of the model with specific properties, e.g., extract the linear part of the model, access only the polynomial constraints,... In all these methods the type of the expressions is specified using type values from e_expression_type and their combination.

See also

e_expression_type

- model get (unsigned int __type)
  Retrieve a subpart of a model.
- constraint_const_iterator get_begin (unsigned int __type) const
  Get an iterator to (read only) access constraints of a specified type.
- constraint_const_iterator get_end (unsigned int __type) const
  Get an iterator pointing past the constraints of a specified type.
- constraint_iterator get_begin (unsigned int __type)
  Get an iterator to access constraints of a specified type.
- constraint_iterator get_end (unsigned int __type)
  Get an iterator pointing past the constraints of a specified type.
- unsigned int n (unsigned int __type) const

- bool global_info (basic_alltype &b, const std::string &i) const
- bool global_info (basic_alltype &b, const char *i) const

- basic_alltype global_info (const std::string &i) const
- basic_alltype global_info (const char *i) const

Public Attributes

- std::list< delta_id > deltas
- std::list< delta_id >::iterator deltanew_it
- std::list< delta_id >::iterator splithook_deltanew_it
- std::set< search_node_id > parents_in_graph
- std::map< delta_id, undelta > undeltas
- vdbl::standard_table * dtable
- vdbl::tableid dtable_id

- vdbl::standard_table * gitable
- vdbl::tableid gitable_id
- work_node_context * wnc
- vdbl::viewdbase * __vdb
- vdbl::viewdbase * __vdbf
- std::map< transaction_number, std::list< std::vector< delta > > > proposed_splits
- std::map< delta_id, transaction_number > split_delta_ids
- std::vector< annotation > _ann

Additional data needed in the solution process

These members keep data which is needed in the solution process to aid the various inference engines.

- std::vector< interval > node_ranges
- std::vector< unsigned int > node_order
- bool infeasible
- double log_vol
- double gain_factor
- datamap globalinfo

Protected Member Functions

- search_node_relation parent_relation () const

Protected Attributes

- gptr< model > * _m
- gptr< search_node > * __global_model
- gptr< vdbl::database > * __dbase
- vdbl::userid _dbuser
- search_node_relation _snr
- search_node_id _id
- std::vector< annotation > _keep
- vdbl::rowid _rid

Friends

- class delta
- class undelta
- class certificate
- class wnc_hook_base
- work_node operator+ (const work_node &_w, const delta_id &_d)

  Add one delta to a work node.
- work_node operator- (const work_node &_w, const delta_id &_d)

  Remove one delta from a work node.
- work_node & operator+= (work_node &_w, const delta_id &_d)

  Add one delta to a work node.
- work_node & operator-= (work_node &_w, const delta_id &_d)

  Remove one delta from a work node.

- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node operator+ (const work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Add a number of deltas to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node operator- (const work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Remove a number of deltas from a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node & operator+= (work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Add a number of deltas to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node & operator-= (work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Remove a number of deltas from a work node.
- class delta_base
- class certificate_base
- class dag_delta
- class dag_undelta
- class search_graph

10.285.1 Detailed Description

This is class of nodes, an enhanced full node, which is constructed, whenever a search focus is set to a specific search node in the search graph. It stores a complete model along with all annotations, a fair amount of additional information (including some caches) and the undelta information for all deltas, which had to be applied to construct that work node from the closest full node ancestor in the search graph.

10.285.2 Member Typedef Documentation

10.285.2.1 typedef constraint_iterator_base<expression_walker, const std::vector<expression_walker>*, const std::vector<expression_walker>&,const expression_walker*, const expression_walker&, std::vector<expression_walker>::const_iterator> coco::work_node::constraint_const_iterator

This type defines a special const iterator, which iterates through all constraints of a specified type.

See also

get_begin

Definition at line 418 of file search_node.h.

10.285.2.2 typedef constraint_iterator_base<expression_walker, std::vector<expression_walker>*, std::vector<expression_walker>&,expression_walker*,expression_walker&, std::vector<expression_walker>::iterator> coco::work_node::constraint_iterator

This type defines a special iterator, which iterates through all constraints of a specified type.

See also

get_begin

Definition at line 432 of file search_node.h.

10.285.2.3 typedef uint32_t coco::work_node::transaction_number

The transaction numbers are used for keeping track of split deltas, whose contents are stored in the proposed_splits member locally in the work node.

Definition at line 508 of file search_node.h.

10.285.3 Constructor & Destructor Documentation

10.285.3.1 coco::work_node::work_node (const work_node & __w)

Standard Copy Constructor

10.285.3.2 coco::work_node::work_node (const full_node & __f, const std::list< delta_id > & __de, vdbl::viewdbase * vdbf = NULL)

Expansion Constructor from full_node

10.285.3.3 coco::work_node::work_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __m, gptr< vdbl::database > & __d, const std::vector< annotation > & __an, const std::list< delta_id > & __de, gptr< search_node > * _gm, search_node_relation snr = snr_worknode, vdbl::viewdbase * vdbf = NULL)

This constructor generates a new work node (a standalone worknode by default) with search_node_id i and search_node_relation __snr. The parameter __m is used to specify the full model, while the parameters _gm, __d, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized by parameter __an, and the list of delta ids is set to __de.

10.285.3.4 virtual coco::work_node::~work_node ( ) [virtual]

Standard Destructor

10.285.4 Member Function Documentation

10.285.4.1 double coco::work_node::compute_log_volume (const std::vector< interval > & _r) const

This method computes a finite approximation to the logarithmic volume of the box determined by the range vector _r. Note that _r is a range vector, i.e. it contains a range for every node in the DAG and is indexed by node numbers and not by variable numbers.

10.285.4.2 gptr<vdbl::database>* coco::coco::search_node::database ( ) const [inline, inherited]

This is the accessor method for the search database.

Definition at line 157 of file search_graph.cc.

10.285.4.3 double coco::work_node::gain ( ) const [inline]

This method returns the current gain_factor of the work_node.

Definition at line 757 of file search_node.h.

10.285.4.4 model coco::work_node::get (unsigned int __type) [inline]

The get function returns a subpart of the model, which only consists of objective function and constraints of the specified type ___type. usage e.g.:

```
get(ex_bounds|ex_linear|ex_quadratic)
```

for a full quadratic model, equivalent:

```
get(ex_atmquad)
```

Bug The get function is not yet properly implemented.

Definition at line 655 of file search_node.h.

10.285.4.5 const annotation& coco::coco::full_node::get_annotation (unsigned int *i*) const [inline, inherited]

A call to this method returns a const reference to the ith annotation.

Definition at line 362 of file search_graph.cc.

10.285.4.6 const std::vector<annotation>& coco::coco::full_node::get_annotations ( ) const [inline, inherited]

A call to this method returns a const reference to the whole vector of annotations.

Definition at line 367 of file search_graph.cc.

10.285.4.7 constraint_const_iterator coco::work_node::get_begin (unsigned int __type) const

The get_begin() function returns a constraint_const_iterator pointing to the first constraint of type ___type. The iterator then can be used to iterate through all such constraints. Usage e.g.:

```
constraint_const_iterator b = get_begin(ex_linear);
while(b != get_end(ex_linear))
{
    do_something_const(*b);
    ++b;
}
```

to iterate through all linear constraints, which are not bound constraints.

10.285.4.8 constraint_iterator coco::work_node::get_begin (unsigned int __type)

The get_begin() function returns a constraint_iterator pointing to the first constraint of type ___type. The iterator then can be used to iterate through all such constraints. Usage e.g.:

```
constraint_iterator b = get_begin(ex_linear);
while(b != get_end(ex_linear))
{
    do_something(*b);
    ++b;
}
```

to iterate through all linear constraints, which are not bound constraints.

10.285.4.9 certificate coco::work_node::get_certificate (const delta_id & *_id*)

This method returns the certificate associated to the delta with delta_id _id.

10.285.4.10 const certificate& coco::work_node::get_certificate (const delta_id & *_id*) const

This method returns a const reference to the certificate associated to the delta with delta_id _id.

10.285.4.11 const vdbl::database* coco::coco::full_node::get_database ( ) const [inline, inherited]

This method returns a const pointer pointing to a locally stored copy of the search database.

Bug This method needs to be replaced, once the COCONUT environment really gets distributed.

Definition at line 378 of file search_graph.cc.

10.285.4.12 vdbl::database* coco::coco::full_node::get_database_ptr ( ) const [inline, inherited]

This method returns a const pointer pointing to a locally stored copy of the search database.

Bug This method needs to be replaced, once the COCONUT environment really gets distributed.

Definition at line 389 of file search_graph.cc.

10.285.4.13 vdbl::userid coco::coco::search_node::get_dbuserid ( ) const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 151 of file search_graph.cc.

10.285.4.14 delta coco::work_node::get_delta (const delta_id & *_id*)

This method returns the delta with delta_id _id.

10.285.4.15 const delta& coco::work_node::get_delta (const delta_id & *_id*) const

This method returns a const reference to the delta with delta_id _id.

10.285.4.16 constraint_const_iterator coco::work_node::get_end (unsigned int *__type*) const

The get_end() function returns a constraint_const_iterator pointing after the last constraint of type __type. The iterator then can be used to stop the iterating process through all such constraints at the end. Usage e.g.:

```
constraint_const_iterator b = get_begin(ex_linear);
while(b != get_end(ex_linear))
{
    do_something_const(*b);
    ++b;
}
```

to iterate through all linear constraints, which are not bound constraints.

10.285.4.17 constraint_iterator coco::work_node::get_end (unsigned int *__type*)

The get_end() function returns a constraint_iterator pointing after the last constraint of type __type. The iterator then can be used to stop the iterating process through all such constraints at the end. Usage e.g.:

```
constraint_iterator b = get_begin(ex_linear);
while(b != get_end(ex_linear))
{
    do_something(*b);
    ++b;
}
```

to iterate through all linear constraints, which are not bound constraints.

10.285.4.18 vdbl::viewdbase& coco::work_node::get_fullviewdbase ( ) [inline]

This method returns the actual full view database.

Definition at line 629 of file search_node.h.

10.285.4.19 const vdbl::viewdbase& coco::work_node::get_fullviewdbase ( ) const [inline]

This method returns the actual full view database.

Definition at line 632 of file search_node.h.

10.285.4.20 search_node_id coco::coco::search_node::get_id ( ) const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 160 of file search_graph.cc.

10.285.4.21 inference_module_cache& coco::work_node::get_inf_mod_cache ( ) [inline]

This method returns the inference module cache for this work node

Definition at line 635 of file search_node.h.

10.285.4.22 const model* coco::work_node::get_model ( ) const [inline]

This method returns a const pointer pointing to a locally stored copy of the model stored in this full node.

Reimplemented from coco::coco::full_node.

Definition at line 609 of file search_node.h.

10.285.4.23 model* coco::work_node::get_model ( ) [inline]

This method returns a const pointer pointing to a locally stored copy of the model stored in this full node.

Definition at line 613 of file search_node.h.

10.285.4.24 model* coco::coco::full_node::get_model_ptr ( ) const [inline, inherited]

This method returns a pointer pointing to a locally stored copy of the model stored in this full node.

Definition at line 383 of file search_graph.cc.

10.285.4.25 vdbl::rowid coco::coco::search_node::get_rowid ( ) const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 166 of file search_graph.cc.

10.285.4.26 transaction_number coco::work_node::get_transaction_number ( ) [inline]

This method returns a unique transaction number for split deltas.

Definition at line 518 of file search_node.h.

10.285.4.27 vdbl::viewdbase& coco::work_node::get_viewdbase ( ) [inline]

This method returns the actual view database.

Definition at line 623 of file search_node.h.

10.285.4.28 const vdbl::viewdbase& coco::work_node::get_viewdbase ( ) const [inline]

This method returns the actual view database.

Definition at line 626 of file search_node.h.

10.285.4.29 work_node_context* coco::work_node::get_work_node_context ( ) [inline]

This method returns the actual work node context

Definition at line 617 of file search_node.h.

10.285.4.30 const work_node_context* coco::work_node::get_work_node_context ( ) const [inline]

This method returns the actual work node context

Definition at line 620 of file search_node.h.

10.285.4.31 const datamap& coco::work_node::global_info ( ) const [inline]

This method returns the global data used by the algorithm. See `globalinfo`.

Definition at line 760 of file search_node.h.

10.285.4.32 bool coco::work_node::global_info (basic_alltype & *b*, const std::string & *i*) const

This method returns the entry with name `i` in the global data used by the algorithm, setting `b`. It returns whether extraction was successful. See `globalinfo`.

10.285.4.33 bool coco::work_node::global_info (basic_alltype & *b*, const char * *i*) const [inline]

This method returns the entry with name `i` in the global data used by the algorithm, setting `b`. It returns whether extraction was successful. See `globalinfo`.

Definition at line 767 of file search_node.h.

10.285.4.34 basic_alltype coco::work_node::global_info (const std::string & *i*) const

This method returns the entry with name `i` in the global data used by the algorithm. See `globalinfo`.

10.285.4.35 basic_alltype coco::work_node::global_info (const char * *i*) const [inline]

This method returns the entry with name `i` in the global data used by the algorithm. See `globalinfo`.

Definition at line 774 of file search_node.h.

10.285.4.36 gptr<search_node>* coco::coco::search_node::global_model ( ) const [inline, inherited]

This is the accessor method for the global model.

Definition at line 154 of file search_graph.cc.

10.285.4.37 void coco::work_node::init_cnumbers ( )

This method computes the cache values n_bds, n_lin, n_quad, n_poly, and n_other and the log volume.

10.285.4.38 bool coco::coco::full_node::is_delta ( ) const [inline, virtual, inherited]

This method is called to determine whether a search node stores only deltas (delta_node) or a full model (full_node). In case of a full_node it returns `false`.

Reimplemented from coco::coco::search_node.

Definition at line 354 of file search_graph.cc.

10.285.4.39 void coco::coco::search_node::keep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to be the keeper of the annotation `_an`.

Definition at line 173 of file search_graph.cc.

10.285.4.40 void coco::coco::search_node::keep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to be the keeper of all the annotations in `_anv`.

Definition at line 177 of file search_graph.cc.

10.285.4.41 double coco::work_node::log_volume ( ) const [inline]

This method returns the finite approximation to the logarithmic volume of the work_node.

See also

log_vol

Definition at line 754 of file search_node.h.

10.285.4.42 void coco::work_node::make_node_order ( )

A call to this method recalculates the node_order for the model DAG stored in the work node.

10.285.4.43 void coco::work_node::make_node_ranges (bool *keep_old_ranges*)

A call to this method adapts the node_ranges vector to the .

10.285.4.44 unsigned int coco::work_node::n (unsigned int *__type*) const

This method returns the number of constraints of type `__type`. Usage, e.g.:

```
n(ex_linear|ex_equality)
```

returns the number of linear equality constraints.

10.285.4.45 unsigned int coco::coco::full_node::n_annotations ( ) const [inline, inherited]

This method returns the number of annotations stored in this full node.

Definition at line 358 of file search_graph.cc.

10.285.4.46 work_node& coco::work_node::operator= (const work_node & *__w*)

Standard Assignment Operator

10.285.4.47 work_node& coco::work_node::operator= (const full_node & *__f*)

Special Assignment Operator --- be careful when using it!

Reimplemented from coco::coco::full_node.

10.285.4.48 search_node_relation coco::coco::search_node::parent_relation ( ) const [inline, protected, inherited]

A call to this method returns the relation of this node to its parent in the search graph.

Definition at line 110 of file search_graph.cc.

10.285.4.49 double coco::work_node::reset_gain ( ) [inline]

This method returns the current gain_factor of the work_node and resets it to 1.

Definition at line 779 of file search_node.h.

10.285.4.50 void coco::work_node::reset_node_ranges ( )

This method resets all unused node ranges to $[-\infty, \infty]$.

10.285.4.51 void coco::work_node::set_globalinfo ( )

This method extracts the information from the "global info" table and stores it in the `globalinfo` cache of the work_node.

10.285.4.52 void coco::coco::search_node::set_id (const search_node_id & *i*) [inline, inherited]

This is the set method for the search node id of this node.

Definition at line 163 of file search_graph.cc.

10.285.4.53 void coco::coco::search_node::set_rowid (const vdbl::rowid & *i*) [inline, inherited]

This is the set method for the row id of this node.

Definition at line 169 of file search_graph.cc.

10.285.4.54 void coco::search_node::unkeep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of the annotation _an.

Definition at line 82 of file search_node.hpp.

10.285.4.55 void coco::search_node::unkeep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of all the annotations in _anv.

Definition at line 93 of file search_node.hpp.

10.285.5 Friends And Related Function Documentation

10.285.5.1 friend class certificate [friend]

Definition at line 808 of file search_node.h.

10.285.5.2 friend class certificate_base [friend, inherited]

Definition at line 393 of file search_graph.cc.

10.285.5.3 friend class dag_delta [friend, inherited]

Definition at line 394 of file search_graph.cc.

10.285.5.4 friend class dag_undelta [friend, inherited]

Definition at line 395 of file search_graph.cc.

10.285.5.5 friend class delta [friend]

Definition at line 806 of file search_node.h.

10.285.5.6 friend class delta_base [friend, inherited]

Definition at line 392 of file search_graph.cc.

10.285.5.7 work_node operator+ (const work_node & _w, const delta_id & _d) [friend]

This function adds the delta with delta_id _i to work_node _w and returns a new work_node having the delta applied.

Definition at line 298 of file search_node.hpp.

10.285.5.8 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node operator+ (const work_node & _w, const _Ctr< delta_id, _Al > & _d) [friend]

This function adds all the deltas whose delta_ids are in the linear container _d to work_node _w and returns a new work_node having all these deltas applied.

Definition at line 325 of file search_node.hpp.

10.285.5.9 work_node& operator+= (work_node & _w, const delta_id & _d) [friend]

This function adds the delta with delta_id _i to work_node _w changing it in that process. The function returns _w.

Definition at line 311 of file search_node.hpp.

10.285.5.10 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node& operator+= (work_node & _w, const _Ctr< delta_id, _Al > & _d) [friend]

This function adds the deltas whose delta_ids are in the linear container _d to work_node _w changing it in that process. The function returns _w.

Definition at line 341 of file search_node.hpp.

10.285.5.11 work_node operator- (const work_node & _w, const delta_id & _d) [friend]

This function removes the delta with delta_id _i from work_node _w and returns a new work_node having the delta unapplied.

Definition at line 39 of file search_node.cc.

10.285.5.12 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node operator- (const work_node & _w, const _Ctr< delta_id, _Al > & _d) [friend]

This function removes all the deltas whose delta_ids are in the linear container _d from work_node _w and returns a new work_node having all these deltas unapplied.

Definition at line 365 of file search_node.hpp.

10.285.5.13 work_node& operator-= (work_node & _w, const delta_id & _d) [friend]

This function removes (unapplies) the delta with delta_id _i from work_node _w changing it in that process. The function returns _w.

Definition at line 80 of file search_node.cc.

10.285.5.14 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node& operator-= (work_node & _w, const _Ctr< delta_id, _Al > & _d) [friend]

This function removes (unapplies) the deltas whose delta_ids are in the linear container _d from work_node _w changing it in that process. The function returns _w.

Definition at line 383 of file search_node.hpp.

10.285.5.15 friend class search_graph [friend, inherited]

Definition at line 188 of file search_graph.cc.

10.285.5.16 friend class undelta [friend]

Definition at line 807 of file search_node.h.

10.285.5.17 friend class wnc_hook_base [friend]

Definition at line 809 of file search_node.h.

10.285.6 Member Data Documentation

10.285.6.1 gptr<vdbl::database>* coco::coco::search_node::__dbase [protected, inherited]

This is a pointer to a global pointer to the search database associated to this search graph.

Definition at line 89 of file search_graph.cc.

10.285.6.2 gptr<search_node>* coco::coco::search_node::__global_model [protected, inherited]

This is a pointer to a global pointer to the top of the search graph. It is NULL for a standalone node or the top of the graph.

Definition at line 86 of file search_graph.cc.

10.285.6.3 vdbl::viewdbase* coco::work_node::__vdb

This is a restricted view to the search database to be used by the inference engines and report modules.

Definition at line 471 of file search_node.h.

10.285.6.4 vdbl::viewdbase* coco::work_node::__vdbf

This is a full view to the search database to be used by the inference engines and report modules.

Definition at line 474 of file search_node.h.

10.285.6.5 std::vector<annotation> coco::coco::full_node::_ann [inherited]

This is the vector of annotations active at this full node.

Definition at line 283 of file search_graph.cc.

10.285.6.6 vdbl::userid coco::coco::search_node::_dbuser [protected, inherited]

This member stores the database user id under which this search is performed. For the in-memory database this is irrelevant.

Definition at line 92 of file search_graph.cc.

10.285.6.7 search_node_id coco::coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 97 of file search_graph.cc.

10.285.6.8 std::vector<annotation> coco::coco::search_node::_keep [protected, inherited]

The _keep member is a list of all those annotations which are kept by this search node. If the node is destroyed all kept annotations are removed from the search database.

Definition at line 101 of file search_graph.cc.

10.285.6.9 gptr<model>* coco::coco::full_node::_m [protected, inherited]

This is a pointer to a global pointer to the model kept in this full node.

Definition at line 279 of file search_graph.cc.

10.285.6.10 vdbl::rowid coco::coco::search_node::_rid [protected, inherited]

This row id specifies the row in the search info table where the hook information for this search_node is stored

Definition at line 105 of file search_graph.cc.

10.285.6.11 search_node_relation coco::coco::search_node::_snr [protected, inherited]

This member specifies the relation of the search node to its parent(s) in the search graph.

Definition at line 95 of file search_graph.cc.

10.285.6.12 std::list<delta_id>::iterator coco::work_node::deltanew_it

This iterator points to the first 'new' delta of the work_node.

Definition at line 443 of file search_node.h.

10.285.6.13 std::list<delta_id> coco::work_node::deltas

This is the list of all deltas, which had to be applied to construct the given work_node. This includes the deltas necessary to extract the search_node to the work_node from the graph, as well as the "new" deltas, which have been applied by the solution strategy while the work node was analyzed by various inference engines.

Definition at line 440 of file search_node.h.

10.285.6.14 vdbl::standard_table* coco::work_node::dtable

This is a pointer to the "deltas" table in the search database.

Definition at line 456 of file search_node.h.

10.285.6.15 vdbl::tableid coco::work_node::dtable_id

This is the table identifier of the "deltas" table in the search database.

Definition at line 459 of file search_node.h.

10.285.6.16 double coco::work_node::gain_factor

This member keeps an improvement factor (in [0,1]) in bounds sizes since it was last reset.

Definition at line 499 of file search_node.h.

10.285.6.17 vdbl::standard_table* coco::work_node::gitable

This is a pointer to the "global info" table in the search database.

Definition at line 461 of file search_node.h.

10.285.6.18 vdbl::tableid coco::work_node::gitable_id

This is the table identifier of the "global info" table in the search database.

Definition at line 464 of file search_node.h.

10.285.6.19 datamap coco::work_node::globalinfo

This member keeps a cache of the global data used in the algorithm

Definition at line 501 of file search_node.h.

10.285.6.20 bool coco::work_node::infeasible

This boolean specifies, whether the problem represented by this work node is known to be infeasible.

Definition at line 492 of file search_node.h.

10.285.6.21 double coco::work_node::log_vol

This member keeps a finite approximation to the logarithmic volume of the box formed by the variable bounds of the original variables.

Definition at line 496 of file search_node.h.

10.285.6.22 std::vector<unsigned int> coco::work_node::node_order

The node_order vector provides a linear order on the DAG nodes, which is compatible with the DAG structure, i.e., all ancestors of a node are ordered after the node. The vector contains the order as a sequence node numbers of the DAG nodes.

Definition at line 489 of file search_node.h.

10.285.6.23 std::vector<interval> coco::work_node::node_ranges

The node_ranges vector keeps the current best known ranges of all nodes in the model DAG. The vector is indexed by the node numbers of the DAG nodes.

Definition at line 484 of file search_node.h.

10.285.6.24 std::set<search_node_id> coco::work_node::parents_in_graph

This parameter stores the parents of the work node in the graph

Definition at line 450 of file search_node.h.

10.285.6.25 std::map<transaction_number,std::list<std::vector<delta> > > coco::work_node::proposed_splits

This member stores all splits which have been proposed by the various inference engines that can be used to calculate possible splits. Every proposed split (being associated with a split_delta) gets assigned a unique transaction number. This is later used for unapplying the split_delta if necessary. Every split is itself a list of vectors of deltas. Every vector of deltas in the list is associated with one child node generated by the split, and the vector of deltas contains all deltas which have to be applied to the work node for constructing the respective child node from the work node.

Definition at line 531 of file search_node.h.

10.285.6.26 std::map<delta_id, transaction_number> coco::work_node::split_delta_ids

This map connects split_delta delta identifiers with transaction numbers in the proposed_splits map.

Definition at line 534 of file search_node.h.

10.285.6.27 std::list<delta_id>::iterator coco::work_node::splithook_deltanew_it

This iterator points to the first 'new' delta of the work_node generated due to splits and hooks.

Definition at line 447 of file search_node.h.

10.285.6.28 std::map<delta_id,undelta> coco::work_node::undeltas

This map connects delta identifiers of previously applied deltas with their corresponding undelta information.

See also

undelta

Definition at line 454 of file search_node.h.

10.285.6.29 work_node_context* coco::work_node::wnc

This is the work node context corresponding to the current work node.

Definition at line 468 of file search_node.h.

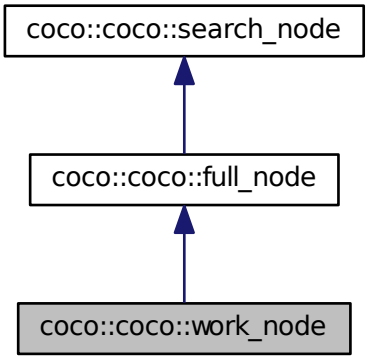
The documentation for this class was generated from the following file:

- search_node.h

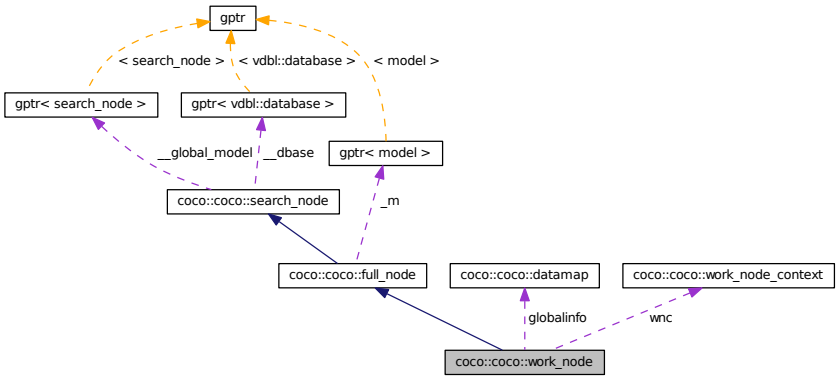
10.286 coco::coco::work_node Class Reference

Work node, which is passed to the inference engines.

Inheritance diagram for coco::coco::work_node:



Collaboration diagram for coco::coco::work_node:



Classes

- class constraint_iterator_base

  The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

Public Types

- typedef constraint_iterator_base < expression_walker, const std::vector< expression_walker > *, const std::vector < expression_walker > &, const expression_walker *, const expression_walker &, std::vector < expression_walker > ::const_iterator > constraint_const_iterator
- typedef constraint_iterator_base < expression_walker, std::vector< expression_walker > *, std::vector < expression_walker > &, expression_walker *, expression_walker &, std::vector < expression_walker > ::iterator > constraint_iterator
- typedef uint32_t transaction_number

Public Member Functions

- transaction_number get_transaction_number ()
- void init_cnumbers ()
- void reset_node_ranges ()
- void make_node_ranges (bool keep_old_ranges)
- void make_node_order ()
- double compute_log_volume (const std::vector< interval > &_r) const
- void set_globalinfo ()
- work_node (const work_node &__w)
- work_node (const full_node &__f, const std::list< delta_id > &__de, vdbl::viewdbase *vdbf=NULL)
- work_node (const search_node_id &_i, const vdbl::userid &_dui, gptr< model > &__m, gptr< vdbl::database > &__d, const std::vector< annotation > &__an, const std::list< delta_id > &__de, gptr< search_node > *_gm, search_node_relation snr=snr_worknode, vdbl::viewdbase *vdbf=NULL)
- virtual ~work_node ()
- const model * get_model () const
- model * get_model ()
- work_node_context * get_work_node_context ()
- const work_node_context * get_work_node_context () const
- vdbl::viewdbase & get_viewdbase ()
- const vdbl::viewdbase & get_viewdbase () const
- vdbl::viewdbase & get_fullviewdbase ()
- const vdbl::viewdbase & get_fullviewdbase () const
- inference_module_cache & get_inf_mod_cache ()
- delta get_delta (const delta_id &_id)
- const delta & get_delta (const delta_id &_id) const
- certificate get_certificate (const delta_id &_id)
- const certificate & get_certificate (const delta_id &_id) const
- double log_volume () const
- double gain () const
- const datamap & global_info () const
- double reset_gain ()
- work_node & operator= (const work_node &__w)
- work_node & operator= (const full_node &__f)
- bool is_delta () const
- unsigned int n_annotations () const
- const annotation & get_annotation (unsigned int i) const
- const std::vector< annotation > & get_annotations () const

- const vdbl::database * get_database () const
- model * get_model_ptr () const
- vdbl::database * get_database_ptr () const
- vdbl::userid get_dbuserid () const
- gptr< search_node > * global_model () const
- gptr< vdbl::database > * database () const
- search_node_id get_id () const
- void set_id (const search_node_id &i)
- vdbl::rowid get_rowid () const
- void set_rowid (const vdbl::rowid &i)
- void keep (const annotation &_an)
- void keep (const std::vector< annotation > &_anv)
- void unkeep (const annotation &_an)
- void unkeep (const std::vector< annotation > &_anv)

Methods for accessing model parts

The methods of this section are used to extract or access parts of the model with specific properties, e.g., extract the linear part of the model, access only the polynomial constraints,... In all these methods the type of the expressions is specified using type values from e_expression_type and their combination.

See also

e_expression_type

- model get (unsigned int __type)
  Retrieve a subpart of a model.
- constraint_const_iterator get_begin (unsigned int __type) const
  Get an iterator to (read only) access constraints of a specified type.
- constraint_const_iterator get_end (unsigned int __type) const
  Get an iterator pointing past the constraints of a specified type.
- constraint_iterator get_begin (unsigned int __type)
  Get an iterator to access constraints of a specified type.
- constraint_iterator get_end (unsigned int __type)
  Get an iterator pointing past the constraints of a specified type.
- unsigned int n (unsigned int __type) const

- bool global_info (basic_alltype &b, const std::string &i) const
- bool global_info (basic_alltype &b, const char *i) const

- basic_alltype global_info (const std::string &i) const
- basic_alltype global_info (const char *i) const

Public Attributes

- std::list< delta_id > deltas
- std::list< delta_id >::iterator deltanew_it
- std::list< delta_id >::iterator splithook_deltanew_it
- std::set< search_node_id > parents_in_graph
- std::map< delta_id, undelta > undeltas
- vdbl::standard_table * dtable
- vdbl::tableid dtable_id
- vdbl::standard_table * gitable
- vdbl::tableid gitable_id
- work_node_context * wnc
- vdbl::viewdbase * __vdb
- vdbl::viewdbase * __vdbf
- std::map< transaction_number, std::list< std::vector< delta > > > proposed_splits
- std::map< delta_id, transaction_number > split_delta_ids
- std::vector< annotation > _ann

Additional data needed in the solution process

These members keep data which is needed in the solution process to aid the various inference engines.

- std::vector< interval > node_ranges
- std::vector< unsigned int > node_order
- bool infeasible
- double log_vol
- double gain_factor
- datamap globalinfo

Protected Member Functions

- search_node_relation parent_relation () const

Protected Attributes

- gptr< model > * _m
- gptr< search_node > * __global_model
- gptr< vdbl::database > * __dbase
- vdbl::userid _dbuser
- search_node_relation _snr
- search_node_id _id
- std::vector< annotation > _keep
- vdbl::rowid _rid

Friends

- class delta
- class undelta
- class certificate
- class wnc_hook_base
- work_node operator+ (const work_node &_w, const delta_id &_d)

  Add one delta to a work node.
- work_node operator- (const work_node &_w, const delta_id &_d)

  Remove one delta from a work node.
- work_node & operator+= (work_node &_w, const delta_id &_d)

  Add one delta to a work node.
- work_node & operator-= (work_node &_w, const delta_id &_d)

  Remove one delta from a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node operator+ (const work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Add a number of deltas to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node operator- (const work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Remove a number of deltas from a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node & operator+= (work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Add a number of deltas to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node & operator-= (work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Remove a number of deltas from a work node.
- class delta_base
- class certificate_base
- class dag_delta
- class dag_undelta
- class search_graph

10.286.1 Detailed Description

This is class of nodes, an enhanced full node, which is constructed, whenever a search focus is set to a specific search node in the search graph. It stores a complete model along with all annotations, a fair amount of additional information (including some caches) and the undelta information for all deltas, which had to be applied to construct that work node from the closest full node ancestor in the search graph.

10.286.2 Member Typedef Documentation

10.286.2.1 typedef constraint_iterator_base<expression_walker, const std::vector<expression_walker>*, const std::vector<expression_walker>&,const expression_walker*, const expression_walker&, std::vector<expression_walker>::const_iterator> coco::coco::work_node::constraint_const_iterator

This type defines a special const iterator, which iterates through all constraints of a specified type.

See also

get_begin

Definition at line 418 of file search_graph.cc.

10.286.2.2 typedef constraint_iterator_base<expression_walker, std::vector<expression_walker>*, std::vector<expression_walker>&,expression_walker*,expression_walker&, std::vector<expression_walker>::iterator> coco::coco::work_node::constraint_iterator

This type defines a special iterator, which iterates through all constraints of a specified type.

See also

get_begin

Definition at line 432 of file search_graph.cc.

10.286.2.3 typedef uint32_t coco::coco::work_node::transaction_number

The transaction numbers are used for keeping track of split deltas, whose contents are stored in the proposed_splits member locally in the work node.

Definition at line 508 of file search_graph.cc.

10.286.3 Constructor & Destructor Documentation

10.286.3.1 coco::work_node::work_node (const work_node & __w) [inline]

Standard Copy Constructor

Definition at line 100 of file search_node.hpp.

10.286.3.2 coco::work_node::work_node (const full_node & __f, const std::list< delta_id > & __de, vdbl::viewdbase * vdbf = NULL)

Expansion Constructor from full_node

Definition at line 202 of file search_node.cc.

10.286.3.3 coco::work_node::work_node (const search_node_id & _i, const vdbl::userid & _dui, gptr< model > & __m, gptr< vdbl::database > & __d, const std::vector< annotation > & __an, const std::list< delta_id > & __de, gptr< search_node > * _gm, search_node_relation snr = snr_worknode, vdbl::viewdbase * vdbf = NULL)

This constructor generates a new work node (a standalone worknode by default) with search_node_id i and search_node_relation __snr. The parameter __m is used to specify the full model, while the parameters _gm, __d, and _dui initialize the global model, search database, and the database user id, respectively. The vector of annotations is initialized by parameter __an, and the list of delta ids is set to __de.

Definition at line 168 of file search_node.cc.

10.286.3.4 coco::work_node::~work_node ( ) [inline, virtual]

Standard Destructor

Definition at line 136 of file search_node.hpp.

10.286.4 Member Function Documentation

10.286.4.1 double coco::work_node::compute_log_volume (const std::vector< interval > & _r) const

This method computes a finite approximation to the logarithmic volume of the box determined by the range vector _r. Note that _r is a range vector, i.e. it contains a range for every node in the DAG and is indexed by node numbers and not by variable numbers.

Definition at line 285 of file search_node.cc.

10.286.4.2 gptr<vdbl::database>* coco::coco::search_node::database ( ) const [inline, inherited]

This is the accessor method for the search database.

Definition at line 157 of file search_graph.cc.

10.286.4.3 double coco::coco::work_node::gain ( ) const [inline]

This method returns the current gain_factor of the work_node.

Definition at line 757 of file search_graph.cc.

10.286.4.4 model coco::coco::work_node::get (unsigned int *__type*) [inline]

The get function returns a subpart of the model, which only consists of objective function and constraints of the specified type ___type. usage e.g.:

```
get(ex_bounds|ex_linear|ex_quadratic)
```

for a full quadratic model, equivalent:

```
get(ex_atmquad)
```

Bug The get function is not yet properly implemented.

Definition at line 655 of file search_graph.cc.

10.286.4.5 const annotation& coco::coco::full_node::get_annotation (unsigned int *i*) const [inline, inherited]

A call to this method returns a const reference to the ith annotation.

Definition at line 362 of file search_graph.cc.

10.286.4.6 const std::vector<annotation>& coco::coco::full_node::get_annotations ( ) const [inline, inherited]

A call to this method returns a const reference to the whole vector of annotations.

Definition at line 367 of file search_graph.cc.

10.286.4.7 work_node::constraint_const_iterator coco::work_node::get_begin (unsigned int *__type*) const [inline]

The get_begin() function returns a constraint_const_iterator pointing to the first constraint of type ___type. The iterator then can be used to iterate through all such constraints. Usage e.g.:

```
constraint_const_iterator b = get_begin(ex_linear);
while(b != get_end(ex_linear))
{
    do_something_const(*b);
    ++b;
}
```

to iterate through all linear constraints, which are not bound constraints.

Definition at line 139 of file search_node.hpp.

10.286.4.8 work_node::constraint_iterator coco::work_node::get_begin (unsigned int *__type*) [inline]

The get_begin() function returns a constraint_iterator pointing to the first constraint of type ___type. The iterator then can be used to iterate through all such constraints. Usage e.g.:

```
constraint_iterator b = get_begin(ex_linear);
while(b != get_end(ex_linear))
{
    do_something(*b);
    ++b;
}
```

to iterate through all linear constraints, which are not bound constraints.

Definition at line 154 of file search_node.hpp.

10.286.4.9 certificate coco::work_node::get_certificate (const delta_id & *_id*) [inline]

This method returns the certificate associated to the delta with delta_id _id.

Definition at line 460 of file search_node.hpp.

10.286.4.10 const certificate & coco::work_node::get_certificate (const delta_id & *_id*) const [inline]

This method returns a const reference to the certificate associated to the delta with delta_id _id.

Definition at line 484 of file search_node.hpp.

10.286.4.11 const vdbl::database* coco::coco::full_node::get_database ( ) const [inline, inherited]

This method returns a const pointer pointing to a locally stored copy of the search database.

Bug This method needs to be replaced, once the COCONUT environment really gets distributed.

Definition at line 378 of file search_graph.cc.

10.286.4.12 vdbl::database* coco::coco::full_node::get_database_ptr ( ) const [inline, inherited]

This method returns a const pointer pointing to a locally stored copy of the search database.

Bug This method needs to be replaced, once the COCONUT environment really gets distributed.

Definition at line 389 of file search_graph.cc.

10.286.4.13 vdbl::userid coco::coco::search_node::get_dbuserid ( ) const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 151 of file search_graph.cc.

10.286.4.14 delta coco::work_node::get_delta (const delta_id & *_id*) [inline]

This method returns the delta with delta_id _id.

Definition at line 428 of file search_node.hpp.

10.286.4.15 const delta & coco::work_node::get_delta (const delta_id & *_id*) const [inline]

This method returns a const reference to the delta with delta_id _id.

Definition at line 444 of file search_node.hpp.

10.286.4.16 work_node::constraint_const_iterator coco::work_node::get_end (unsigned int *__type*) const [inline]

The get_end() function returns a constraint_const_iterator pointing after the last constraint of type __type. The iterator then can be used to stop the iterating process through all such constraints at the end. Usage e.g.:

```
constraint_const_iterator b = get_begin(ex_linear);
while(b != get_end(ex_linear))
{
    do_something_const(*b);
    ++b;
}
```

to iterate through all linear constraints, which are not bound constraints.

Definition at line 147 of file search_node.hpp.

10.286.4.17 work_node::constraint_iterator coco::work_node::get_end (unsigned int *__type*) [inline]

The get_end() function returns a constraint_iterator pointing after the last constraint of type __type. The iterator then can be used to stop the iterating process through all such constraints at the end. Usage e.g.:

```
constraint_iterator b = get_begin(ex_linear);
while(b != get_end(ex_linear))
{
    do_something(*b);
    ++b;
}
```

to iterate through all linear constraints, which are not bound constraints.

Definition at line 161 of file search_node.hpp.

10.286.4.18 vdbl::viewdbase& coco::coco::work_node::get_fullviewdbase ( ) [inline]

This method returns the actual full view database.

Definition at line 629 of file search_graph.cc.

10.286.4.19 const vdbl::viewdbase& coco::coco::work_node::get_fullviewdbase ( ) const [inline]

This method returns the actual full view database.

Definition at line 632 of file search_graph.cc.

10.286.4.20 search_node_id coco::coco::search_node::get_id ( ) const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 160 of file search_graph.cc.

10.286.4.21 inference_module_cache& coco::coco::work_node::get_inf_mod_cache ( ) [inline]

This method returns the inference module cache for this work node

Definition at line 635 of file search_graph.cc.

10.286.4.22 const model* coco::coco::work_node::get_model ( ) const [inline]

This method returns a const pointer pointing to a locally stored copy of the model stored in this full node.

Reimplemented from coco::coco::full_node.

Definition at line 609 of file search_graph.cc.

10.286.4.23 model* coco::coco::work_node::get_model ( ) [inline]

This method returns a const pointer pointing to a locally stored copy of the model stored in this full node.

Definition at line 613 of file search_graph.cc.

10.286.4.24 model* coco::coco::full_node::get_model_ptr ( ) const [inline, inherited]

This method returns a pointer pointing to a locally stored copy of the model stored in this full node.

Definition at line 383 of file search_graph.cc.

10.286.4.25 vdbl::rowid coco::coco::search_node::get_rowid ( ) const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 166 of file search_graph.cc.

10.286.4.26 transaction_number coco::coco::work_node::get_transaction_number ( ) [inline]

This method returns a unique transaction number for split deltas.

Definition at line 518 of file search_graph.cc.

10.286.4.27 vdbl::viewdbase& coco::coco::work_node::get_viewdbase ( ) [inline]

This method returns the actual view database.

Definition at line 623 of file search_graph.cc.

10.286.4.28 const vdbl::viewdbase& coco::coco::work_node::get_viewdbase ( ) const [inline]

This method returns the actual view database.

Definition at line 626 of file search_graph.cc.

10.286.4.29 work_node_context* coco::coco::work_node::get_work_node_context ( ) [inline]

This method returns the actual work node context

Definition at line 617 of file search_graph.cc.

10.286.4.30 const work_node_context* coco::coco::work_node::get_work_node_context ( ) const [inline]

This method returns the actual work node context

Definition at line 620 of file search_graph.cc.

10.286.4.31 const datamap& coco::coco::work_node::global_info ( ) const [inline]

This method returns the global data used by the algorithm. See `globalinfo`.

Definition at line 760 of file search_graph.cc.

10.286.4.32 bool coco::work_node::global_info (basic_alltype & *b*, const std::string & *i*) const

This method returns the entry with name `i` in the global data used by the algorithm, setting `b`. It returns whether extraction was successful. See `globalinfo`.

Definition at line 340 of file search_node.cc.

10.286.4.33 bool coco::coco::work_node::global_info (basic_alltype & *b*, const char * *i*) const [inline]

This method returns the entry with name `i` in the global data used by the algorithm, setting `b`. It returns whether extraction was successful. See `globalinfo`.

Definition at line 767 of file search_graph.cc.

10.286.4.34 basic_alltype coco::work_node::global_info (const std::string & *i*) const

This method returns the entry with name `i` in the global data used by the algorithm. See `globalinfo`.

Definition at line 348 of file search_node.cc.

10.286.4.35 basic_alltype coco::coco::work_node::global_info (const char * *i*) const [inline]

This method returns the entry with name `i` in the global data used by the algorithm. See `globalinfo`.

Definition at line 774 of file search_graph.cc.

10.286.4.36 gptr<search_node>* coco::coco::search_node::global_model ( ) const [inline, inherited]

This is the accessor method for the global model.

Definition at line 154 of file search_graph.cc.

10.286.4.37 void coco::work_node::init_cnumbers ( )

This method computes the cache values n_bds, n_lin, n_quad, n_poly, and n_other and the log volume.

Definition at line 234 of file search_node.cc.

10.286.4.38 bool coco::coco::full_node::is_delta ( ) const [inline, virtual, inherited]

This method is called to determine whether a search node stores only deltas (delta_node) or a full model (full_node). In case of a full_node it returns `false`.

Reimplemented from coco::coco::search_node.

Definition at line 354 of file search_graph.cc.

10.286.4.39 void coco::coco::search_node::keep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to be the keeper of the annotation `_an`.

Definition at line 173 of file search_graph.cc.

10.286.4.40 void coco::coco::search_node::keep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to be the keeper of all the annotations in `_anv`.

Definition at line 177 of file search_graph.cc.

10.286.4.41 double coco::coco::work_node::log_volume ( ) const [inline]

This method returns the finite approximation to the logarithmic volume of the work_node.

See also

log_vol

Definition at line 754 of file search_graph.cc.

10.286.4.42 void coco::work_node::make_node_order ( )

A call to this method recalculates the node_order for the model DAG stored in the work node.

Definition at line 277 of file search_node.cc.

10.286.4.43 void coco::work_node::make_node_ranges (bool *keep_old_ranges*)

A call to this method adapts the node_ranges vector to the .

Definition at line 256 of file search_node.cc.

10.286.4.44 unsigned int coco::work_node::n (unsigned int *__type*) const [inline]

This method returns the number of constraints of type ___type. Usage, e.g.:

```
n(ex_linear|ex_equality)
```

returns the number of linear equality constraints.

Definition at line 518 of file search_node.hpp.

10.286.4.45 unsigned int coco::coco::full_node::n_annotations ( ) const [inline, inherited]

This method returns the number of annotations stored in this full node.

Definition at line 358 of file search_graph.cc.

10.286.4.46 work_node & coco::work_node::operator= (const work_node & *__w*) [inline]

Standard Assignment Operator

Definition at line 255 of file search_node.hpp.

10.286.4.47 work_node & coco::work_node::operator= (const full_node & *__f*) [inline]

Special Assignment Operator --- be careful when using it!

Reimplemented from coco::coco::full_node.

Definition at line 223 of file search_node.hpp.

10.286.4.48 search_node_relation coco::coco::search_node::parent_relation ( ) const [inline, protected, inherited]

A call to this method returns the relation of this node to its parent in the search graph.

Definition at line 110 of file search_graph.cc.

10.286.4.49 double coco::coco::work_node::reset_gain ( ) [inline]

This method returns the current gain_factor of the work_node and resets it to 1.

Definition at line 779 of file search_graph.cc.

10.286.4.50 void coco::work_node::reset_node_ranges ( ) [inline]

This method resets all unused node ranges to $[-\infty, \infty]$.

Definition at line 508 of file search_node.hpp.

10.286.4.51 void coco::work_node::set_globalinfo ( )

This method extracts the information from the "global info" table and stores it in the `globalinfo` cache of the work_node.

Definition at line 313 of file search_node.cc.

10.286.4.52 void coco::coco::search_node::set_id (const search_node_id & *i*) [inline, inherited]

This is the set method for the search node id of this node.

Definition at line 163 of file search_graph.cc.

10.286.4.53 void coco::coco::search_node::set_rowid (const vdbl::rowid & *i*) [inline, inherited]

This is the set method for the row id of this node.

Definition at line 169 of file search_graph.cc.

10.286.4.54 void coco::search_node::unkeep (const annotation & *_an*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of the annotation `_an`.

Definition at line 82 of file search_node.hpp.

10.286.4.55 void coco::search_node::unkeep (const std::vector< annotation > & *_anv*) [inline, inherited]

A call to this method informs the search node to no longer be the keeper of all the annotations in `_anv`.

Definition at line 93 of file search_node.hpp.

10.286.5 Friends And Related Function Documentation

10.286.5.1 friend class certificate [friend]

Definition at line 808 of file search_graph.cc.

10.286.5.2 friend class certificate_base [friend, inherited]

Definition at line 393 of file search_graph.cc.

10.286.5.3 friend class dag_delta [friend, inherited]

Definition at line 394 of file search_graph.cc.

10.286.5.4 friend class dag_undelta [friend, inherited]

Definition at line 395 of file search_graph.cc.

10.286.5.5 friend class delta [friend]

Definition at line 806 of file search_graph.cc.

10.286.5.6 friend class delta_base [friend, inherited]

Definition at line 392 of file search_graph.cc.

10.286.5.7 work_node operator+ (const work_node & _w, const delta_id & _d) [friend]

This function adds the delta with delta_id _i to work_node _w and returns a new work_node having the delta applied.

Definition at line 298 of file search_node.hpp.

10.286.5.8 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node operator+ (const work_node & _w, const _Ctr< delta_id, _Al > & _d) [friend]

This function adds all the deltas whose delta_ids are in the linear container _d to work_node _w and returns a new work_node having all these deltas applied.

Definition at line 325 of file search_node.hpp.

10.286.5.9 work_node& operator+= (work_node & _w, const delta_id & _d) [friend]

This function adds the delta with delta_id _i to work_node _w changing it in that process. The function returns _w.

Definition at line 311 of file search_node.hpp.

10.286.5.10 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node& operator+= (work_node & _w, const _Ctr< delta_id, _Al > & _d) [friend]

This function adds the deltas whose delta_ids are in the linear container _d to work_node _w changing it in that process. The function returns _w.

Definition at line 341 of file search_node.hpp.

10.286.5.11 work_node operator- (const work_node & _w, const delta_id & _d) [friend]

This function removes the delta with delta_id _i from work_node _w and returns a new work_node having the delta unapplied.

Definition at line 39 of file search_node.cc.

10.286.5.12 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node operator- (const work_node & _w, const _Ctr< delta_id, _Al > & _d) [friend]

This function removes all the deltas whose delta_ids are in the linear container _d from work_node _w and returns a new work_node having all these deltas unapplied.

Definition at line 365 of file search_node.hpp.

10.286.5.13 work_node& operator-= (work_node & _w, const delta_id & _d) [friend]

This function removes (unapplies) the delta with delta_id _i from work_node _w changing it in that process. The function returns _w.

Definition at line 80 of file search_node.cc.

10.286.5.14 template<template< class _Tp, class _TA > class _Ctr, class _Al > work_node& operator-= (work_node & _w, const _Ctr< delta_id, _Al > & _d) [friend]

This function removes (unapplies) the deltas whose delta_ids are in the linear container _d from work_node _w changing it in that process. The function returns _w.

Definition at line 383 of file search_node.hpp.

10.286.5.15 friend class search_graph [friend, inherited]

Definition at line 188 of file search_graph.cc.

10.286.5.16 friend class undelta [friend]

Definition at line 807 of file search_graph.cc.

10.286.5.17 friend class wnc_hook_base [friend]

Definition at line 809 of file search_graph.cc.

10.286.6 Member Data Documentation

10.286.6.1 gptr<vdbl::database>* coco::coco::search_node::__dbase [protected, inherited]

This is a pointer to a global pointer to the search database associated to this search graph.

Definition at line 89 of file search_graph.cc.

10.286.6.2 gptr<search_node>* coco::coco::search_node::__global_model [protected, inherited]

This is a pointer to a global pointer to the top of the search graph. It is NULL for a standalone node or the top of the graph.

Definition at line 86 of file search_graph.cc.

10.286.6.3 vdbl::viewdbase* coco::coco::work_node::__vdb

This is a restricted view to the search database to be used by the inference engines and report modules.

Definition at line 471 of file search_graph.cc.

10.286.6.4 vdbl::viewdbase* coco::coco::work_node::__vdbf

This is a full view to the search database to be used by the inference engines and report modules.

Definition at line 474 of file search_graph.cc.

10.286.6.5 std::vector<annotation> coco::coco::full_node::_ann [inherited]

This is the vector of annotations active at this full node.

Definition at line 283 of file search_graph.cc.

10.286.6.6 vdbl::userid coco::coco::search_node::_dbuser [protected, inherited]

This member stores the database user id under which this search is performed. For the in-memory database this is irrelevant.

Definition at line 92 of file search_graph.cc.

10.286.6.7 search_node_id coco::coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 97 of file search_graph.cc.

10.286.6.8 std::vector<annotation> coco::coco::search_node::_keep [protected, inherited]

The _keep member is a list of all those annotations which are kept by this search node. If the node is destroyed all kept annotations are removed from the search database.

Definition at line 101 of file search_graph.cc.

10.286.6.9 gptr<model>* coco::coco::full_node::_m [protected, inherited]

This is a pointer to a global pointer to the model kept in this full node.

Definition at line 279 of file search_graph.cc.

10.286.6.10 vdbl::rowid coco::coco::search_node::_rid [protected, inherited]

This row id specifies the row in the search info table where the hook information for this search_node is stored

Definition at line 105 of file search_graph.cc.

10.286.6.11 search_node_relation coco::coco::search_node::_snr [protected, inherited]

This member specifies the relation of the search node to its parent(s) in the search graph.

Definition at line 95 of file search_graph.cc.

10.286.6.12 std::list<delta_id>::iterator coco::coco::work_node::deltanew_it

This iterator points to the first 'new' delta of the work_node.

Definition at line 443 of file search_graph.cc.

10.286.6.13 std::list<delta_id> coco::coco::work_node::deltas

This is the list of all deltas, which had to be applied to construct the given work_node. This includes the deltas necessary to extract the search_node to the work_node from the graph, as well as the "new" deltas,

which have been applied by the solution strategy while the work node was analyzed by various inference engines.

Definition at line 440 of file search_graph.cc.

10.286.6.14 vdbl::standard_table* coco::coco::work_node::dtable

This is a pointer to the "deltas" table in the search database.

Definition at line 456 of file search_graph.cc.

10.286.6.15 vdbl::tableid coco::coco::work_node::dtable_id

This is the table identifier of the "deltas" table in the search database.

Definition at line 459 of file search_graph.cc.

10.286.6.16 double coco::coco::work_node::gain_factor

This member keeps an improvement factor (in [0,1]) in bounds sizes since it was last reset.

Definition at line 499 of file search_graph.cc.

10.286.6.17 vdbl::standard_table* coco::coco::work_node::gitable

This is a pointer to the "global info" table in the search database.

Definition at line 461 of file search_graph.cc.

10.286.6.18 vdbl::tableid coco::coco::work_node::gitable_id

This is the table identifier of the "global info" table in the search database.

Definition at line 464 of file search_graph.cc.

10.286.6.19 datamap coco::coco::work_node::globalinfo

This member keeps a cache of the global data used in the algorithm

Definition at line 501 of file search_graph.cc.

10.286.6.20 bool coco::coco::work_node::infeasible

This boolean specifies, whether the problem represented by this work node is known to be infeasible.

Definition at line 492 of file search_graph.cc.

10.286.6.21 double coco::coco::work_node::log_vol

This member keeps a finite approximation to the logarithmic volume of the box formed by the variable bounds of the original variables.

Definition at line 496 of file search_graph.cc.

10.286.6.22 std::vector<unsigned int> coco::coco::work_node::node_order

The node_order vector provides a linear order on the DAG nodes, which is compatible with the DAG structure, i.e., all ancestors of a node are ordered after the node. The vector contains the order as a sequence node numbers of the DAG nodes.

Definition at line 489 of file search_graph.cc.

10.286.6.23 std::vector<interval> coco::coco::work_node::node_ranges

The node_ranges vector keeps the current best known ranges of all nodes in the model DAG. The vector is indexed by the node numbers of the DAG nodes.

Definition at line 484 of file search_graph.cc.

10.286.6.24 std::set<search_node_id> coco::coco::work_node::parents_in_graph

This parameter stores the parents of the work node in the graph

Definition at line 450 of file search_graph.cc.

10.286.6.25 std::map<transaction_number,std::list<std::vector<delta> > > coco::coco::work_node::proposed_splits

This member stores all splits which have been proposed by the various inference engines that can be used to calculate possible splits. Every proposed split (being associated with a split_delta) gets assigned a unique transaction number. This is later used for unapplying the split_delta if necessary. Every split is itself a list of vectors of deltas. Every vector of deltas in the list is associated with one child node generated by the split, and the vector of deltas contains all deltas which have to be applied to the work node for constructing the respective child node from the work node.

Definition at line 531 of file search_graph.cc.

10.286.6.26 std::map<delta_id, transaction_number> coco::coco::work_node::split_delta_ids

This map connects split_delta delta identifiers with transaction numbers in the proposed_splits map.

Definition at line 534 of file search_graph.cc.

10.286.6.27 std::list<delta_id>::iterator coco::coco::work_node::splithook_deltanew_it

This iterator points to the first 'new' delta of the work_node generated due to splits and hooks.

Definition at line 447 of file search_graph.cc.

10.286.6.28 std::map<delta_id,undelta> coco::coco::work_node::undeltas

This map connects delta identifiers of previously applied deltas with their corresponding undelta information.

See also

undelta

Definition at line 454 of file search_graph.cc.

10.286.6.29 work_node_context* coco::coco::work_node::wnc

This is the work node context corresponding to the current work node.

Definition at line 468 of file search_graph.cc.

The documentation for this class was generated from the following files:

- search_node.h
- search_node.cc
- search_node.hpp

10.287 coco::work_node_comp_hook Class Reference

The work_node_comp_hook class (work node computation hook)

`#include <comp_hook.h>`

Public Member Functions

- work_node_comp_hook (const wnc_hook_base &w)
- work_node_comp_hook (const work_node_comp_hook &wh)
- ~work_node_comp_hook ()
- work_node_comp_hook & operator= (const work_node_comp_hook &w)
- void operator() (const work_node &wn, dbt_row &dbr, std::list< delta > *add_ds=NULL, std::list< certificate > *add_cs=NULL)
- bool init_columns (vdbl::standard_table &stable)
- bool drop_columns (vdbl::standard_table &stable)
- const std::string & name () const

10.287.1 Detailed Description

The work_node_comp_hook class is used to compute info about work_nodes just when the work node is split and stored in the search graph. It is, like the delta class, designed as a wrapper class in order to overcome the problem that copy constructors cannot be overloaded. Work node computation hooks are stored in the search_graph, can be registered and released with search_graph methods.

10.287.2 Constructor & Destructor Documentation

10.287.2.1 coco::work_node_comp_hook::work_node_comp_hook (const wnc_hook_base & *w*) [inline]

Constructor, which constructs a work node computation hook from a wnc_hook_base. The clone operation for the wnc_hook_base will be called in that process.

Definition at line 193 of file comp_hook.h.

10.287.2.2 coco::work_node_comp_hook::work_node_comp_hook (const work_node_comp_hook & *wh*) [inline]

Copy Constructor, which constructs a new work node computation hook from an existing one. The clone operation for the wnc_hook_base will be called in that process, effectively overloading the copy constructor for the wnc_hook_base.

Definition at line 198 of file comp_hook.h.

10.287.2.3 coco::work_node_comp_hook::~work_node_comp_hook ( ) [inline]

Standard Destructor calling `delete` on `__wh`.

Definition at line 203 of file comp_hook.h.

10.287.3 Member Function Documentation

10.287.3.1 bool coco::work_node_comp_hook::drop_columns (vdbl::standard_table & *stable*) [inline]

Upon unregistering this hook, destroy the column(s) in the "search info" table in which the result(s) of this hook are stored.

Definition at line 227 of file comp_hook.h.

10.287.3.2 bool coco::work_node_comp_hook::init_columns (vdbl::standard_table & *stable*) [inline]

Upon registering this hook, initialize the column(s) in the "search info" table in which the result(s) of this hook are stored.

Definition at line 222 of file comp_hook.h.

10.287.3.3 const std::string & coco::work_node_comp_hook::name ( ) const [inline]

Return the identifier string for this work node computation hook.

Definition at line 232 of file comp_hook.h.

10.287.3.4 void coco::work_node_comp_hook::operator() (const work_node & *wn*, dbt_row & *dbr*, std::list< delta > * *add_ds* = `NULL`, std::list< certificate > * *add_cs* = `NULL`) [inline]

The evaluation operator of this computation hook. It stores the result of the hook applied to work_node `wn` in dbt_row `dbr`, which in return is stored in the "search info" table of the search database. The hook can compute additional deltas and add them to `add_ds`, which are applied to the work node during the split.

Definition at line 216 of file comp_hook.h.

10.287.3.5 work_node_comp_hook & coco::work_node_comp_hook::operator= (const work_node_comp_hook & *w*) [inline]

Assignment operator, which uses the clone operation for the wnc_hook_base, effectively overloading the assignment operator for the wnc_hook_base.

Definition at line 209 of file comp_hook.h.

The documentation for this class was generated from the following file:

- comp_hook.h

10.288 coco::work_node_context Class Reference

The evaluation context when retrieving from the search database.

`#include <wnodectx.h>`

Public Member Functions

- work_node_context ()
- work_node_context (const work_node *_i)
- work_node_context (const work_node_context &_w)
- virtual ~work_node_context ()
- work_node_context & operator= (const work_node_context &_w)
- const work_node * wn () const

10.288.1 Detailed Description

This class provides the evaluation context when a view to the search database is created for an inference engine. This mainly influences the stored procedures (vdbl::method).

10.288.2 Constructor & Destructor Documentation

10.288.2.1 coco::work_node_context::work_node_context ( ) [inline]

Standard Constructor

Definition at line 50 of file wnodectx.h.

10.288.2.2 coco::work_node_context::work_node_context (const work_node * _i) [inline]

Constructor which initializes the work_node to _i.

Definition at line 52 of file wnodectx.h.

10.288.2.3 coco::work_node_context::work_node_context (const work_node_context & _w) [inline]

Standard Copy Constructor

Definition at line 54 of file wnodectx.h.

10.288.2.4 virtual coco::work_node_context::~work_node_context ( ) [inline, virtual]

Standard Destructor

Definition at line 56 of file wnodectx.h.

10.288.3 Member Function Documentation

10.288.3.1 work_node_context& coco::work_node_context::operator= (const work_node_context & _w) [inline]

Standard Assignment Operator

Definition at line 59 of file wnodectx.h.

10.288.3.2 const work_node* coco::work_node_context::wn ( ) const [inline]

This method returns a global pointer to the work_node of this context.

Definition at line 64 of file wnodectx.h.

The documentation for this class was generated from the following file:

- wnodectx.h

10.289 coco::coco::work_node_context Class Reference

The evaluation context when retrieving from the search database.

Public Member Functions

- work_node_context ()
- work_node_context (const work_node *_i)
- work_node_context (const work_node_context &_w)
- virtual ~work_node_context ()
- work_node_context & operator= (const work_node_context &_w)
- const work_node * wn () const

10.289.1 Detailed Description

This class provides the evaluation context when a view to the search database is created for an inference engine. This mainly influences the stored procedures (vdbl::method).

10.289.2 Constructor & Destructor Documentation

10.289.2.1 coco::coco::work_node_context::work_node_context ( ) [inline]

Standard Constructor

Definition at line 50 of file search_graph.cc.

10.289.2.2 coco::coco::work_node_context::work_node_context (const work_node * _i) [inline]

Constructor which initializes the work_node to _i.

Definition at line 52 of file search_graph.cc.

10.289.2.3 coco::coco::work_node_context::work_node_context (const work_node_context & _w) [inline]

Standard Copy Constructor

Definition at line 54 of file search_graph.cc.

10.289.2.4 virtual coco::coco::work_node_context::~work_node_context ( ) [inline, virtual]

Standard Destructor

Definition at line 56 of file search_graph.cc.

10.289.3 Member Function Documentation

10.289.3.1 work_node_context& coco::coco::work_node_context::operator= (const work_node_context & _w) [inline]

Standard Assignment Operator

Definition at line 59 of file search_graph.cc.

10.289.3.2 const work_node* coco::coco::work_node_context::wn ( ) const [inline]

This method returns a global pointer to the work_node of this context.

Definition at line 64 of file search_graph.cc.

The documentation for this class was generated from the following file:

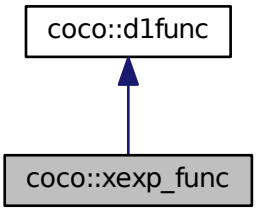
- wnodectx.h

10.290 coco::xexp_func Class Reference

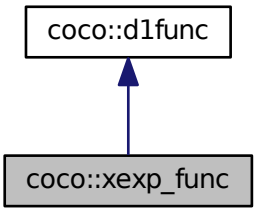
The xexp_func class (one dimensional function)

`#include <xexp.h>`

Inheritance diagram for coco::xexp_func:



Collaboration diagram for coco::xexp_func:



Public Types

- enum d1func_point_t { d1fp_unknown = -2, d1fp_regular = -1, d1fp_extremum = 0, d1fp_evenpoleup = 1, d1fp_evenpoledn = 2, d1fp_oddpoleup = 3, d1fp_oddpoledn = 4, d1fp_jumpldef = 5, d1fp_jumpludef = 6, d1fp_point = 7, d1fp_undefined = 8, d1fp_wild = 9, d1fp_jumprdef = 10, d1fp_jumprudef = 11 }
- enum d1solve_t { d1solve_0, d1solve_1, d1solve_2 }
- typedef std::triple< interval, interval, d1func_point_t > point_info
- typedef std::vector< point_info > point_list

Public Member Functions

- virtual uint64_t thash ()
- virtual uint64_t chash ()
- virtual const char * func_name () const

- virtual std::string dag_out () const
- virtual std::string print_params () const
- virtual double eval (double x, unsigned int k) const
- virtual std::pair< double, double > evald (double x, unsigned int k) const
- virtual std::triple< double, double, double > evalt (double x, unsigned int k) const
- virtual void evalf (double x, double *r, unsigned int k) const
- virtual interval imperfect_eval (const interval &x, unsigned int k) const
- virtual bool operator== (const d1func &b) const
- virtual void raise_order (unsigned int neworder)
- xexp_func (double _c, unsigned int MAXORDER=5)
- virtual int differentiability (const interval &i, unsigned int k) const
- virtual diffNumber eval (const diffNumber &x, unsigned int k) const
- virtual hessNumber eval (const hessNumber &x, unsigned int k) const
- virtual interval eval (const interval &x, unsigned int k) const
- virtual diffI eval (const diffI &x, unsigned int k) const
- virtual ihessNumber eval (const ihessNumber &x, unsigned int k) const
- virtual Islope eval (const Islope &x, unsigned int k) const
- virtual interval_set eval (const interval_set &x, unsigned int k) const
- virtual std::ostream & print_add (std::ostream &o) const
- virtual interval eval_slp (const interval &z, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &y, const interval &x, unsigned int k) const
- virtual interval eval_slp2 (const interval &z, const interval &x, unsigned int k) const
- virtual void internal_eval_d1fp (const diffI &x, diffI &res, const std::vector< d1func_point_t > &ptin, std::vector< d1func_point_t > &ptres, unsigned int order) const
- virtual interval_set intersect_inv (const interval &x, const interval &r, unsigned int k) const
- virtual interval_set intersect_inv (const interval_set &x, const interval_set &r, unsigned int k) const
- virtual convex_info convexity (const interval &i, unsigned int k) const
- virtual d1func_monotonicity_t monotonicity (const interval &i, unsigned int k) const
- virtual int degree_update (int in_degree) const
- virtual bool order_is_supported (unsigned int k) const
- virtual unsigned int supported_eval_order () const
- virtual const func_info & ff (unsigned int k) const
- virtual bool operator!= (const d1func &b) const
- const char * toString (const d1func::d1func_point_t &t) const

Public Attributes

- volatile int MAXLEN

Protected Attributes

- double dod_left
- double dod_right
- int basic_diff
- std::vector< func_info > _f

10.290.1 Detailed Description

xexp_func is a one dimensional function defined by x∗exp(c∗x), where c is a real constant.

10.290.2 Member Typedef Documentation

10.290.2.1 typedef std::triple<interval, interval, d1func_point_t> coco::d1func::point_info [inherited]

The information on one point .first is the point, .second the value

Definition at line 110 of file d1func.h.

10.290.2.2 typedef std::vector<point_info> coco::d1func::point_list [inherited]

a list of points for function value, first and second derivatives

Definition at line 113 of file d1func.h.

10.290.3 Member Enumeration Documentation

10.290.3.1 enum coco::d1func::d1func_point_t [inherited]

The possible classes for special points in the lists: d1fp_regular: At this point the function is regular (for internal use only!) d1fp_extremum: The point is a local extremum d1fp_evenpoleup: At this point the function has a positive even pole d1fp_evenpoledn: At this point the function has a negative even pole d1fp_oddpoleup: At this point the function has an odd pole like 1/x at 0 d1fp_oddpoledn: At this point the function has an odd pole like -1/x at 0 d1fp_jumpldef: At this point the function has a jump (left side defined) d1fp_jumpludef: At this point the function has a jump (left side not defined) d1fp_point-: At this point the function has the specified value d1fp_undefined: In the given interval the function is undefined d1fp_wild: In the given interval the function has "wild" behaviour d1fp_jumprdef: At this point the function has a jump (right side defined) d1fp_jumprudef: At this point the function has a jump (right side not defined) d1fp_unknown: At this point the function has a unknown behaviour (for internal use only!)

Enumerator:

d1fp_unknown

d1fp_regular

d1fp_extremum

d1fp_evenpoleup

d1fp_evenpoledn

d1fp_oddpoleup

d1fp_oddpoledn

d1fp_jumpldef

d1fp_jumpludef

d1fp_point

d1fp_undefined

d1fp_wild
d1fp_jumprdef
d1fp_jumprudef

Definition at line 95 of file d1func.h.

10.290.3.2 enum coco::d1func::d1solve_t [inherited]

The possible functions for 1D Newton solving: d1solve_0: f(x) = c d1solve_1: f(x) - f(z) - f'(x)(x-z) = 0 d1solve_2: 1/(x-z)(f(x)-f(z))(1+(x-w)(x-z)) - (f(w)-f(z))/(w-z) - (x-w)/(x-z) f'(x) = 0

Enumerator:

d1solve_0
d1solve_1
d1solve_2

Definition at line 106 of file d1func.h.

10.290.4 Constructor & Destructor Documentation

10.290.4.1 coco::xexp_func::xexp_func (double *_c*, unsigned int *MAXORDER* = 5) [inline]

Definition at line 336 of file xexp.h.

10.290.5 Member Function Documentation

10.290.5.1 virtual uint64_t coco::xexp_func::chash ( ) [inline, virtual]

the const hash value for hash calculation in the DAG

Implements coco::d1func.

Definition at line 54 of file xexp.h.

10.290.5.2 convex_info coco::d1func::convexity (const interval & *i*, unsigned int *k*) const [virtual, inherited]

return convexity information on `i`.

Definition at line 2308 of file d1func.cc.

10.290.5.3 virtual std::string coco::xexp_func::dag_out ( ) const [inline, virtual]

the parameters written in DAG format (it must contain necessary ':'s)

Reimplemented from coco::d1func.

Definition at line 60 of file xexp.h.

10.290.5.4 virtual int coco::d1func::degree_update (int *in_degree*) const [inline, virtual, inherited]

return degree information for in-argument degree `in_degree`.

Definition at line 457 of file d1func.h.

10.290.5.5 virtual int coco::xexp_func::differentiability (const interval & *i*, unsigned int *k*) const [inline, virtual]

return differentiability information on `i`.

Reimplemented from coco::d1func.

Definition at line 342 of file xexp.h.

10.290.5.6 virtual double coco::xexp_func::eval (double *x*, unsigned int *k*) const [inline, virtual]

the imperfect function evaluations

Implements coco::d1func.

Definition at line 66 of file xexp.h.

10.290.5.7 diffNumber coco::d1func::eval (const diffNumber & *x*, unsigned int *k*) const [virtual, inherited]

the diffNumber evaluation

Reimplemented in coco::dag_d1func.

Definition at line 349 of file d1func.cc.

10.290.5.8 hessNumber coco::d1func::eval (const hessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the hessNumber evaluation

Definition at line 334 of file d1func.cc.

10.290.5.9 interval coco::d1func::eval (const interval & *x_in*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 390 of file d1func.cc.

10.290.5.10 diffI coco::d1func::eval (const diffI & *x*, unsigned int *k*) const [virtual, inherited]

the interval diffNumber evaluation

Definition at line 715 of file d1func.cc.

10.290.5.11 ihessNumber coco::d1func::eval (const ihessNumber & *x*, unsigned int *k*) const [virtual, inherited]

the interval hessNumber evaluation

the hessNumber evaluation

Definition at line 411 of file d1func.cc.

10.290.5.12 Islope coco::d1func::eval (const Islope & *x*, unsigned int *k*) const [virtual, inherited]

the interval slope evaluation

the interval diffNumber evaluation

Definition at line 429 of file d1func.cc.

10.290.5.13 interval_set coco::d1func::eval (const interval_set & *x*, unsigned int *k*) const [virtual, inherited]

the range evaluation

Definition at line 794 of file d1func.cc.

10.290.5.14 interval coco::d1func::eval_slp (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval first order slope evaluation at center z

Definition at line 1403 of file d1func.cc.

10.290.5.15 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *y*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at centers z and y

Definition at line 1984 of file d1func.cc.

10.290.5.16 interval coco::d1func::eval_slp2 (const interval & *z*, const interval & *x*, unsigned int *k*) const [virtual, inherited]

the interval second order slope evaluation at double center z

Definition at line 1991 of file d1func.cc.

10.290.5.17 virtual std::pair<double,double> coco::xexp_func::evald (double *x*, unsigned int *k*) const [inline, virtual]

the point and derivative evaluation

Reimplemented from coco::d1func.

Definition at line 86 of file xexp.h.

10.290.5.18 virtual void coco::xexp_func::evalf (double *x*, double * *r*, unsigned int *k*) const [inline, virtual]

the point and derivative evaluation

Reimplemented from coco::d1func.

Definition at line 174 of file xexp.h.

10.290.5.19 virtual std::triple<double,double,double> coco::xexp_func::evalt (double *x*, unsigned int *k*) const [inline, virtual]

the point and derivative evaluation

Reimplemented from coco::d1func.

Definition at line 125 of file xexp.h.

10.290.5.20 virtual const func_info& coco::d1func::ff (unsigned int *k*) const [inline, virtual, inherited]

retrieve the k th special points list

Definition at line 469 of file d1func.h.

10.290.5.21 virtual const char* coco::xexp_func::func_name ( ) const [inline, virtual]

the name of the d1func.

Implements coco::d1func.

Definition at line 57 of file xexp.h.

10.290.5.22 virtual interval coco::xexp_func::imperfect_eval (const interval & *x*, unsigned int *k*) const [inline, virtual]

the imperfect function evaluations

Implements coco::d1func.

Definition at line 233 of file xexp.h.

10.290.5.23 void coco::d1func::internal_eval_d1fp (const diffI & *x*, diffI & *res*, const std::vector< d1func_point_t > & *ptin*, std::vector< d1func_point_t > & *ptres*, unsigned int *k*) const [virtual, inherited]

perform internal evaluations to update eventual point lists of "higher" functions.

Definition at line 168 of file d1func.cc.

10.290.5.24 interval_set coco::d1func::intersect_inv (const interval & *x*, const interval & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1708 of file d1func.cc.

10.290.5.25 interval_set coco::d1func::intersect_inv (const interval_set & *x*, const interval_set & *r*, unsigned int *k*) const [virtual, inherited]

back propagation

Definition at line 1778 of file d1func.cc.

10.290.5.26 d1func_monotonicity_t coco::d1func::monotonicity (const interval & *x*, unsigned int *k*) const [virtual, inherited]

return monotonicity information on `i`.

Definition at line 2185 of file d1func.cc.

10.290.5.27 virtual bool coco::d1func::operator!= (const d1func & *b*) const [inline, virtual, inherited]

the other comparison operator

Definition at line 476 of file d1func.h.

10.290.5.28 virtual bool coco::xexp_func::operator== (const d1func & *b*) const [inline, virtual]

the comparison operator

Reimplemented from coco::d1func.

Definition at line 251 of file xexp.h.

10.290.5.29 virtual bool coco::d1func::order_is_supported (unsigned int *k*) const [inline, virtual, inherited]

check whether evaluation of this order is supported.

Definition at line 460 of file d1func.h.

10.290.5.30 virtual std::ostream& coco::d1func::print_add (std::ostream & *o*) const [inline, virtual, inherited]

additional info written in DAG format (it must contain dag lines)

Reimplemented in coco::dag_d1func.

Definition at line 208 of file d1func.h.

10.290.5.31 virtual std::string coco::xexp_func::print_params ( ) const [inline, virtual]

the parameters written in printable form

Reimplemented from coco::d1func.

Definition at line 63 of file xexp.h.

10.290.5.32 virtual void coco::xexp_func::raise_order (unsigned int *neworder*) [inline, virtual]

raise the order of the supported function evaluation to neworder

Reimplemented from coco::d1func.

Definition at line 258 of file xexp.h.

10.290.5.33 virtual unsigned int coco::d1func::supported_eval_order ( ) const [inline, virtual, inherited]

return the maximal evaluation order supported.

Definition at line 466 of file d1func.h.

10.290.5.34 virtual uint64_t coco::xexp_func::thash ( ) [inline, virtual]

the pure hash value for hash calculation in the DAG

Implements coco::d1func.

Definition at line 51 of file xexp.h.

10.290.5.35 const char * coco::d1func::toString (const d1func::d1func_point_t & *t*) const [inherited]

Definition at line 2415 of file d1func.cc.

10.290.6 Member Data Documentation

10.290.6.1 std::vector<func_info> coco::d1func::_f [protected, inherited]

the function info tables for function values and derivatives.

Definition at line 163 of file d1func.h.

10.290.6.2 int coco::d1func::basic_diff [protected, inherited]

basic differentiability

Definition at line 160 of file d1func.h.

10.290.6.3 double coco::d1func::dod_left [protected, inherited]

the left limit of the DOD

Definition at line 156 of file d1func.h.

10.290.6.4 double coco::d1func::dod_right [protected, inherited]

the right limit of the DOD

Definition at line 158 of file d1func.h.

10.290.6.5 volatile int coco::d1func::MAXLEN [inherited]

maximal lenght of the interval set returned

Definition at line 167 of file d1func.h.

The documentation for this class was generated from the following file:

- xexp.h

10.291 coco::xxxNumber_eval Class Reference

Forward function range evaluation.

`#include <numfeval.h>`

Inheritance diagram for coco::xxxNumber_eval:

Collaboration diagram for coco::xxxNumber_eval:

Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker

Public Member Functions

- xxxNumber_eval (const std::vector< xxxNumber_t > &__x, const variable_indicator &__v, const model &__m, std::vector< xxxNumber_t > *__c)
- xxxNumber_eval (const xxxNumber_eval &__v)
- ~xxxNumber_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_point (const std::vector< xxxNumber_t > &__x, const variable_indicator &__v)
- void new_interval (const std::vector< xxxNumber_t > &__x, const variable_indicator &__v)
- int preorder (const node_data_type &__data)
- void postorder (const node_data_type &__data)
- int collect (const node_data_type &__data, const return_value &__rval)
- int vcollect (const return_value &__rval)
- return_value value ()
- return_value vvalue ()

- void vinit ()
- virtual int initialize (const node_data_type &__data)
- virtual void calculate (const node_data_type &__data)
- virtual void retrieve_from_cache (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)
- virtual int update (const node_data_type &__data, const return_value &__rval)
- virtual int update (const return_value &__rval)

- void initialize ()
- int initialize (const expression_node &__data)
- void calculate (const expression_node &__data)
- void retrieve_from_cache (const expression_node &__data)
- int update (const xxxNumber_t &__rval)
- int update (const expression_node &__data, const xxxNumber_t &__rval)
- xxxNumber_t calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

10.291.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural xxxNumber extension.

10.291.2 Member Typedef Documentation

10.291.2.1 typedef _Base::const_walker coco::coco::cached_forward_evaluator_base::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Definition at line 724 of file search_graph.cc.

10.291.2.2 typedef _Base::node_data_type coco::coco::cached_forward_evaluator_base::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Definition at line 720 of file search_graph.cc.

10.291.2.3 typedef _Base::return_value coco::coco::cached_forward_evaluator_base::return_value [inherited]

This type is the result type of the evaluator.

Definition at line 722 of file search_graph.cc.

10.291.3 Constructor & Destructor Documentation

10.291.3.1 coco::xxxNumber_eval::xxxNumber_eval (const std::vector< xxxNumber_t > & __x, const variable_indicator & __v, const model & __m, std::vector< xxxNumber_t > * __c) [inline]

Constructor: ___x is the box over which the function range shall be evaluated, ___v is a variable indicator specifying the variables that have changed value since the last evaluation, ___m specifies the DAG, and ___c the cache (may be NULL). The boolean do_i determines whether the natural xxxNumber extension will use known node ranges for reducing the ranges.

Definition at line 184 of file numfeval.h.

10.291.3.2 coco::xxxNumber_eval::xxxNumber_eval (const xxxNumber_eval & __v) [inline]

Standard Copy Constructor

Definition at line 261 of file numfeval.h.

10.291.3.3 coco::xxxNumber_eval::~xxxNumber_eval ( ) [inline]

Standard Destructor

Definition at line 264 of file numfeval.h.

10.291.4 Member Function Documentation

10.291.4.1 void coco::xxxNumber_eval::calculate (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 408 of file numfeval.h.

10.291.4.2 virtual void coco::coco::cached_forward_evaluator_base::calculate (const node_data_type & __data) [inline, virtual, inherited]

This method is called right after all children of a node have been visited. The ___data parameter contains the node data of the graph node being visited.

Definition at line 797 of file search_graph.cc.

10.291.4.3 xxxNumber_t coco::xxxNumber_eval::calculate_value (bool eval_all) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< xxxNumber_eval_type, expression_node, xxxNumber_t, expression_const_walker >.

Definition at line 894 of file numfeval.h.

10.291.4.4 virtual void coco::coco::cached_forward_evaluator_base::cleanup (const node_data_type & __data) [inline, virtual, inherited]

The cleanup method is called just before calculate_value and should be used to clean up dynamically allocated data. The __data parameter contains the node data of the graph node being visited.

Definition at line 805 of file search_graph.cc.

10.291.4.5 int coco::coco::cached_forward_evaluator_base::collect (const node_data_type & __data, const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each normal node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 751 of file search_graph.cc.

10.291.4.6 void coco::xxxNumber_eval::initialize ( ) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::coco::cached_forward_evaluator_base< xxxNumber_eval_type, expression_node, xxxNumber_t, expression_const_walker >.

Definition at line 306 of file numfeval.h.

10.291.4.7 int coco::xxxNumber_eval::initialize (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 308 of file numfeval.h.

10.291.4.8 virtual int coco::coco::cached_forward_evaluator_base::initialize (const node_data_type & __data) [inline, virtual, inherited]

This method is called at a normal node before any children are visited. The __data parameter contains the node data of the graph node being visited. The return value determines how the graph walk proceeds.

<0	perform a short-cut (the short_cut_to method will be called),
0	don't visit the children, proceed with postorder,
>0	continue with the walk by visiting the children.

Definition at line 793 of file search_graph.cc.

10.291.4.9 bool coco::xxxNumber_eval::is_cached (const node_data_type & __data) [inline, protected, virtual]

This function determines, whether the range for this node is already available.

Reimplemented from coco::coco::cached_forward_evaluator_base< xxxNumber_eval_type, expression_node, xxxNumber_t, expression_const_walker >.

Definition at line 114 of file numfeval.h.

10.291.4.10 void coco::xxxNumber_eval::new_interval (const std::vector< xxxNumber_t > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation point to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 287 of file numfeval.h.

10.291.4.11 void coco::xxxNumber_eval::new_point (const std::vector< xxxNumber_t > & __x, const variable_indicator & __v) [inline]

This method changes the evaluation point to __x. The parameter __v specifies which variables have changed value since the last evaluation.

Definition at line 278 of file numfeval.h.

10.291.4.12 void coco::coco::cached_forward_evaluator_base::postorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right after all children of a node have been visited. It is translated into calls to calculate and cleanup.

Definition at line 745 of file search_graph.cc.

10.291.4.13 int coco::coco::cached_forward_evaluator_base::preorder (const node_data_type & __data) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before any children of a node are visited. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Definition at line 733 of file search_graph.cc.

10.291.4.14 void coco::xxxNumber_eval::retrieve_from_cache (const expression_node & __data) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 417 of file numfeval.h.

10.291.4.15 virtual void coco::coco::cached_forward_evaluator_base::retrieve_from_cache (const node_data_type & __data) [inline, virtual, inherited]

The retrieve_from_cache method is called for retrieving the result for this node from the cache or calculate it without visiting the node's children.

Definition at line 801 of file search_graph.cc.

10.291.4.16 expression_const_walker coco::xxxNumber_eval::short_cut_to (const expression_node & __data) [inline]

NOP version, not needed

Definition at line 267 of file numfeval.h.

10.291.4.17 int coco::xxxNumber_eval::update (const xxxNumber_t & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 424 of file numfeval.h.

10.291.4.18 int coco::xxxNumber_eval::update (const expression_node & __data, const xxxNumber_t & __rval) [inline]

This is an evaluator method, as defined for the various evaluators.

Definition at line 433 of file numfeval.h.

10.291.4.19 virtual int coco::coco::cached_forward_evaluator_base::update (const node_data_type & __data, const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 815 of file search_graph.cc.

10.291.4.20 virtual int coco::coco::cached_forward_evaluator_base::update (const return_value & __rval) [inline, virtual, inherited]

The update method is called for each virtual node everytime a child node has been visited passing the return value of the child. The `__data` parameter contains the node data of the graph node being visited. The return value has the following effect:

<0	stop visiting children of this node,
0	continue with the graph walk downwards the next child,
>0	skip as many children.

Definition at line 827 of file search_graph.cc.

10.291.4.21 return_value coco::coco::cached_forward_evaluator_base::value ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a normal node to retrieve the return value. It is translated to a call to calculate_value with parameter `false`.

Definition at line 763 of file search_graph.cc.

10.291.4.22 int coco::coco::cached_forward_evaluator_base::vcollect (const return_value & __rval) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called for each virtual node everytime a child node has been visited passing the return value of the child. It is translated to a call to update.

Definition at line 757 of file search_graph.cc.

10.291.4.23 void coco::coco::cached_forward_evaluator_base::vinit ( ) [inline, inherited]

This method is needed by a visitor using recursive_short_cut_walk for traversing the graph. It is called before performing any graph walk when at a virtual node. It is translated to initialize.

Definition at line 772 of file search_graph.cc.

10.291.4.24 return_value coco::coco::cached_forward_evaluator_base::vvalue ( ) [inline, inherited]

This method is needed by a visitor (see VGTL documentation) and is called right before the end of the visit of a virtual node to retrieve the return value. It is translated to a call to calculate_value with parameter `true`.

Definition at line 768 of file search_graph.cc.

The documentation for this class was generated from the following file:

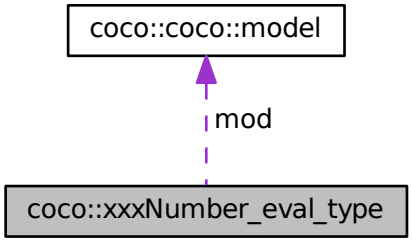
- numfeval.h

10.292 coco::xxxNumber_eval_type Struct Reference

Visitor data for xxxNumber_eval.

`#include <numfeval.h>`

Collaboration diagram for coco::xxxNumber_eval_type:



Public Attributes

- const std::vector< xxxNumber_t > * x
- std::vector< xxxNumber_t > * cache
- const model * mod

- void * p
- xxxNumber_t r
- unsigned int n

10.292.1 Detailed Description

This class is the visitor data type for the xxxNumber_eval evaluator.

10.292.2 Member Data Documentation

10.292.2.1 std::vector<xxxNumber_t>* coco::xxxNumber_eval_type::cache

function range the function range cache

Definition at line 76 of file numfeval.h.

10.292.2.2 const model* coco::xxxNumber_eval_type::mod

the DAG

Definition at line 77 of file numfeval.h.

10.292.2.3 unsigned int coco::xxxNumber_eval_type::n

children counter

Definition at line 80 of file numfeval.h.

10.292.2.4 void* coco::xxxNumber_eval_type::p

additional data for

Definition at line 78 of file numfeval.h.

10.292.2.5 xxxNumber_t coco::xxxNumber_eval_type::r

return value

Definition at line 79 of file numfeval.h.

10.292.2.6 const std::vector<xxxNumber_t>* coco::xxxNumber_eval_type::x

Definition at line 75 of file numfeval.h.

The documentation for this struct was generated from the following file:

- numfeval.h

10.293 yy_buffer_state Struct Reference

Public Attributes

- std::istream * yy_input_file
- char * yy_ch_buf
- char * yy_buf_pos
- yy_size_t yy_buf_size
- int yy_n_chars
- int yy_is_our_buffer
- int yy_is_interactive
- int yy_at_bol
- int yy_bs_lineno
- int yy_bs_column
- int yy_fill_buffer
- int yy_buffer_status

10.293.1 Detailed Description

Definition at line 189 of file dbcompare_sortorder_scanner.cc.

10.293.2 Member Data Documentation

10.293.2.1 int yy_buffer_state::yy_at_bol

Definition at line 239 of file dbcompare_expression_scanner.cc.

10.293.2.2 int yy_buffer_state::yy_bs_column

The column count.

Definition at line 242 of file dbcompare_expression_scanner.cc.

10.293.2.3 int yy_buffer_state::yy_bs_lineno

The line count.

Definition at line 241 of file dbcompare_expression_scanner.cc.

10.293.2.4 char * yy_buffer_state::yy_buf_pos

Definition at line 210 of file dbcompare_expression_scanner.cc.

10.293.2.5 yy_size_t yy_buffer_state::yy_buf_size

Definition at line 215 of file dbcompare_expression_scanner.cc.

10.293.2.6 int yy_buffer_state::yy_buffer_status

Definition at line 249 of file dbcompare_expression_scanner.cc.

10.293.2.7 char * yy_buffer_state::yy_ch_buf

Definition at line 209 of file dbcompare_expression_scanner.cc.

10.293.2.8 int yy_buffer_state::yy_fill_buffer

Definition at line 247 of file dbcompare_expression_scanner.cc.

10.293.2.9 std::istream * yy_buffer_state::yy_input_file

Definition at line 207 of file dbcompare_expression_scanner.cc.

10.293.2.10 int yy_buffer_state::yy_is_interactive

Definition at line 233 of file dbcompare_expression_scanner.cc.

10.293.2.11 int yy_buffer_state::yy_is_our_buffer

Definition at line 226 of file dbcompare_expression_scanner.cc.

10.293.2.12 int yy_buffer_state::yy_n_chars

Definition at line 220 of file dbcompare_expression_scanner.cc.

The documentation for this struct was generated from the following files:

- dbcompare_expression_scanner.cc
- dbcompare_sortorder_scanner.cc
- Number_scanner.cc

10.294 yy_trans_info Struct Reference

Public Attributes

- flex_int32_t yy_verify
- flex_int32_t yy_nxt

10.294.1 Detailed Description

Definition at line 326 of file dbcompare_sortorder_scanner.cc.

10.294.2 Member Data Documentation

10.294.2.1 flex_int32_t yy_trans_info::yy_nxt

Definition at line 344 of file dbcompare_expression_scanner.cc.

10.294.2.2 flex_int32_t yy_trans_info::yy_verify

Definition at line 343 of file dbcompare_expression_scanner.cc.

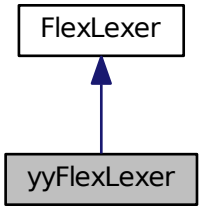
The documentation for this struct was generated from the following files:

- dbcompare_expression_scanner.cc
- dbcompare_sortorder_scanner.cc
- Number_scanner.cc

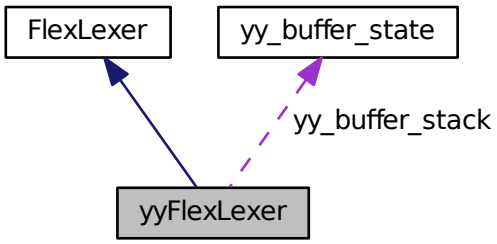
10.295 yyFlexLexer Class Reference

`#include <FlexLexer.h>`

Inheritance diagram for yyFlexLexer:



Collaboration diagram for yyFlexLexer:



Public Member Functions

- yyFlexLexer (FLEX_STD istream *arg_yyin=0, FLEX_STD ostream *arg_yyout=0)
- virtual ~yyFlexLexer ()
- void yy_switch_to_buffer (struct yy_buffer_state *new_buffer)
- struct yy_buffer_state * yy_create_buffer (FLEX_STD istream *s, int size)
- void yy_delete_buffer (struct yy_buffer_state *b)
- void yyrestart (FLEX_STD istream *s)
- void yypush_buffer_state (struct yy_buffer_state *new_buffer)
- void yypop_buffer_state (void)
- virtual int yylex ()
- virtual void switch_streams (FLEX_STD istream *new_in, FLEX_STD ostream *new_out)
- const char * YYText ()
- int YYLeng ()
- int yylex (FLEX_STD istream *new_in, FLEX_STD ostream *new_out=0)
- int lineno () const
- int debug () const
- void set_debug (int flag)

Protected Member Functions

- virtual int LexerInput (char *buf, int max_size)
- virtual void LexerOutput (const char *buf, int size)
- virtual void LexerError (const char *msg)
- void yyunput (int c, char *buf_ptr)
- int yyinput ()
- void yy_load_buffer_state ()
- void yy_init_buffer (struct yy_buffer_state *b, FLEX_STD istream *s)
- void yy_flush_buffer (struct yy_buffer_state *b)
- void yy_push_state (int new_state)
- void yy_pop_state ()
- int yy_top_state ()
- yy_state_type yy_get_previous_state ()
- yy_state_type yy_try_NUL_trans (yy_state_type current_state)
- int yy_get_next_buffer ()
- void yyensure_buffer_stack (void)

Protected Attributes

- int yy_start_stack_ptr
- int yy_start_stack_depth
- int * yy_start_stack
- FLEX_STD istream * yyin
- FLEX_STD ostream * yyout
- char yy_hold_char
- int yy_n_chars
- char * yy_c_buf_p
- int yy_init
- int yy_start

- int yy_did_buffer_switch_on_eof
- size_t yy_buffer_stack_top
- size_t yy_buffer_stack_max
- struct yy_buffer_state ** yy_buffer_stack
- yy_state_type yy_last_accepting_state
- char * yy_last_accepting_cpos
- yy_state_type * yy_state_buf
- yy_state_type * yy_state_ptr
- char * yy_full_match
- int * yy_full_state
- int yy_full_lp
- int yy_lp
- int yy_looking_for_trail_begin
- int yy_more_flag
- int yy_more_len
- int yy_more_offset
- int yy_prev_more_offset
- char * yytext
- int yyleng
- int yylineno
- int yy_flex_debug

10.295.1 Constructor & Destructor Documentation

10.295.1.1 yyFlexLexer::yyFlexLexer (FLEX_STD istream * *arg_yyin* = 0, FLEX_STD ostream * *arg_yyout* = 0)

Definition at line 893 of file dbcompare_expression_scanner.cc.

10.295.1.2 yyFlexLexer::~yyFlexLexer ( ) [virtual]

Definition at line 921 of file dbcompare_expression_scanner.cc.

10.295.2 Member Function Documentation

10.295.2.1 int FlexLexer::debug ( ) const [inline, inherited]

Definition at line 91 of file FlexLexer.h.

10.295.2.2 void yyFlexLexer::LexerError (const char * *msg*) [protected, virtual]

Definition at line 1567 of file dbcompare_expression_scanner.cc.

10.295.2.3 int yyFlexLexer::LexerInput (char * *buf*, int *max_size*) [protected, virtual]

Definition at line 943 of file dbcompare_expression_scanner.cc.

10.295.2.4 void yyFlexLexer::LexerOutput (const char * *buf*, int *size*) [protected, virtual]

Definition at line 970 of file dbcompare_expression_scanner.cc.

10.295.2.5 int FlexLexer::lineno ( ) const [inline, inherited]

Definition at line 89 of file FlexLexer.h.

10.295.2.6 void FlexLexer::set_debug (int *flag*) [inline, inherited]

Definition at line 92 of file FlexLexer.h.

10.295.2.7 void yyFlexLexer::switch_streams (FLEX_STD istream * *new_in*, FLEX_STD ostream * *new_out*) [virtual]

Implements FlexLexer.

Definition at line 928 of file dbcompare_expression_scanner.cc.

10.295.2.8 YY_BUFFER_STATE yyFlexLexer::yy_create_buffer (FLEX_STD istream * *s*, int *size*) [read, virtual]

Allocate and initialize an input buffer state.

Parameters

file	A readable stream.
size	The character buffer size in bytes. When in doubt, use YY_BUF_SIZE.

Returns

the allocated buffer state.

Implements FlexLexer.

Definition at line 1334 of file dbcompare_expression_scanner.cc.

10.295.2.9 void yyFlexLexer::yy_delete_buffer (struct yy_buffer_state * *b*) [virtual]

Destroy the buffer.

Parameters

b	a buffer created with yy_create_buffer()

Implements FlexLexer.

Definition at line 1362 of file dbcompare_expression_scanner.cc.

10.295.2.10 void yyFlexLexer::yy_flush_buffer (struct yy_buffer_state * *b*) [protected]

Discard all buffered characters. On the next scan, YY_INPUT will be called.

Parameters

b	the buffer state to be flushed, usually `YY_CURRENT_BUFFER`.

Definition at line 1408 of file dbcompare_expression_scanner.cc.

10.295.2.11 int yyFlexLexer::yy_get_next_buffer ( ) [protected]

Definition at line 982 of file dbcompare_expression_scanner.cc.

10.295.2.12 yy_state_type yyFlexLexer::yy_get_previous_state ( ) [protected]

Definition at line 1108 of file dbcompare_expression_scanner.cc.

10.295.2.13 void yyFlexLexer::yy_init_buffer (struct yy_buffer_state * *b*, FLEX_STD istream * *s*) [protected]

Definition at line 1381 of file dbcompare_expression_scanner.cc.

10.295.2.14 void yyFlexLexer::yy_load_buffer_state ( ) [protected]

Definition at line 1320 of file dbcompare_expression_scanner.cc.

10.295.2.15 void yyFlexLexer::yy_pop_state ( ) [protected]

Definition at line 1550 of file dbcompare_expression_scanner.cc.

10.295.2.16 void yyFlexLexer::yy_push_state (int *new_state*) [protected]

Definition at line 1525 of file dbcompare_expression_scanner.cc.

10.295.2.17 void yyFlexLexer::yy_switch_to_buffer (struct yy_buffer_state * *new_buffer*) [virtual]

Switch to a different input buffer.

Parameters

new_buffer	The new input buffer.

Implements FlexLexer.

Definition at line 1289 of file dbcompare_expression_scanner.cc.

10.295.2.18 int yyFlexLexer::yy_top_state ( ) [protected]

Definition at line 1558 of file dbcompare_expression_scanner.cc.

10.295.2.19 yy_state_type yyFlexLexer::yy_try_NUL_trans (yy_state_type *current_state*) [protected]

Definition at line 1140 of file dbcompare_expression_scanner.cc.

10.295.2.20 void yyFlexLexer::yyensure_buffer_stack (void) [protected]

Definition at line 1486 of file dbcompare_expression_scanner.cc.

10.295.2.21 int yyFlexLexer::yyinput ( ) [protected]

Definition at line 1200 of file dbcompare_expression_scanner.cc.

10.295.2.22 int FlexLexer::YYLeng ( ) [inline, inherited]

Definition at line 66 of file FlexLexer.h.

10.295.2.23 int FlexLexer::yylex (FLEX_STD istream * *new_in*, FLEX_STD ostream * *new_out* = 0) [inline, inherited]

Definition at line 78 of file FlexLexer.h.

10.295.2.24 int yyFlexLexer::yylex ( ) [virtual]

Implements FlexLexer.

Definition at line 319 of file dbcompare_expression_scanner.cc.

10.295.2.25 void yyFlexLexer::yypop_buffer_state (void)

Removes and deletes the top of the stack, if present. The next element becomes the new top.

Definition at line 1467 of file dbcompare_expression_scanner.cc.

10.295.2.26 void yyFlexLexer::yypush_buffer_state (struct yy_buffer_state * *new_buffer*)

Pushes the new state onto the stack. The new state becomes the current state. This function will allocate the stack if necessary.

Parameters

new_buffer	The new state.

Definition at line 1437 of file dbcompare_expression_scanner.cc.

10.295.2.27 void yyFlexLexer::yyrestart (FLEX_STD istream * *s*) [virtual]

Immediately switch to a different input stream.

Parameters

input_file	A readable stream.

Note

This function does not reset the start condition to `INITIAL` .

Implements FlexLexer.

Definition at line 1272 of file dbcompare_expression_scanner.cc.

10.295.2.28 const char* FlexLexer::YYText ( ) [inline, inherited]

Definition at line 65 of file FlexLexer.h.

10.295.2.29 void yyFlexLexer::yyunput (int *c*, char * *buf_ptr*) [protected]

Definition at line 1163 of file dbcompare_expression_scanner.cc.

10.295.3 Member Data Documentation

10.295.3.1 struct yy_buffer_state** yyFlexLexer::yy_buffer_stack [protected]

Stack as an array.

Definition at line 177 of file FlexLexer.h.

10.295.3.2 size_t yyFlexLexer::yy_buffer_stack_max [protected]

capacity of stack.

Definition at line 176 of file FlexLexer.h.

10.295.3.3 size_t yyFlexLexer::yy_buffer_stack_top [protected]

index of top of stack.

Definition at line 175 of file FlexLexer.h.

10.295.3.4 char* yyFlexLexer::yy_c_buf_p [protected]

Definition at line 165 of file FlexLexer.h.

10.295.3.5 int yyFlexLexer::yy_did_buffer_switch_on_eof [protected]

Definition at line 172 of file FlexLexer.h.

10.295.3.6 int FlexLexer::yy_flex_debug [protected, inherited]

Definition at line 98 of file FlexLexer.h.

10.295.3.7 int yyFlexLexer::yy_full_lp [protected]

Definition at line 191 of file FlexLexer.h.

10.295.3.8 char* yyFlexLexer::yy_full_match [protected]

Definition at line 189 of file FlexLexer.h.

10.295.3.9 int* yyFlexLexer::yy_full_state [protected]

Definition at line 190 of file FlexLexer.h.

10.295.3.10 char yyFlexLexer::yy_hold_char [protected]

Definition at line 159 of file FlexLexer.h.

10.295.3.11 int yyFlexLexer::yy_init [protected]

Definition at line 167 of file FlexLexer.h.

10.295.3.12 char* yyFlexLexer::yy_last_accepting_cpos [protected]

Definition at line 184 of file FlexLexer.h.

10.295.3.13 yy_state_type yyFlexLexer::yy_last_accepting_state [protected]

Definition at line 183 of file FlexLexer.h.

10.295.3.14 int yyFlexLexer::yy_looking_for_trail_begin [protected]

Definition at line 194 of file FlexLexer.h.

10.295.3.15 int yyFlexLexer::yy_lp [protected]

Definition at line 193 of file FlexLexer.h.

10.295.3.16 int yyFlexLexer::yy_more_flag [protected]

Definition at line 196 of file FlexLexer.h.

10.295.3.17 int yyFlexLexer::yy_more_len [protected]

Definition at line 197 of file FlexLexer.h.

10.295.3.18 int yyFlexLexer::yy_more_offset [protected]

Definition at line 198 of file FlexLexer.h.

10.295.3.19 int yyFlexLexer::yy_n_chars [protected]

Definition at line 162 of file FlexLexer.h.

10.295.3.20 int yyFlexLexer::yy_prev_more_offset [protected]

Definition at line 199 of file FlexLexer.h.

10.295.3.21 int yyFlexLexer::yy_start [protected]

Definition at line 168 of file FlexLexer.h.

10.295.3.22 int* yyFlexLexer::yy_start_stack [protected]

Definition at line 145 of file FlexLexer.h.

10.295.3.23 int yyFlexLexer::yy_start_stack_depth [protected]

Definition at line 144 of file FlexLexer.h.

10.295.3.24 int yyFlexLexer::yy_start_stack_ptr [protected]

Definition at line 143 of file FlexLexer.h.

10.295.3.25 yy_state_type* yyFlexLexer::yy_state_buf [protected]

Definition at line 186 of file FlexLexer.h.

10.295.3.26 yy_state_type* yyFlexLexer::yy_state_ptr [protected]

Definition at line 187 of file FlexLexer.h.

10.295.3.27 FLEX_STD istream* yyFlexLexer::yyin [protected]

Definition at line 155 of file FlexLexer.h.

10.295.3.28 int FlexLexer::yyleng [protected, inherited]

Definition at line 96 of file FlexLexer.h.

10.295.3.29 int FlexLexer::yylineno [protected, inherited]

Definition at line 97 of file FlexLexer.h.

10.295.3.30 FLEX_STD ostream* yyFlexLexer::yyout [protected]

Definition at line 156 of file FlexLexer.h.

10.295.3.31 char* FlexLexer::yytext [protected, inherited]

Definition at line 95 of file FlexLexer.h.

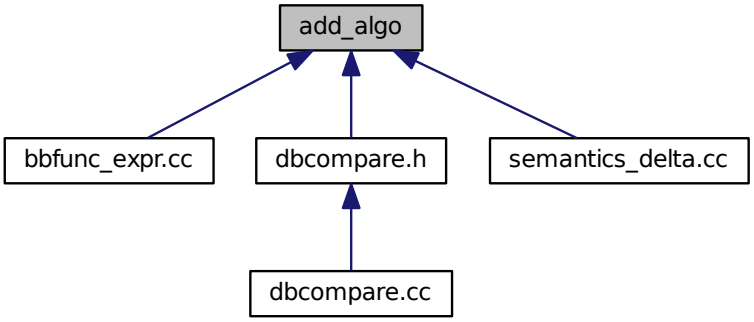
The documentation for this class was generated from the following files:

- FlexLexer.h
- dbcompare_expression_scanner.cc
- dbcompare_sortorder_scanner.cc
- Number_scanner.cc

11 File Documentation

11.1 add_algo File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define _CPP_ADD_ALGO 1

11.1.1 Detailed Description

This is a like a Standard C++ Library header. You should `#include` this header in your programs, rather than any of the "st[dl]_*.h" implementation files.

Definition in file add_algo.

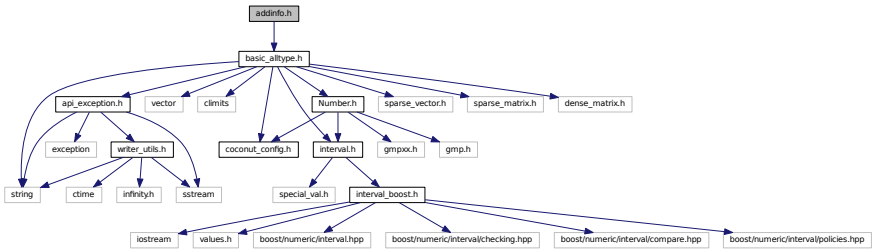
11.1.2 Define Documentation

11.1.2.1 #define _CPP_ADD_ALGO 1

Definition at line 38 of file add_algo.

11.2 addinfo.h File Reference

`#include <basic_alltype.h>` Include dependency graph for addinfo.h:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef basic_alltype coco::additional_info_u

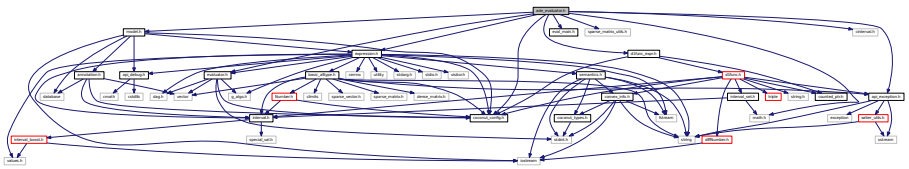
  type definition for backwards compatibility with earlier API versions

11.2.1 Detailed Description

Definition in file addinfo.h.

11.3 ade_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <sparse_matrix_-utils.h>` `#include <math.h>` `#include <cinterval.h>` `#include <d1func_expr.-h>` `#include <api_exception.h>` Include dependency graph for ade_evaluator.h:



Classes

- struct coco::analyticd_eval_type
- class coco::analyticd_eval

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

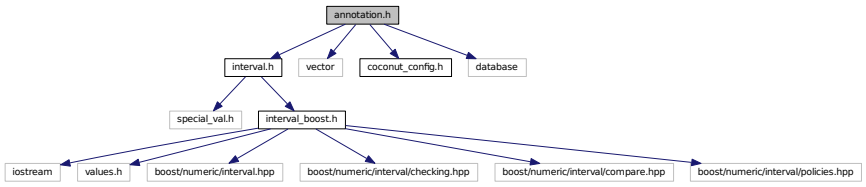
- typedef analyticd(∗ coco::analyticd_evaluator)(const std::vector< analyticd > ∗__x, const variable_indicator &__v)

11.3.1 Detailed Description

Definition in file ade_evaluator.h.

11.4 annotation.h File Reference

`#include <interval.h>` `#include <vector>` `#include <coconut_config.h>` `#include <database>` Include dependency graph for annotation.h:



This graph shows which files directly or indirectly include this file:

Classes

- class coco::annotation

  Annotations for Models.

Namespaces

- namespace coco

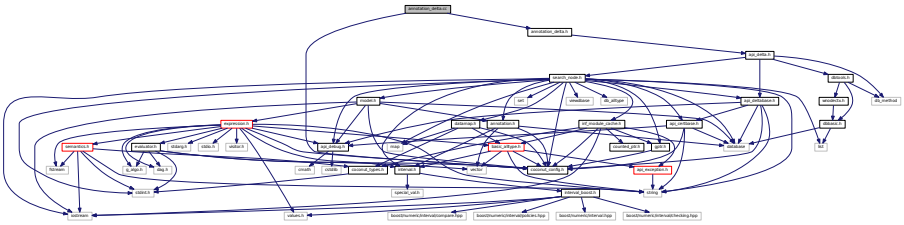
  the main namespace of the COCONUT API

11.4.1 Detailed Description

Definition in file annotation.h.

11.5 annotation_delta.cc File Reference

`#include <coconut_config.h>` `#include <annotation_delta.h>` Include dependency graph for annotation_delta.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

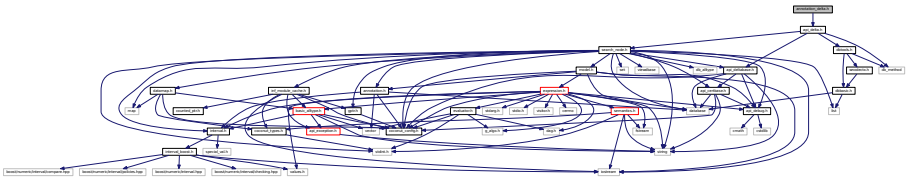
- typedef std::vector< std::pair < vdbl::tableid, vdbl::rowid > > coco::trvec

11.5.1 Detailed Description

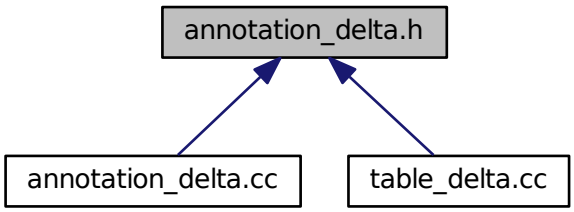
Definition in file annotation_delta.cc.

11.6 annotation_delta.h File Reference

`#include <api_delta.h>` Include dependency graph for annotation_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::annotation_undelta

  the undelta class for annotation changes
- class coco::annotation_delta

  the delta class for annotation changes

Namespaces

- namespace coco

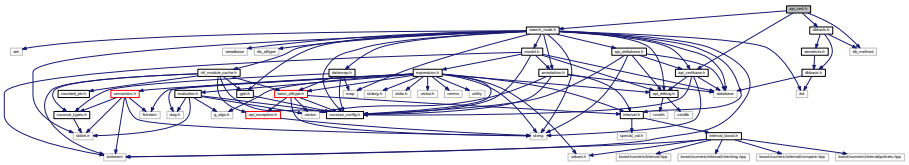
  the main namespace of the COCONUT API

11.6.1 Detailed Description

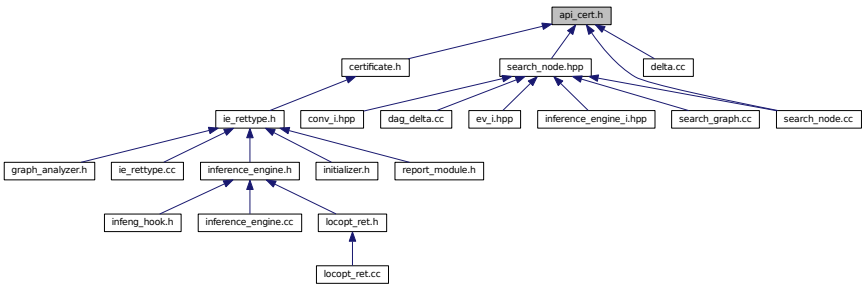
Definition in file annotation_delta.h.

11.7 api_cert.h File Reference

`#include <api_certbase.h> #include <search_node.h> #include <dbtools.-h> #include <db_method>` Include dependency graph for api_cert.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::no_certificate

  The not-certified certificate.
- class coco::split_certificate

  The certificate for deltas formed by splits.
- class coco::compound_certificate

  The certificate for deltas formed by compressing bound_delta entries.
- class coco::rigorous_module_certificate

  The certificate for deltas computed by rigorous inference engines.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

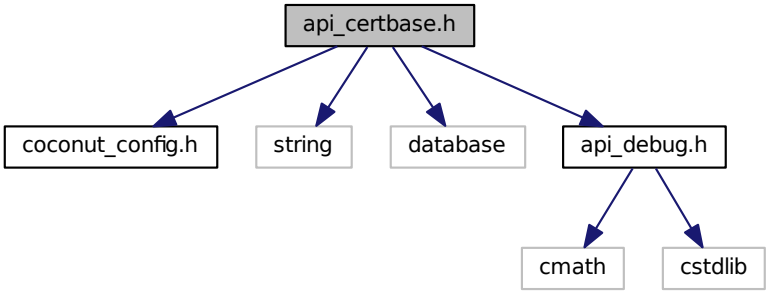
11.7.1 Detailed Description

This is an internal header file, which is not intended for use outside the API.

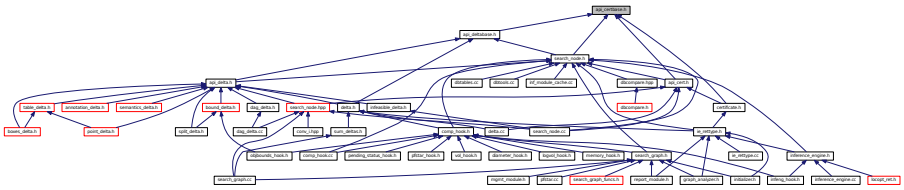
Definition in file api_cert.h.

11.8 api_certbase.h File Reference

`#include <coconut_config.h>` `#include <string>` `#include <database>` `#include <api_debug.h>` Include dependency graph for api_certbase.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::certificate

  The certificate class (certifies deltas for rigorous mode operation)
- class coco::certificate_base

  Base class for the certificates.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- std::ostream & coco::operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.

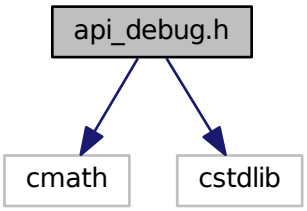
11.8.1 Detailed Description

This is an internal header file not intended for use outside the API

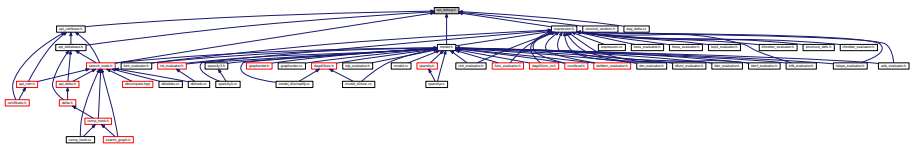
Definition in file api_certbase.h.

11.9 api_debug.h File Reference

`#include <cmath>` `#include <cstdlib>` Include dependency graph for api_debug.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco
  the main namespace of the COCONUT API

Defines

- #define DEBUG_CERTIFICATE 0
- #define DEBUG_DELTA 0
- #define __DEBUG_DAG_DELTA 0
- #define DEBUG_RANDOM 0
- #define DEBUG_EXPRESSIONS 0
- #define DEBUG_SEARCH_NODE 0
- #define MODEL_INLINE_DEBUG 0
- #define MODEL_INLINE_DEBUG_SIMPLIFY 0

11.9.1 Detailed Description

Definition in file api_debug.h.

11.9.2 Define Documentation

11.9.2.1 #define __DEBUG_DAG_DELTA 0

Definition at line 45 of file api_debug.h.

11.9.2.2 #define DEBUG_CERTIFICATE 0

Definition at line 37 of file api_debug.h.

11.9.2.3 #define DEBUG_DELTA 0

Definition at line 41 of file api_debug.h.

11.9.2.4 #define DEBUG_EXPRESSIONS 0

Definition at line 53 of file api_debug.h.

11.9.2.5 #define DEBUG_RANDOM 0

Definition at line 49 of file api_debug.h.

11.9.2.6 #define DEBUG_SEARCH_NODE 0

Definition at line 57 of file api_debug.h.

11.9.2.7 #define MODEL_INLINE_DEBUG 0

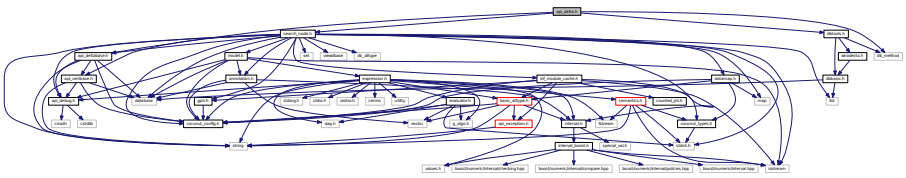
Definition at line 61 of file api_debug.h.

11.9.2.8 #define MODEL_INLINE_DEBUG_SIMPLIFY 0

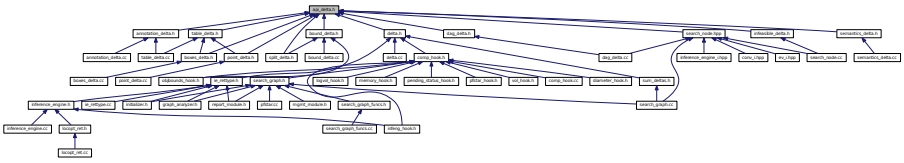
Definition at line 65 of file api_debug.h.

11.10 api_delta.h File Reference

`#include <api_deltabase.h>` `#include <search_node.h>` `#include <dbtools.-h>` `#include <db_method>` Include dependency graph for api_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::delta_get_action

  Stored procedure class for computing the delta action specifier.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

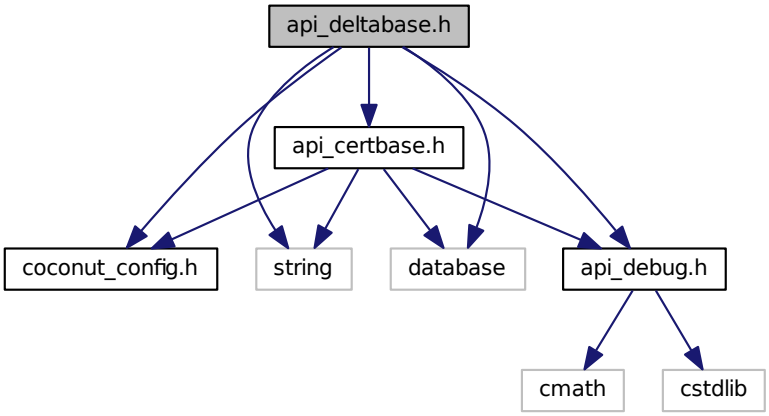
11.10.1 Detailed Description

This is an internal header file, which is not intended for use outside the API.

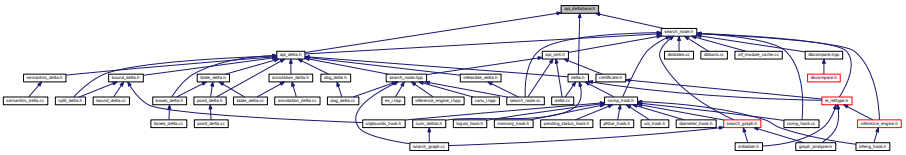
Definition in file api_delta.h.

11.11 api_deltabase.h File Reference

`#include <coconut_config.h>` `#include <string>` `#include <database>` `#include <api_certbase.h>` `#include <api_debug.h>` Include dependency graph for api_deltabase.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::delta

  The delta class (updates to work nodes)
- class coco::delta_base

  Base class for the deltas.
- class coco::undelta

  The undelta class (undo of updates to work nodes)
- class coco::undelta_base

  Base class for the undeltas.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef vdbl::rowid coco::delta_id

  The class for delta ids.

Functions

- std::ostream & coco::operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.

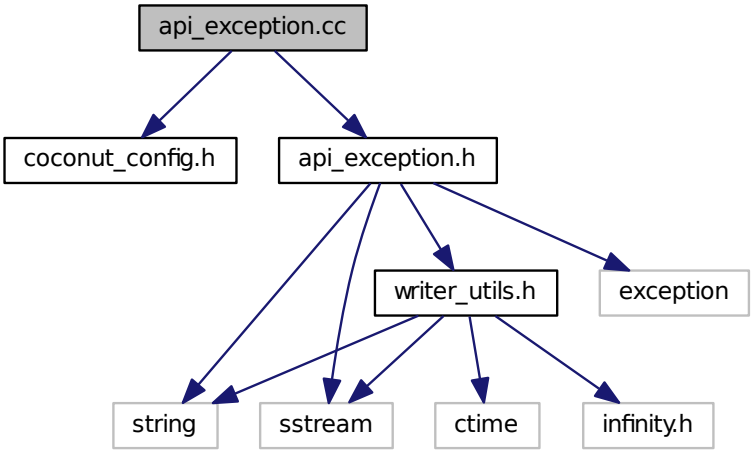
11.11.1 Detailed Description

This is an internal header file not intended for use outside the API

Definition in file api_deltabase.h.

11.12 api_exception.cc File Reference

`#include <coconut_config.h>` `#include <api_exception.h>` Include dependency graph for api_exception.cc:



Namespaces

- namespace coco

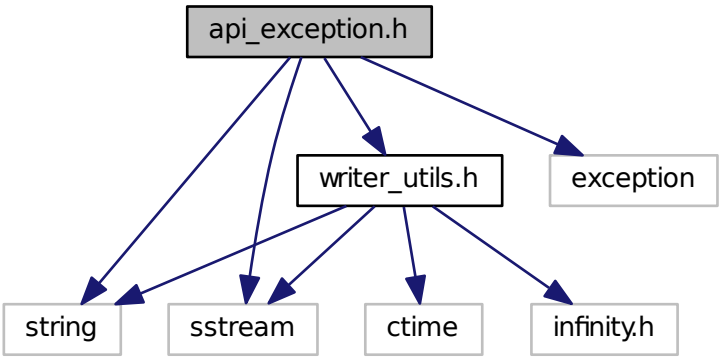
  the main namespace of the COCONUT API

11.12.1 Detailed Description

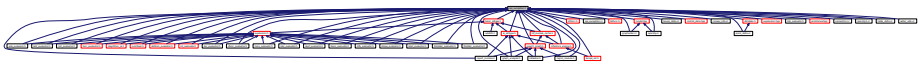
Definition in file api_exception.cc.

11.13 api_exception.h File Reference

`#include <string>` `#include <exception>` `#include <sstream>` `#include <writer_utils.h>` Include dependency graph for api_exception.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::api_exception

  API exception class.
- class coco::nyi_exception

  Not Yet Implemented exception class.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Enumerations

- enum coco::api_exception_type { coco::apiee_internal = 1, coco::apiee_evaluator = 2, coco::apiee_io = 3, coco::apiee_delta = 4, coco::apiee_search_graph = 5, coco::apiee_communication_data = 6, coco::apiee_inference_engine = 7, coco::apiee_graph_analyzer = 8, coco::apiee_management_module = 9, coco::apiee_initializer = 10, coco::apiee_report_module = 11, coco::apiee_oom = 12, coco::apiee_nyi = 13, coco::apiee_other = 14 }

  Enum for classifying api_exceptions.

11.13.1 Detailed Description

Definition in file api_exception.h.

11.14 api_extradocu.h File Reference

Namespaces

- namespace coco

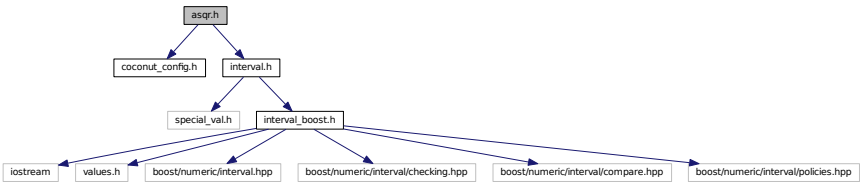
  the main namespace of the COCONUT API

11.14.1 Detailed Description

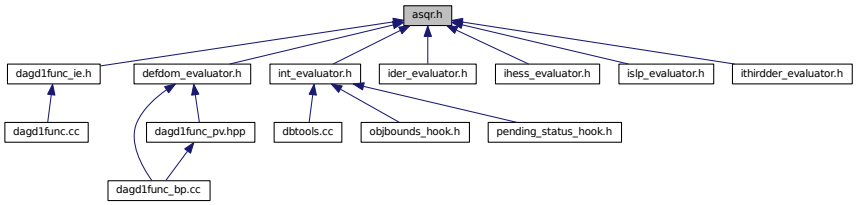
Definition in file api_extradocu.h.

11.15 asqr.h File Reference

`#include <coconut_config.h>` `#include <interval.h>` Include dependency graph for asqr.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

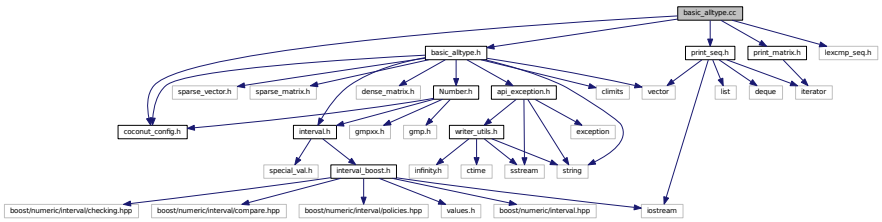
- interval coco::flb (const interval &x1, const interval &x2, const interval &x3)
- interval coco::asqr_find_f1min (const interval &c1, const interval &c3, const interval &x2)
- interval coco::eval_asqr (const interval &x1, const interval &x2, const interval &x3)

11.15.1 Detailed Description

Definition in file asqr.h.

11.16 basic_alltype.cc File Reference

`#include <coconut_config.h>` `#include <basic_alltype.h>` `#include <print-_seq.h>` `#include <print_matrix.h>` `#include <lexcmp_seq.h>` Include dependency graph for basic_alltype.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

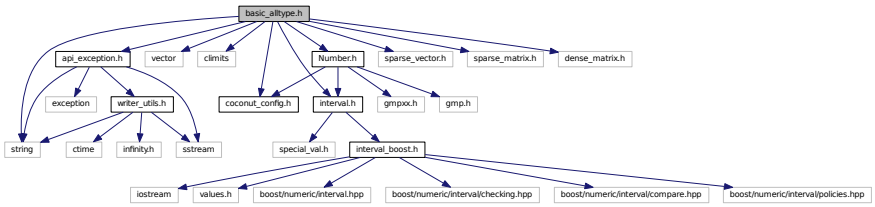
- const basic_alltype & coco::basic_alltype_empty ()
- std::ostream & coco::operator<< (std::ostream &os, const basic_alltype &b)
- bool coco::less_than (const basic_alltype &a, const basic_alltype &b)
- bool coco::less_equal (const basic_alltype &a, const basic_alltype &b)

11.16.1 Detailed Description

Definition in file basic_alltype.cc.

11.17 basic_alltype.h File Reference

`#include <string>` `#include <vector>` `#include <climits>` `#include <coconut-_config.h>` `#include <interval.h>` `#include <sparse_vector.h>` `#include <sparse-_matrix.h>` `#include <dense_matrix.h>` `#include <Number.h>` `#include <api-_exception.h>` Include dependency graph for basic_alltype.h:



This graph shows which files directly or indirectly include this file:

Classes

- class coco::basic_alltype

  The basic alltype which can hold any of a number of basic types.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define BASIC_ALLTYPE_CHECK 1
- #define ALLTYPE_INTERVAL -6
- #define ALLTYPE_VOID_PTR -5
- #define ALLTYPE_DOUBLE -4
- #define ALLTYPE_UINT -3
- #define ALLTYPE_INT -2
- #define ALLTYPE_BOOL -1
- #define ALLTYPE_EMPTY 0
- #define ALLTYPE_ALLOCED_B 1
- #define ALLTYPE_ALLOCED_N 2
- #define ALLTYPE_ALLOCED_U 3
- #define ALLTYPE_ALLOCED_D 4
- #define ALLTYPE_ALLOCED_I 5
- #define ALLTYPE_ALLOCED_C 6
- #define ALLTYPE_ALLOCED_X 7
- #define ALLTYPE_ALLOCED_VB 8
- #define ALLTYPE_ALLOCED_VN 9
- #define ALLTYPE_ALLOCED_VU 10
- #define ALLTYPE_ALLOCED_VD 11
- #define ALLTYPE_ALLOCED_VI 12
- #define ALLTYPE_ALLOCED_VC 13
- #define ALLTYPE_ALLOCED_VX 14
- #define ALLTYPE_ALLOCED_SB 15
- #define ALLTYPE_ALLOCED_SN 16
- #define ALLTYPE_ALLOCED_SU 17
- #define ALLTYPE_ALLOCED_SD 18
- #define ALLTYPE_ALLOCED_SI 19
- #define ALLTYPE_ALLOCED_SC 20
- #define ALLTYPE_ALLOCED_SX 21
- #define ALLTYPE_ALLOCED_S 22
- #define ALLTYPE_ALLOCED_Y 23
- #define ALLTYPE_ALLOCED_DM 24
- #define ALLTYPE_ALLOCED_NDM 25
- #define ALLTYPE_ALLOCED_IDM 26
- #define ALLTYPE_ALLOCED_CDM 27
- #define ALLTYPE_ALLOCED_XDM 28
- #define ALLTYPE_ALLOCED_SM 29
- #define ALLTYPE_ALLOCED_NSM 30
- #define ALLTYPE_ALLOCED_ISM 31
- #define ALLTYPE_ALLOCED_CSM 32
- #define ALLTYPE_ALLOCED_XSM 33
- #define ALLTYPE_FIRST_UNDEF 34
- #define ALLTYPE_ANY 99

- #define BASIC_ALLTYPE_NAMES
- #define BASIC_ALLTYPE_ASSERT(A)
- #define BASIC_ALLTYPE_ASSERT2(A, B)
- #define BASIC_ALLTYPE_THROW throw(api_exception)
- #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)
- #define BASIC_ALLTYPE_CHECK 1
- #define ALLTYPE_INTERVAL -6
- #define ALLTYPE_VOID_PTR -5
- #define ALLTYPE_DOUBLE -4
- #define ALLTYPE_UINT -3
- #define ALLTYPE_INT -2
- #define ALLTYPE_BOOL -1
- #define ALLTYPE_EMPTY 0
- #define ALLTYPE_ALLOCED_B 1
- #define ALLTYPE_ALLOCED_N 2
- #define ALLTYPE_ALLOCED_U 3
- #define ALLTYPE_ALLOCED_D 4
- #define ALLTYPE_ALLOCED_I 5
- #define ALLTYPE_ALLOCED_C 6
- #define ALLTYPE_ALLOCED_X 7
- #define ALLTYPE_ALLOCED_VB 8
- #define ALLTYPE_ALLOCED_VN 9
- #define ALLTYPE_ALLOCED_VU 10
- #define ALLTYPE_ALLOCED_VD 11
- #define ALLTYPE_ALLOCED_VI 12
- #define ALLTYPE_ALLOCED_VC 13
- #define ALLTYPE_ALLOCED_VX 14
- #define ALLTYPE_ALLOCED_SB 15
- #define ALLTYPE_ALLOCED_SN 16
- #define ALLTYPE_ALLOCED_SU 17
- #define ALLTYPE_ALLOCED_SD 18
- #define ALLTYPE_ALLOCED_SI 19
- #define ALLTYPE_ALLOCED_SC 20
- #define ALLTYPE_ALLOCED_SX 21
- #define ALLTYPE_ALLOCED_S 22
- #define ALLTYPE_ALLOCED_Y 23
- #define ALLTYPE_ALLOCED_DM 24
- #define ALLTYPE_ALLOCED_NDM 25
- #define ALLTYPE_ALLOCED_IDM 26
- #define ALLTYPE_ALLOCED_CDM 27
- #define ALLTYPE_ALLOCED_XDM 28
- #define ALLTYPE_ALLOCED_SM 29
- #define ALLTYPE_ALLOCED_NSM 30
- #define ALLTYPE_ALLOCED_ISM 31
- #define ALLTYPE_ALLOCED_CSM 32
- #define ALLTYPE_ALLOCED_XSM 33
- #define ALLTYPE_FIRST_UNDEF 34
- #define ALLTYPE_ANY 99
- #define BASIC_ALLTYPE_NAMES
- #define BASIC_ALLTYPE_ASSERT(A)

- #define BASIC_ALLTYPE_ASSERT2(A, B)
- #define BASIC_ALLTYPE_THROW throw(api_exception)
- #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)
- #define BASIC_ALLTYPE_CHECK 1
- #define ALLTYPE_INTERVAL -6
- #define ALLTYPE_VOID_PTR -5
- #define ALLTYPE_DOUBLE -4
- #define ALLTYPE_UINT -3
- #define ALLTYPE_INT -2
- #define ALLTYPE_BOOL -1
- #define ALLTYPE_EMPTY 0
- #define ALLTYPE_ALLOCED_B 1
- #define ALLTYPE_ALLOCED_N 2
- #define ALLTYPE_ALLOCED_U 3
- #define ALLTYPE_ALLOCED_D 4
- #define ALLTYPE_ALLOCED_I 5
- #define ALLTYPE_ALLOCED_C 6
- #define ALLTYPE_ALLOCED_X 7
- #define ALLTYPE_ALLOCED_VB 8
- #define ALLTYPE_ALLOCED_VN 9
- #define ALLTYPE_ALLOCED_VU 10
- #define ALLTYPE_ALLOCED_VD 11
- #define ALLTYPE_ALLOCED_VI 12
- #define ALLTYPE_ALLOCED_VC 13
- #define ALLTYPE_ALLOCED_VX 14
- #define ALLTYPE_ALLOCED_SB 15
- #define ALLTYPE_ALLOCED_SN 16
- #define ALLTYPE_ALLOCED_SU 17
- #define ALLTYPE_ALLOCED_SD 18
- #define ALLTYPE_ALLOCED_SI 19
- #define ALLTYPE_ALLOCED_SC 20
- #define ALLTYPE_ALLOCED_SX 21
- #define ALLTYPE_ALLOCED_S 22
- #define ALLTYPE_ALLOCED_Y 23
- #define ALLTYPE_ALLOCED_DM 24
- #define ALLTYPE_ALLOCED_NDM 25
- #define ALLTYPE_ALLOCED_IDM 26
- #define ALLTYPE_ALLOCED_CDM 27
- #define ALLTYPE_ALLOCED_XDM 28
- #define ALLTYPE_ALLOCED_SM 29
- #define ALLTYPE_ALLOCED_NSM 30
- #define ALLTYPE_ALLOCED_ISM 31
- #define ALLTYPE_ALLOCED_CSM 32
- #define ALLTYPE_ALLOCED_XSM 33
- #define ALLTYPE_FIRST_UNDEF 34
- #define ALLTYPE_ANY 99
- #define BASIC_ALLTYPE_NAMES
- #define BASIC_ALLTYPE_ASSERT(A)
- #define BASIC_ALLTYPE_ASSERT2(A, B)
- #define BASIC_ALLTYPE_THROW throw(api_exception)

- #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)
- #define BASIC_ALLTYPE_CHECK 1
- #define ALLTYPE_INTERVAL -6
- #define ALLTYPE_VOID_PTR -5
- #define ALLTYPE_DOUBLE -4
- #define ALLTYPE_UINT -3
- #define ALLTYPE_INT -2
- #define ALLTYPE_BOOL -1
- #define ALLTYPE_EMPTY 0
- #define ALLTYPE_ALLOCED_B 1
- #define ALLTYPE_ALLOCED_N 2
- #define ALLTYPE_ALLOCED_U 3
- #define ALLTYPE_ALLOCED_D 4
- #define ALLTYPE_ALLOCED_I 5
- #define ALLTYPE_ALLOCED_C 6
- #define ALLTYPE_ALLOCED_X 7
- #define ALLTYPE_ALLOCED_VB 8
- #define ALLTYPE_ALLOCED_VN 9
- #define ALLTYPE_ALLOCED_VU 10
- #define ALLTYPE_ALLOCED_VD 11
- #define ALLTYPE_ALLOCED_VI 12
- #define ALLTYPE_ALLOCED_VC 13
- #define ALLTYPE_ALLOCED_VX 14
- #define ALLTYPE_ALLOCED_SB 15
- #define ALLTYPE_ALLOCED_SN 16
- #define ALLTYPE_ALLOCED_SU 17
- #define ALLTYPE_ALLOCED_SD 18
- #define ALLTYPE_ALLOCED_SI 19
- #define ALLTYPE_ALLOCED_SC 20
- #define ALLTYPE_ALLOCED_SX 21
- #define ALLTYPE_ALLOCED_S 22
- #define ALLTYPE_ALLOCED_Y 23
- #define ALLTYPE_ALLOCED_DM 24
- #define ALLTYPE_ALLOCED_NDM 25
- #define ALLTYPE_ALLOCED_IDM 26
- #define ALLTYPE_ALLOCED_CDM 27
- #define ALLTYPE_ALLOCED_XDM 28
- #define ALLTYPE_ALLOCED_SM 29
- #define ALLTYPE_ALLOCED_NSM 30
- #define ALLTYPE_ALLOCED_ISM 31
- #define ALLTYPE_ALLOCED_CSM 32
- #define ALLTYPE_ALLOCED_XSM 33
- #define ALLTYPE_FIRST_UNDEF 34
- #define ALLTYPE_ANY 99
- #define BASIC_ALLTYPE_NAMES
- #define BASIC_ALLTYPE_ASSERT(A)
- #define BASIC_ALLTYPE_ASSERT2(A, B)
- #define BASIC_ALLTYPE_THROW throw(api_exception)
- #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)
- #define BASIC_ALLTYPE_CHECK 1

- #define ALLTYPE_INTERVAL -6
- #define ALLTYPE_VOID_PTR -5
- #define ALLTYPE_DOUBLE -4
- #define ALLTYPE_UINT -3
- #define ALLTYPE_INT -2
- #define ALLTYPE_BOOL -1
- #define ALLTYPE_EMPTY 0
- #define ALLTYPE_ALLOCED_B 1
- #define ALLTYPE_ALLOCED_N 2
- #define ALLTYPE_ALLOCED_U 3
- #define ALLTYPE_ALLOCED_D 4
- #define ALLTYPE_ALLOCED_I 5
- #define ALLTYPE_ALLOCED_C 6
- #define ALLTYPE_ALLOCED_X 7
- #define ALLTYPE_ALLOCED_VB 8
- #define ALLTYPE_ALLOCED_VN 9
- #define ALLTYPE_ALLOCED_VU 10
- #define ALLTYPE_ALLOCED_VD 11
- #define ALLTYPE_ALLOCED_VI 12
- #define ALLTYPE_ALLOCED_VC 13
- #define ALLTYPE_ALLOCED_VX 14
- #define ALLTYPE_ALLOCED_SB 15
- #define ALLTYPE_ALLOCED_SN 16
- #define ALLTYPE_ALLOCED_SU 17
- #define ALLTYPE_ALLOCED_SD 18
- #define ALLTYPE_ALLOCED_SI 19
- #define ALLTYPE_ALLOCED_SC 20
- #define ALLTYPE_ALLOCED_SX 21
- #define ALLTYPE_ALLOCED_S 22
- #define ALLTYPE_ALLOCED_Y 23
- #define ALLTYPE_ALLOCED_DM 24
- #define ALLTYPE_ALLOCED_NDM 25
- #define ALLTYPE_ALLOCED_IDM 26
- #define ALLTYPE_ALLOCED_CDM 27
- #define ALLTYPE_ALLOCED_XDM 28
- #define ALLTYPE_ALLOCED_SM 29
- #define ALLTYPE_ALLOCED_NSM 30
- #define ALLTYPE_ALLOCED_ISM 31
- #define ALLTYPE_ALLOCED_CSM 32
- #define ALLTYPE_ALLOCED_XSM 33
- #define ALLTYPE_FIRST_UNDEF 34
- #define ALLTYPE_ANY 99
- #define BASIC_ALLTYPE_NAMES
- #define BASIC_ALLTYPE_ASSERT(A)
- #define BASIC_ALLTYPE_ASSERT2(A, B)
- #define BASIC_ALLTYPE_THROW throw(api_exception)
- #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)
- #define BASIC_ALLTYPE_CHECK 1
- #define ALLTYPE_INTERVAL -6
- #define ALLTYPE_VOID_PTR -5

- #define ALLTYPE_DOUBLE -4
- #define ALLTYPE_UINT -3
- #define ALLTYPE_INT -2
- #define ALLTYPE_BOOL -1
- #define ALLTYPE_EMPTY 0
- #define ALLTYPE_ALLOCED_B 1
- #define ALLTYPE_ALLOCED_N 2
- #define ALLTYPE_ALLOCED_U 3
- #define ALLTYPE_ALLOCED_D 4
- #define ALLTYPE_ALLOCED_I 5
- #define ALLTYPE_ALLOCED_C 6
- #define ALLTYPE_ALLOCED_X 7
- #define ALLTYPE_ALLOCED_VB 8
- #define ALLTYPE_ALLOCED_VN 9
- #define ALLTYPE_ALLOCED_VU 10
- #define ALLTYPE_ALLOCED_VD 11
- #define ALLTYPE_ALLOCED_VI 12
- #define ALLTYPE_ALLOCED_VC 13
- #define ALLTYPE_ALLOCED_VX 14
- #define ALLTYPE_ALLOCED_SB 15
- #define ALLTYPE_ALLOCED_SN 16
- #define ALLTYPE_ALLOCED_SU 17
- #define ALLTYPE_ALLOCED_SD 18
- #define ALLTYPE_ALLOCED_SI 19
- #define ALLTYPE_ALLOCED_SC 20
- #define ALLTYPE_ALLOCED_SX 21
- #define ALLTYPE_ALLOCED_S 22
- #define ALLTYPE_ALLOCED_Y 23
- #define ALLTYPE_ALLOCED_DM 24
- #define ALLTYPE_ALLOCED_NDM 25
- #define ALLTYPE_ALLOCED_IDM 26
- #define ALLTYPE_ALLOCED_CDM 27
- #define ALLTYPE_ALLOCED_XDM 28
- #define ALLTYPE_ALLOCED_SM 29
- #define ALLTYPE_ALLOCED_NSM 30
- #define ALLTYPE_ALLOCED_ISM 31
- #define ALLTYPE_ALLOCED_CSM 32
- #define ALLTYPE_ALLOCED_XSM 33
- #define ALLTYPE_FIRST_UNDEF 34
- #define ALLTYPE_ANY 99
- #define BASIC_ALLTYPE_NAMES
- #define BASIC_ALLTYPE_ASSERT(A)
- #define BASIC_ALLTYPE_ASSERT2(A, B)
- #define BASIC_ALLTYPE_THROW throw(api_exception)
- #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)

Functions

- const basic_alltype & coco::basic_alltype_empty ()
- std::ostream & coco::operator<< (std::ostream &os, const basic_alltype &b)
- bool coco::less_than (const basic_alltype &a, const basic_alltype &b)
- bool coco::less_equal (const basic_alltype &a, const basic_alltype &b)

11.17.1 Detailed Description

Definition in file basic_alltype.h.

11.17.2 Define Documentation

11.17.2.1 #define ALLTYPE_ALLOCED_B 1

a vector<bool> is stored

Definition at line 63 of file search_graph.cc.

11.17.2.2 #define ALLTYPE_ALLOCED_B 1

11.17.2.3 #define ALLTYPE_ALLOCED_B 1

11.17.2.4 #define ALLTYPE_ALLOCED_B 1

11.17.2.5 #define ALLTYPE_ALLOCED_B 1

11.17.2.6 #define ALLTYPE_ALLOCED_B 1

11.17.2.7 #define ALLTYPE_ALLOCED_C 6

11.17.2.8 #define ALLTYPE_ALLOCED_C 6

11.17.2.9 #define ALLTYPE_ALLOCED_C 6

11.17.2.10 #define ALLTYPE_ALLOCED_C 6

a vector<string> is stored

Definition at line 68 of file search_graph.cc.

11.17.2.11 #define ALLTYPE_ALLOCED_C 6

11.17.2.12 #define ALLTYPE_ALLOCED_C 6

11.17.2.13 #define ALLTYPE_ALLOCED_CDM 27

11.17.2.14 #define ALLTYPE_ALLOCED_CDM 27

a dense_matrix<string> is stored

Definition at line 93 of file search_graph.cc.

11.17.2.15 #define ALLTYPE_ALLOCED_CDM 27

11.17.2.16 #define ALLTYPE_ALLOCED_CDM 27

11.17.2.17 #define ALLTYPE_ALLOCED_CDM 27

11.17.2.18 #define ALLTYPE_ALLOCED_CDM 27

11.17.2.19 #define ALLTYPE_ALLOCED_CSM 32

11.17.2.20 #define ALLTYPE_ALLOCED_CSM 32

11.17.2.21 #define ALLTYPE_ALLOCED_CSM 32

a sparse_matrix<string> is stored

Definition at line 99 of file search_graph.cc.

11.17.2.22 #define ALLTYPE_ALLOCED_CSM 32

11.17.2.23 #define ALLTYPE_ALLOCED_CSM 32

11.17.2.24 #define ALLTYPE_ALLOCED_CSM 32

11.17.2.25 #define ALLTYPE_ALLOCED_D 4

11.17.2.26 #define ALLTYPE_ALLOCED_D 4

11.17.2.27 #define ALLTYPE_ALLOCED_D 4

11.17.2.28 #define ALLTYPE_ALLOCED_D 4

a vector<double> is stored

Definition at line 66 of file search_graph.cc.

11.17.2.29 #define ALLTYPE_ALLOCED_D 4

11.17.2.30 #define ALLTYPE_ALLOCED_D 4

11.17.2.31 #define ALLTYPE_ALLOCED_DM 24

11.17.2.32 #define ALLTYPE_ALLOCED_DM 24

a dense_matrix<double> is stored

Definition at line 90 of file search_graph.cc.

11.17.2.33 #define ALLTYPE_ALLOCED_DM 24

11.17.2.34 #define ALLTYPE_ALLOCED_DM 24

11.17.2.35 #define ALLTYPE_ALLOCED_DM 24

11.17.2.36 #define ALLTYPE_ALLOCED_DM 24

11.17.2.37 #define ALLTYPE_ALLOCED_I 5

11.17.2.38 #define ALLTYPE_ALLOCED_I 5

11.17.2.39 #define ALLTYPE_ALLOCED_I 5

11.17.2.40 #define ALLTYPE_ALLOCED_I 5

a vector<interval> is stored

Definition at line 67 of file search_graph.cc.

11.17.2.41 #define ALLTYPE_ALLOCED_I 5

11.17.2.42 #define ALLTYPE_ALLOCED_I 5

11.17.2.43 #define ALLTYPE_ALLOCED_IDM 26

a dense_matrix<interval> is stored

Definition at line 92 of file search_graph.cc.

11.17.2.44 #define ALLTYPE_ALLOCED_IDM 26

11.17.2.45 #define ALLTYPE_ALLOCED_IDM 26

11.17.2.46 #define ALLTYPE_ALLOCED_IDM 26

11.17.2.47 #define ALLTYPE_ALLOCED_IDM 26

11.17.2.48 #define ALLTYPE_ALLOCED_IDM 26

11.17.2.49 #define ALLTYPE_ALLOCED_ISM 31

11.17.2.50 #define ALLTYPE_ALLOCED_ISM 31

11.17.2.51 #define ALLTYPE_ALLOCED_ISM 31

a sparse_matrix<interval> is stored

Definition at line 98 of file search_graph.cc.

11.17.2.52 #define ALLTYPE_ALLOCED_ISM 31

11.17.2.53 #define ALLTYPE_ALLOCED_ISM 31

11.17.2.54 #define ALLTYPE_ALLOCED_ISM 31

11.17.2.55 #define ALLTYPE_ALLOCED_N 2

a vector<int> is stored

Definition at line 64 of file search_graph.cc.

11.17.2.56 #define ALLTYPE_ALLOCED_N 2

11.17.2.57 #define ALLTYPE_ALLOCED_N 2

11.17.2.58 #define ALLTYPE_ALLOCED_N 2

11.17.2.59 #define ALLTYPE_ALLOCED_N 2

11.17.2.60 #define ALLTYPE_ALLOCED_N 2

11.17.2.61 #define ALLTYPE_ALLOCED_NDM 25

11.17.2.62 #define ALLTYPE_ALLOCED_NDM 25

11.17.2.63 #define ALLTYPE_ALLOCED_NDM 25

a dense_matrix<int> is stored

Definition at line 91 of file search_graph.cc.

11.17.2.64 #define ALLTYPE_ALLOCED_NDM 25

11.17.2.65 #define ALLTYPE_ALLOCED_NDM 25

11.17.2.66 #define ALLTYPE_ALLOCED_NDM 25

11.17.2.67 #define ALLTYPE_ALLOCED_NSM 30

11.17.2.68 #define ALLTYPE_ALLOCED_NSM 30

11.17.2.69 #define ALLTYPE_ALLOCED_NSM 30

a sparse_matrix<int> is stored

Definition at line 97 of file search_graph.cc.

11.17.2.70 #define ALLTYPE_ALLOCED_NSM 30

11.17.2.71 #define ALLTYPE_ALLOCED_NSM 30

11.17.2.72 #define ALLTYPE_ALLOCED_NSM 30

11.17.2.73 #define ALLTYPE_ALLOCED_S 22

11.17.2.74 #define ALLTYPE_ALLOCED_S 22

11.17.2.75 #define ALLTYPE_ALLOCED_S 22

11.17.2.76 #define ALLTYPE_ALLOCED_S 22

11.17.2.77 #define ALLTYPE_ALLOCED_S 22

a string is stored

Definition at line 87 of file search_graph.cc.

11.17.2.78 #define ALLTYPE_ALLOCED_S 22

11.17.2.79 #define ALLTYPE_ALLOCED_SB 15

11.17.2.80 #define ALLTYPE_ALLOCED_SB 15

11.17.2.81 #define ALLTYPE_ALLOCED_SB 15

a sparse_vector<bool> is stored

Definition at line 79 of file search_graph.cc.

11.17.2.82 #define ALLTYPE_ALLOCED_SB 15

11.17.2.83 #define ALLTYPE_ALLOCED_SB 15

11.17.2.84 #define ALLTYPE_ALLOCED_SB 15

11.17.2.85 #define ALLTYPE_ALLOCED_SC 20

11.17.2.86 #define ALLTYPE_ALLOCED_SC 20

11.17.2.87 #define ALLTYPE_ALLOCED_SC 20

11.17.2.88 #define ALLTYPE_ALLOCED_SC 20

11.17.2.89 #define ALLTYPE_ALLOCED_SC 20

a sparse_vector<string> is stored

Definition at line 84 of file search_graph.cc.

11.17.2.90 #define ALLTYPE_ALLOCED_SC 20

11.17.2.91 #define ALLTYPE_ALLOCED_SD 18

11.17.2.92 #define ALLTYPE_ALLOCED_SD 18

11.17.2.93 #define ALLTYPE_ALLOCED_SD 18

11.17.2.94 #define ALLTYPE_ALLOCED_SD 18

a sparse_vector<double> is stored

Definition at line 82 of file search_graph.cc.

11.17.2.95 #define ALLTYPE_ALLOCED_SD 18

11.17.2.96 #define ALLTYPE_ALLOCED_SD 18

11.17.2.97 #define ALLTYPE_ALLOCED_SI 19

11.17.2.98 #define ALLTYPE_ALLOCED_SI 19

11.17.2.99 #define ALLTYPE_ALLOCED_SI 19

11.17.2.100 #define ALLTYPE_ALLOCED_SI 19

11.17.2.101 #define ALLTYPE_ALLOCED_SI 19

a sparse_vector<interval> is stored

Definition at line 83 of file search_graph.cc.

11.17.2.102 #define ALLTYPE_ALLOCED_SI 19

11.17.2.103 #define ALLTYPE_ALLOCED_SM 29

a sparse_matrix<double> is stored

Definition at line 96 of file search_graph.cc.

11.17.2.104 #define ALLTYPE_ALLOCED_SM 29

11.17.2.105 #define ALLTYPE_ALLOCED_SM 29

11.17.2.106 #define ALLTYPE_ALLOCED_SM 29

11.17.2.107 #define ALLTYPE_ALLOCED_SM 29

11.17.2.108 #define ALLTYPE_ALLOCED_SM 29

11.17.2.109 #define ALLTYPE_ALLOCED_SN 16

11.17.2.110 #define ALLTYPE_ALLOCED_SN 16

11.17.2.111 #define ALLTYPE_ALLOCED_SN 16

11.17.2.112 #define ALLTYPE_ALLOCED_SN 16

a sparse_vector<int> is stored

Definition at line 80 of file search_graph.cc.

11.17.2.113 #define ALLTYPE_ALLOCED_SN 16

11.17.2.114 #define ALLTYPE_ALLOCED_SN 16

11.17.2.115 #define ALLTYPE_ALLOCED_SU 17

11.17.2.116 #define ALLTYPE_ALLOCED_SU 17

11.17.2.117 #define ALLTYPE_ALLOCED_SU 17

11.17.2.118 #define ALLTYPE_ALLOCED_SU 17

11.17.2.119 #define ALLTYPE_ALLOCED_SU 17

a sparse_vector<unsigned int> is stored

Definition at line 81 of file search_graph.cc.

11.17.2.120 #define ALLTYPE_ALLOCED_SU 17

11.17.2.121 #define ALLTYPE_ALLOCED_SX 21

11.17.2.122 #define ALLTYPE_ALLOCED_SX 21

11.17.2.123 #define ALLTYPE_ALLOCED_SX 21

11.17.2.124 #define ALLTYPE_ALLOCED_SX 21

11.17.2.125 #define ALLTYPE_ALLOCED_SX 21

a sparse_vector<Number> is stored

Definition at line 85 of file search_graph.cc.

11.17.2.126 #define ALLTYPE_ALLOCED_SX 21

11.17.2.127 #define ALLTYPE_ALLOCED_U 3

11.17.2.128 #define ALLTYPE_ALLOCED_U 3

a vector<unsigned int> is stored

Definition at line 65 of file search_graph.cc.

11.17.2.129 #define ALLTYPE_ALLOCED_U 3

11.17.2.130 #define ALLTYPE_ALLOCED_U 3

11.17.2.131 #define ALLTYPE_ALLOCED_U 3

11.17.2.132 #define ALLTYPE_ALLOCED_U 3

11.17.2.133 #define ALLTYPE_ALLOCED_VB 8

a vector<vector<bool> > is stored

Definition at line 71 of file search_graph.cc.

11.17.2.134 #define ALLTYPE_ALLOCED_VB 8

11.17.2.135 #define ALLTYPE_ALLOCED_VB 8

11.17.2.136 #define ALLTYPE_ALLOCED_VB 8

11.17.2.137 #define ALLTYPE_ALLOCED_VB 8

11.17.2.138 #define ALLTYPE_ALLOCED_VB 8

11.17.2.139 #define ALLTYPE_ALLOCED_VC 13

11.17.2.140 #define ALLTYPE_ALLOCED_VC 13

a vector<vector<string> > is stored

Definition at line 76 of file search_graph.cc.

11.17.2.141 #define ALLTYPE_ALLOCED_VC 13

11.17.2.142 #define ALLTYPE_ALLOCED_VC 13

11.17.2.143 #define ALLTYPE_ALLOCED_VC 13

11.17.2.144 #define ALLTYPE_ALLOCED_VC 13

11.17.2.145 #define ALLTYPE_ALLOCED_VD 11

11.17.2.146 #define ALLTYPE_ALLOCED_VD 11

11.17.2.147 #define ALLTYPE_ALLOCED_VD 11

a vector<vector<double> > is stored

Definition at line 74 of file search_graph.cc.

11.17.2.148 #define ALLTYPE_ALLOCED_VD 11

11.17.2.149 #define ALLTYPE_ALLOCED_VD 11

11.17.2.150 #define ALLTYPE_ALLOCED_VD 11

11.17.2.151 #define ALLTYPE_ALLOCED_VI 12

11.17.2.152 #define ALLTYPE_ALLOCED_VI 12

11.17.2.153 #define ALLTYPE_ALLOCED_VI 12

a vector<vector<interval> > is stored

Definition at line 75 of file search_graph.cc.

11.17.2.154 #define ALLTYPE_ALLOCED_VI 12

11.17.2.155 #define ALLTYPE_ALLOCED_VI 12

11.17.2.156 #define ALLTYPE_ALLOCED_VI 12

11.17.2.157 #define ALLTYPE_ALLOCED_VN 9

11.17.2.158 #define ALLTYPE_ALLOCED_VN 9

a vector<vector<int> > is stored

Definition at line 72 of file search_graph.cc.

11.17.2.159 #define ALLTYPE_ALLOCED_VN 9

11.17.2.160 #define ALLTYPE_ALLOCED_VN 9

11.17.2.161 #define ALLTYPE_ALLOCED_VN 9

11.17.2.162 #define ALLTYPE_ALLOCED_VN 9

11.17.2.163 #define ALLTYPE_ALLOCED_VU 10

11.17.2.164 #define ALLTYPE_ALLOCED_VU 10

a vector<vector<unsigned int> > is stored

Definition at line 73 of file search_graph.cc.

11.17.2.165 #define ALLTYPE_ALLOCED_VU 10

11.17.2.166 #define ALLTYPE_ALLOCED_VU 10

11.17.2.167 #define ALLTYPE_ALLOCED_VU 10

11.17.2.168 #define ALLTYPE_ALLOCED_VU 10

11.17.2.169 #define ALLTYPE_ALLOCED_VX 14

11.17.2.170 #define ALLTYPE_ALLOCED_VX 14

a vector<vector<Number> > is stored

Definition at line 77 of file search_graph.cc.

11.17.2.171 #define ALLTYPE_ALLOCED_VX 14

11.17.2.172 #define ALLTYPE_ALLOCED_VX 14

11.17.2.173 #define ALLTYPE_ALLOCED_VX 14

11.17.2.174 #define ALLTYPE_ALLOCED_VX 14

11.17.2.175 #define ALLTYPE_ALLOCED_X 7

11.17.2.176 #define ALLTYPE_ALLOCED_X 7

11.17.2.177 #define ALLTYPE_ALLOCED_X 7

11.17.2.178 #define ALLTYPE_ALLOCED_X 7

11.17.2.179 #define ALLTYPE_ALLOCED_X 7

a vector<Number> is stored

Definition at line 69 of file search_graph.cc.

11.17.2.180 #define ALLTYPE_ALLOCED_X 7

11.17.2.181 #define ALLTYPE_ALLOCED_XDM 28

11.17.2.182 #define ALLTYPE_ALLOCED_XDM 28

11.17.2.183 #define ALLTYPE_ALLOCED_XDM 28

a dense_matrix<Number> is stored

Definition at line 94 of file search_graph.cc.

11.17.2.184 #define ALLTYPE_ALLOCED_XDM 28

11.17.2.185 #define ALLTYPE_ALLOCED_XDM 28

11.17.2.186 #define ALLTYPE_ALLOCED_XDM 28

11.17.2.187 #define ALLTYPE_ALLOCED_XSM 33

11.17.2.188 #define ALLTYPE_ALLOCED_XSM 33

11.17.2.189 #define ALLTYPE_ALLOCED_XSM 33

11.17.2.190 #define ALLTYPE_ALLOCED_XSM 33

11.17.2.191 #define ALLTYPE_ALLOCED_XSM 33

a sparse_matrix<Number> is stored

Definition at line 100 of file search_graph.cc.

11.17.2.192 #define ALLTYPE_ALLOCED_XSM 33

11.17.2.193 #define ALLTYPE_ALLOCED_Y 23

11.17.2.194 #define ALLTYPE_ALLOCED_Y 23

11.17.2.195 #define ALLTYPE_ALLOCED_Y 23

11.17.2.196 #define ALLTYPE_ALLOCED_Y 23

11.17.2.197 #define ALLTYPE_ALLOCED_Y 23

11.17.2.198 #define ALLTYPE_ALLOCED_Y 23

a Number is stored

Definition at line 88 of file search_graph.cc.

11.17.2.199 #define ALLTYPE_ANY 99

11.17.2.200 #define ALLTYPE_ANY 99

11.17.2.201 #define ALLTYPE_ANY 99

11.17.2.202 #define ALLTYPE_ANY 99

an unspecified type is stored

Definition at line 104 of file search_graph.cc.

11.17.2.203 #define ALLTYPE_ANY 99

11.17.2.204 #define ALLTYPE_ANY 99

11.17.2.205 #define ALLTYPE_BOOL -1

11.17.2.206 #define ALLTYPE_BOOL -1

11.17.2.207 #define ALLTYPE_BOOL -1

a bool is stored

Definition at line 59 of file search_graph.cc.

11.17.2.208 #define ALLTYPE_BOOL -1

11.17.2.209 #define ALLTYPE_BOOL -1

11.17.2.210 #define ALLTYPE_BOOL -1

11.17.2.211 #define ALLTYPE_DOUBLE -4

11.17.2.212 #define ALLTYPE_DOUBLE -4

a double is stored

Definition at line 56 of file search_graph.cc.

11.17.2.213 #define ALLTYPE_DOUBLE -4

11.17.2.214 #define ALLTYPE_DOUBLE -4

11.17.2.215 #define ALLTYPE_DOUBLE -4

11.17.2.216 #define ALLTYPE_DOUBLE -4

11.17.2.217 #define ALLTYPE_EMPTY 0

11.17.2.218 #define ALLTYPE_EMPTY 0

11.17.2.219 #define ALLTYPE_EMPTY 0

11.17.2.220 #define ALLTYPE_EMPTY 0

the basic_alltype is empty

Definition at line 61 of file search_graph.cc.

11.17.2.221 #define ALLTYPE_EMPTY 0

11.17.2.222 #define ALLTYPE_EMPTY 0

11.17.2.223 #define ALLTYPE_FIRST_UNDEF 34

11.17.2.224 #define ALLTYPE_FIRST_UNDEF 34

11.17.2.225 #define ALLTYPE_FIRST_UNDEF 34

this is undefined

Definition at line 102 of file search_graph.cc.

11.17.2.226 #define ALLTYPE_FIRST_UNDEF 34

11.17.2.227 #define ALLTYPE_FIRST_UNDEF 34

11.17.2.228 #define ALLTYPE_FIRST_UNDEF 34

11.17.2.229 #define ALLTYPE_INT -2

11.17.2.230 #define ALLTYPE_INT -2

11.17.2.231 #define ALLTYPE_INT -2

an int is stored

Definition at line 58 of file search_graph.cc.

11.17.2.232 #define ALLTYPE_INT -2

11.17.2.233 #define ALLTYPE_INT -2

11.17.2.234 #define ALLTYPE_INT -2

11.17.2.235 #define ALLTYPE_INTERVAL -6

11.17.2.236 #define ALLTYPE_INTERVAL -6

an interval is stored

Definition at line 54 of file search_graph.cc.

11.17.2.237 #define ALLTYPE_INTERVAL -6

11.17.2.238 #define ALLTYPE_INTERVAL -6

11.17.2.239 #define ALLTYPE_INTERVAL -6

11.17.2.240 #define ALLTYPE_INTERVAL -6

11.17.2.241 #define ALLTYPE_UINT -3

11.17.2.242 #define ALLTYPE_UINT -3

11.17.2.243 #define ALLTYPE_UINT -3

11.17.2.244 #define ALLTYPE_UINT -3

an unsigned int is stored

Definition at line 57 of file search_graph.cc.

11.17.2.245 #define ALLTYPE_UINT -3

11.17.2.246 #define ALLTYPE_UINT -3

11.17.2.247 #define ALLTYPE_VOID_PTR -5

11.17.2.248 #define ALLTYPE_VOID_PTR -5

11.17.2.249 #define ALLTYPE_VOID_PTR -5

11.17.2.250 #define ALLTYPE_VOID_PTR -5

11.17.2.251 #define ALLTYPE_VOID_PTR -5

a void∗ is stored

Definition at line 55 of file search_graph.cc.

11.17.2.252 #define ALLTYPE_VOID_PTR -5

11.17.2.253 #define BASIC_ALLTYPE_ASSERT(*A*)

Value:

```
do { \
      if(__cont_type != (A)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/1: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.254 #define BASIC_ALLTYPE_ASSERT(*A*)

Value:

```
do { \
      if(__cont_type != (A)) \
        throw api_exception(apiee_internal,\
```

```
          std::string("api/basic_alltype.h/1: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.255 #define BASIC_ALLTYPE_ASSERT(A)

Value:

```
do { \
      if(__cont_type != (A)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/1: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.256 #define BASIC_ALLTYPE_ASSERT(A)

Value:

```
do { \
      if(__cont_type != (A)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/1: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.257 #define BASIC_ALLTYPE_ASSERT(A)

Value:

```
do { \
      if(__cont_type != (A)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/1: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.258 #define BASIC_ALLTYPE_ASSERT2(A, B)

Value:

```
do { \
      if(__cont_type != (A) && __cont_type != (B)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/2: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.259 #define BASIC_ALLTYPE_ASSERT2(A, B)

Value:

```
do { \
      if(__cont_type != (A) && __cont_type != (B)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/2: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.260 #define BASIC_ALLTYPE_ASSERT2(A, B)

Value:

```
do { \
      if(__cont_type != (A) && __cont_type != (B)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/2: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.261 #define BASIC_ALLTYPE_ASSERT2(A, B)

Value:

```
do { \
      if(__cont_type != (A) && __cont_type != (B)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/2: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.262 #define BASIC_ALLTYPE_ASSERT2(A, B)

Value:

```
do { \
      if(__cont_type != (A) && __cont_type != (B)) \
        throw api_exception(apiee_internal,\
          std::string("api/basic_alltype.h/2: Basic alltype ")+type_cstr((A))+\
                      " requested, type was "+type_name());\
    } while(0)
```

11.17.2.263 #define BASIC_ALLTYPE_CHECK 1

11.17.2.264 #define BASIC_ALLTYPE_CHECK 1

11.17.2.265 #define BASIC_ALLTYPE_CHECK 1

11.17.2.266 #define BASIC_ALLTYPE_CHECK 1

11.17.2.267 #define BASIC_ALLTYPE_CHECK 1

11.17.2.268 #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)

Value:

```
do { \
      if (__cont_type != (CT) && is_empty_vector()) \
        /* The const_cast is ugly, but there's no other option. */ \
        const_cast<basic_alltype *>(this)->operator=(std::vector<T>()); \
    } while(0)
```

11.17.2.269 #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)

Value:

```
do { \
      if (__cont_type != (CT) && is_empty_vector()) \
        /* The const_cast is ugly, but there's no other option. */ \
        const_cast<basic_alltype *>(this)->operator=(std::vector<T>()); \
    } while(0)
```

11.17.2.270 #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)

Value:

```
do { \
      if (__cont_type != (CT) && is_empty_vector()) \
        /* The const_cast is ugly, but there's no other option. */ \
        const_cast<basic_alltype *>(this)->operator=(std::vector<T>()); \
    } while(0)
```

11.17.2.271 #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)

Value:

```
do { \
      if (__cont_type != (CT) && is_empty_vector()) \
        /* The const_cast is ugly, but there's no other option. */ \
        const_cast<basic_alltype *>(this)->operator=(std::vector<T>()); \
    } while(0)
```

11.17.2.272 #define BASIC_ALLTYPE_FIX_EMPTY_VECTOR(CT, T)

Value:

```
do { \
      if (__cont_type != (CT) && is_empty_vector()) \
        /* The const_cast is ugly, but there's no other option. */ \
        const_cast<basic_alltype *>(this)->operator=(std::vector<T>()); \
    } while(0)
```

11.17.2.273 #define BASIC_ALLTYPE_NAMES

Value:

```
{ \
      "interval", "void*", "double", "unsigned int", "int", "bool", "empty", \
      "bool vector", "int vector", "unsigned int vector", "double vector", \
      "interval vector", "string vector", "number vector", \
      "bool vector vector", "int vector vector", "unsigned int vector vector",
      "double vector vector", \
      "interval vector vector", "string vector vector", "number vector vector",
       \
      "sparse bool vector", "sparse int vector", "sparse unsigned int vector",
      "sparse double vector", \
      "sparse interval vector", "sparse string vector", "sparse number vector",
       \
      "string", "number", \
      "dense double matrix", "dense int matrix", "dense interval matrix", "
      dense string matrix", "dense number matrix", \
      "sparse double matrix", "sparse int matrix", "sparse interval matrix", "
      sparse string matrix", "sparse number matrix" }
```

11.17.2.274 #define BASIC_ALLTYPE_NAMES

Value:

```
{ \
      "interval", "void*", "double", "unsigned int", "int", "bool", "empty", \
      "bool vector", "int vector", "unsigned int vector", "double vector", \
      "interval vector", "string vector", "number vector", \
      "bool vector vector", "int vector vector", "unsigned int vector vector",
      "double vector vector", \
      "interval vector vector", "string vector vector", "number vector vector",
       \
      "sparse bool vector", "sparse int vector", "sparse unsigned int vector",
      "sparse double vector", \
      "sparse interval vector", "sparse string vector", "sparse number vector",
       \
      "string", "number", \
      "dense double matrix", "dense int matrix", "dense interval matrix", "
      dense string matrix", "dense number matrix", \
      "sparse double matrix", "sparse int matrix", "sparse interval matrix", "
      sparse string matrix", "sparse number matrix" }
```

Definition at line 106 of file search_graph.cc.

11.17.2.275 #define BASIC_ALLTYPE_NAMES

Value:

```
{ \
      "interval", "void*", "double", "unsigned int", "int", "bool", "empty", \
      "bool vector", "int vector", "unsigned int vector", "double vector", \
      "interval vector", "string vector", "number vector", \
      "bool vector vector", "int vector vector", "unsigned int vector vector",
      "double vector vector", \
      "interval vector vector", "string vector vector", "number vector vector",
       \
      "sparse bool vector", "sparse int vector", "sparse unsigned int vector",
      "sparse double vector", \
      "sparse interval vector", "sparse string vector", "sparse number vector",
       \
      "string", "number", \
      "dense double matrix", "dense int matrix", "dense interval matrix", "
      dense string matrix", "dense number matrix", \
      "sparse double matrix", "sparse int matrix", "sparse interval matrix", "
      sparse string matrix", "sparse number matrix" }
```

11.17.2.276 #define BASIC_ALLTYPE_NAMES

Value:

```
{ \
      "interval", "void*", "double", "unsigned int", "int", "bool", "empty", \
      "bool vector", "int vector", "unsigned int vector", "double vector", \
      "interval vector", "string vector", "number vector", \
      "bool vector vector", "int vector vector", "unsigned int vector vector",
      "double vector vector", \
      "interval vector vector", "string vector vector", "number vector vector",
       \
      "sparse bool vector", "sparse int vector", "sparse unsigned int vector",
      "sparse double vector", \
      "sparse interval vector", "sparse string vector", "sparse number vector",
       \
```

```
      "string", "number", \
      "dense double matrix", "dense int matrix", "dense interval matrix", "
      dense string matrix", "dense number matrix", \
      "sparse double matrix", "sparse int matrix", "sparse interval matrix", "
      sparse string matrix", "sparse number matrix" }
```

11.17.2.277 #define BASIC_ALLTYPE_NAMES

Value:

```
{ \
      "interval", "void*", "double", "unsigned int", "int", "bool", "empty", \
      "bool vector", "int vector", "unsigned int vector", "double vector", \
      "interval vector", "string vector", "number vector", \
      "bool vector vector", "int vector vector", "unsigned int vector vector",
      "double vector vector", \
      "interval vector vector", "string vector vector", "number vector vector",
       \
      "sparse bool vector", "sparse int vector", "sparse unsigned int vector",
      "sparse double vector", \
      "sparse interval vector", "sparse string vector", "sparse number vector",
       \
      "string", "number", \
      "dense double matrix", "dense int matrix", "dense interval matrix", "
      dense string matrix", "dense number matrix", \
      "sparse double matrix", "sparse int matrix", "sparse interval matrix", "
      sparse string matrix", "sparse number matrix" }
```

11.17.2.278 #define BASIC_ALLTYPE_NAMES

Value:

```
{ \
      "interval", "void*", "double", "unsigned int", "int", "bool", "empty", \
      "bool vector", "int vector", "unsigned int vector", "double vector", \
      "interval vector", "string vector", "number vector", \
      "bool vector vector", "int vector vector", "unsigned int vector vector",
      "double vector vector", \
      "interval vector vector", "string vector vector", "number vector vector",
       \
      "sparse bool vector", "sparse int vector", "sparse unsigned int vector",
      "sparse double vector", \
      "sparse interval vector", "sparse string vector", "sparse number vector",
       \
      "string", "number", \
      "dense double matrix", "dense int matrix", "dense interval matrix", "
      dense string matrix", "dense number matrix", \
      "sparse double matrix", "sparse int matrix", "sparse interval matrix", "
      sparse string matrix", "sparse number matrix" }
```

11.17.2.279 #define BASIC_ALLTYPE_THROW throw(api_exception)

Definition at line 142 of file search_graph.cc.

11.17.2.280 #define BASIC_ALLTYPE_THROW throw(api_exception)

11.17.2.281 #define BASIC_ALLTYPE_THROW throw(api_exception)

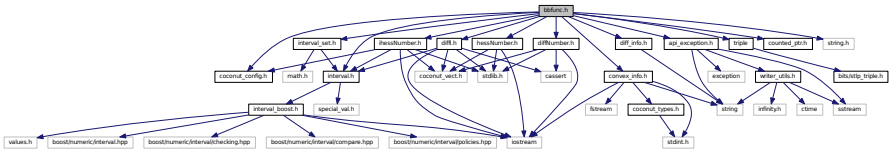
11.17.2.282 #define BASIC_ALLTYPE_THROW throw(api_exception)

11.17.2.283 #define BASIC_ALLTYPE_THROW throw(api_exception)

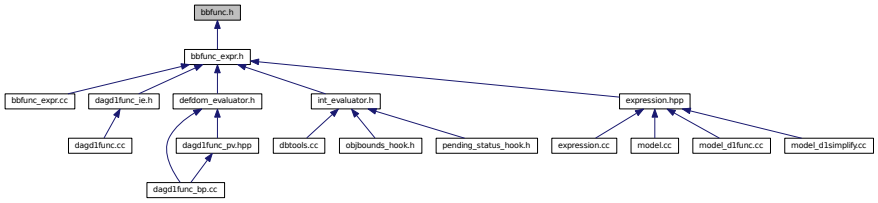
11.17.2.284 #define BASIC_ALLTYPE_THROW throw(api_exception)

11.18 bbfunc.h File Reference

`#include <coconut_config.h>` `#include <interval.h>` `#include <interval_-set.h>` `#include <triple>` `#include <api_exception.h>` `#include <convex_-info.h>` `#include <counted_ptr.h>` `#include <string.h>` `#include <diffNumber.-h>` `#include <hessNumber.h>` `#include <diffI.h>` `#include <ihessNumber.h>` × `#include <diff_info.h>` Include dependency graph for bbfunc.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::bbfunc

  The bbfunc class (black box function)

Namespaces

- namespace coco

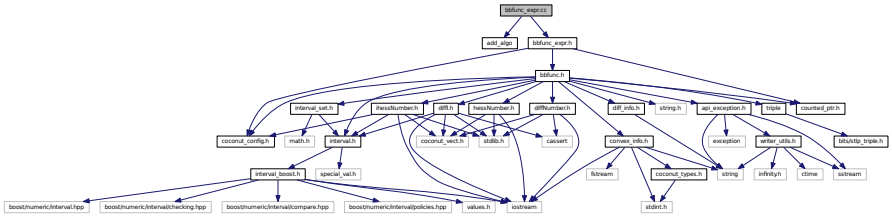
  the main namespace of the COCONUT API

11.18.1 Detailed Description

Definition in file bbfunc.h.

11.19 bbfunc_expr.cc File Reference

`#include <add_algo>` `#include <bbfunc_expr.h>` Include dependency graph for bbfunc-_expr.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define NBB_DEBUG 0
- #define BBFUNC_MAX_EVAL_INTSET 128

11.19.1 Detailed Description

Definition in file bbfunc_expr.cc.

11.19.2 Define Documentation

11.19.2.1 #define BBFUNC_MAX_EVAL_INTSET 128

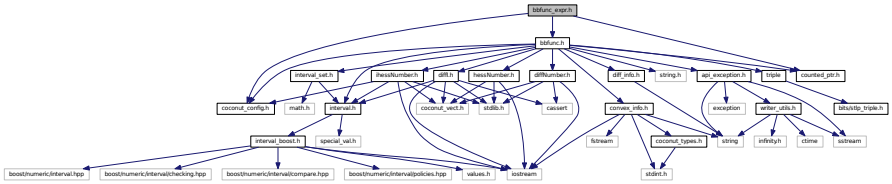
Definition at line 117 of file bbfunc_expr.cc.

11.19.2.2 #define NBB_DEBUG 0

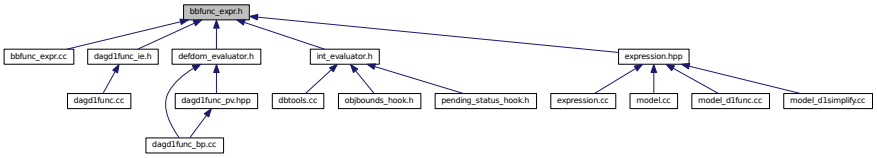
Definition at line 31 of file bbfunc_expr.cc.

11.20 bbfunc_expr.h File Reference

`#include <coconut_config.h>` `#include <bbfunc.h>` `#include <counted_ptr.-h>` Include dependency graph for bbfunc_expr.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::bbfunc_expr

  The bbfunc_expr class (black box function expression)

Namespaces

- namespace coco

  the main namespace of the COCONUT API

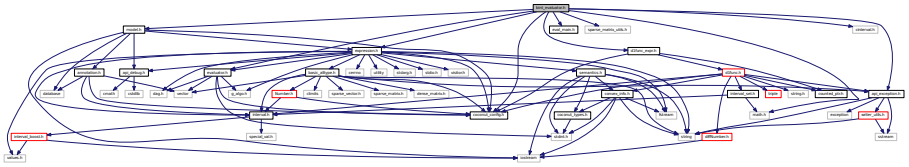
11.20.1 Detailed Description

Definition in file bbfunc_expr.h.

11.21 bint_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <sparse_matrix_-`

utils.h> #include <math.h> #include <cinterval.h> #include <d1func_expr.-h> #include <api_exception.h> Include dependency graph for bint_evaluator.h:



Classes

- struct coco::b_interval_eval_type
- class coco::b_interval_eval

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

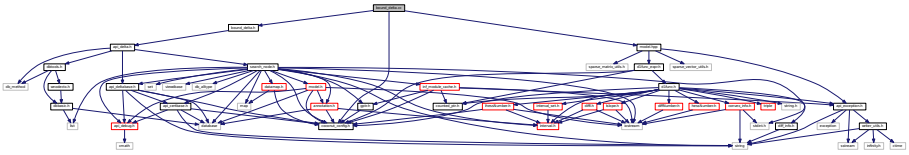
- typedef b_interval(∗ coco::b_interval_evaluator)(const std::vector< b_interval > ∗__x, const variable-_indicator &__v)

11.21.1 Detailed Description

Definition in file bint_evaluator.h.

11.22 bound_delta.cc File Reference

#include <coconut_config.h> #include <bound_delta.h> #include <model.-hpp> Include dependency graph for bound_delta.cc:



Namespaces

- namespace coco

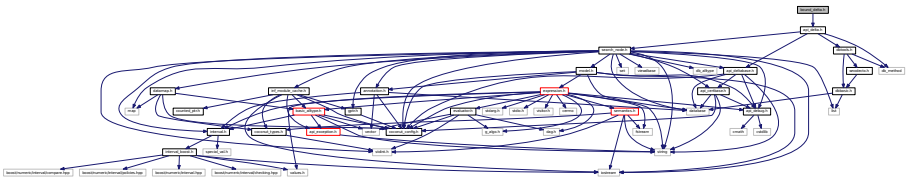
  the main namespace of the COCONUT API

11.22.1 Detailed Description

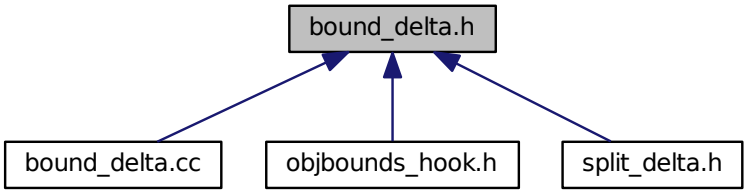
Definition in file bound_delta.cc.

11.23 bound_delta.h File Reference

`#include <api_delta.h>` Include dependency graph for bound_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::bound_undelta

  The bound undelta class for undoing changes to the node bounds in a model.
- class coco::bound_delta

  The bound delta class for changing the node bounds within a model.

Namespaces

- namespace coco

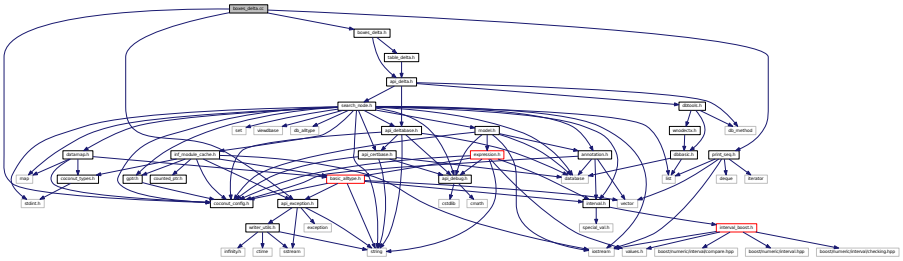
  the main namespace of the COCONUT API

11.23.1 Detailed Description

Definition in file bound_delta.h.

11.24 boxes_delta.cc File Reference

`#include <coconut_config.h>` `#include <boxes_delta.h>` `#include <print_seq.h>` `#include <api_exception.h>` Include dependency graph for boxes_delta.cc:



Namespaces

- namespace coco

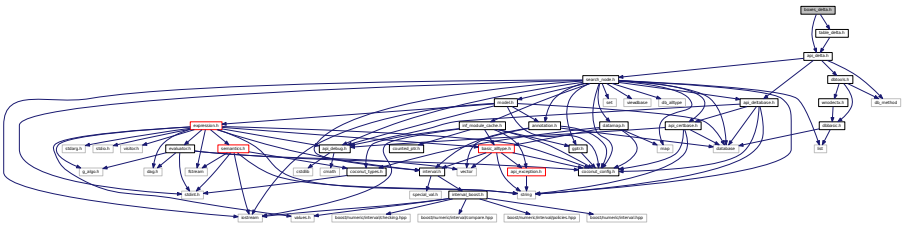
  the main namespace of the COCONUT API

11.24.1 Detailed Description

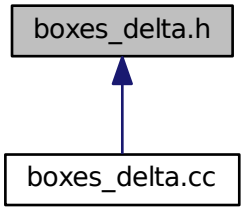
Definition in file boxes_delta.cc.

11.25 boxes_delta.h File Reference

`#include <api_delta.h>` `#include <table_delta.h>` Include dependency graph for boxes_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::boxes_delta
  A delta class which adds new boxes to the search database.

Namespaces

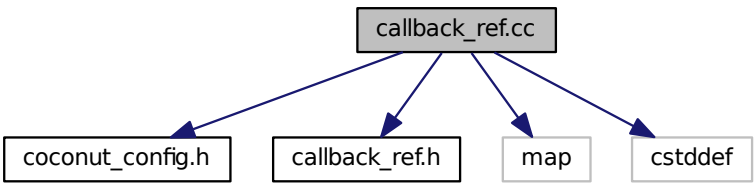
- namespace coco
  the main namespace of the COCONUT API

11.25.1 Detailed Description

Definition in file boxes_delta.h.

11.26 callback_ref.cc File Reference

`#include <coconut_config.h> #include <callback_ref.h> #include <map> ×
#include <cstddef>` Include dependency graph for callback_ref.cc:



Functions

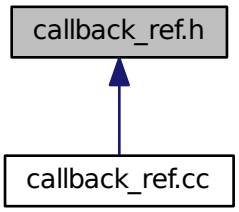
- cb_ref_handle _new_cb_ref (void *dest)
- int delete_cb_ref (cb_ref_handle ref)
- void * cb_ref (cb_ref_handle ref)

11.26.1 Detailed Description

Definition in file callback_ref.cc.

11.27 callback_ref.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define new_cb_ref() _new_cb_ref((void*)this)

Typedefs

- typedef unsigned int cb_ref_handle

Functions

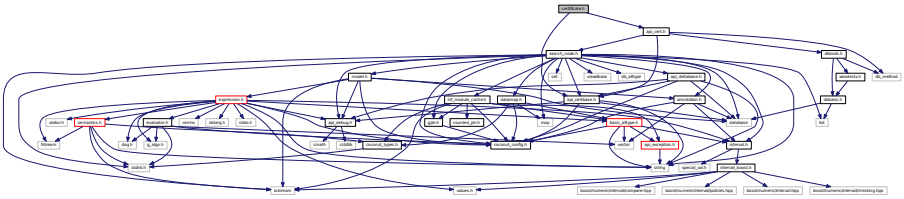
- cb_ref_handle _new_cb_ref (void *dest)
- int delete_cb_ref (cb_ref_handle ref)
- void * cb_ref (cb_ref_handle ref)

11.27.1 Detailed Description

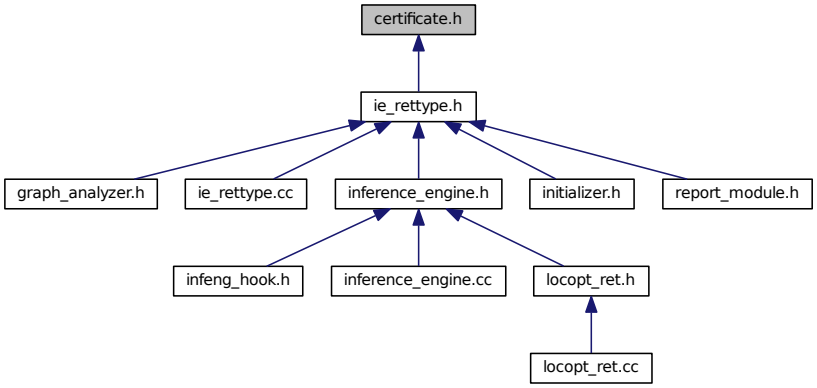
Definition in file callback_ref.h.

11.28 certificate.h File Reference

`#include <api_certbase.h>` `#include <api_cert.h>` Include dependency graph for certificate.h-
:



This graph shows which files directly or indirectly include this file:



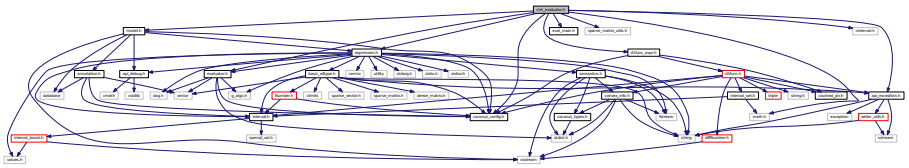
11.28.1 Detailed Description

Definition in file certificate.h.

11.29 cint_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-
h>` `#include <model.h>` `#include <eval_main.h>` `#include <sparse_matrix_-
utils.h>` `#include <math.h>` `#include <cinterval.h>` `#include <d1func_expr.-`

h> #include <api_exception.h> Include dependency graph for cint_evaluator.h:



Classes

- struct coco::cinterval_eval_type
- class coco::cinterval_eval

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef cinterval(∗ coco::cinterval_evaluator)(const std::vector< cinterval > ∗__x, const variable_-indicator &__v)

11.29.1 Detailed Description

Definition in file cint_evaluator.h.

11.30 coconut_config.h File Reference

Defines

- #define PURE_VIRTUAL { throw "Pure virtual function called!"; }
- #define INTERVAL_LIBRARY_FILIB 0
- #define INTERVAL_LIBRARY_BOOST 1
- #define COCONUT_INTERVAL_LIBRARY INTERVAL_LIBRARY_FILIB
- #define COCONUT_LINUX 1
- #define __COCONUT_PRETTY_FUNCTION__ __PRETTY_FUNCTION__

11.30.1 Detailed Description

Definition in file coconut_config.h.

11.30.2 Define Documentation

11.30.2.1 #define __COCONUT_PRETTY_FUNCTION__ __PRETTY_FUNCTION__

Definition at line 56 of file coconut_config.h.

11.30.2.2 #define COCONUT_INTERVAL_LIBRARY INTERVAL_LIBRARY_FILIB

Definition at line 45 of file coconut_config.h.

11.30.2.3 #define COCONUT_LINUX 1

Definition at line 47 of file coconut_config.h.

11.30.2.4 #define INTERVAL_LIBRARY_BOOST 1

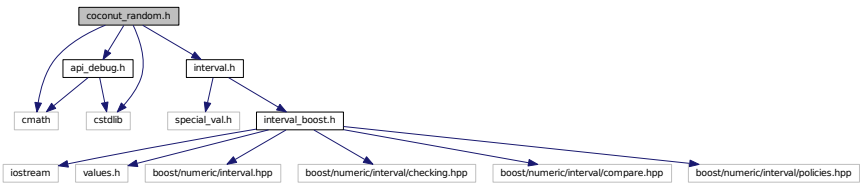
Definition at line 43 of file coconut_config.h.

11.30.2.5 #define INTERVAL_LIBRARY_FILIB 0

Definition at line 42 of file coconut_config.h.

11.31 coconut_random.h File Reference

`#include <cmath>` `#include <cstdlib>` `#include <api_debug.h>` `#include <interval.-h>` Include dependency graph for coconut_random.h:



Classes

- struct coco::coconut_random_f

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define coconut_random random
- #define coconut_seed(A) srandom(A)
- #define COCONUT_RAND_MAX RAND_MAX
- #define COCONUT_RRAND_MIN -1.e08
- #define COCONUT_RRAND_MAX +1.e08
- #define INIT_SEED coconut_random()
- #define coconut_init_random() coconut_seed(INIT_SEED)
- #define COCONUT_RRAND_MIN_BETA 0.01
- #define COCONUT_RRAND_MIN_ALPHA 0.5
- #define COCONUT_RRAND_MAX_ALPHA 0.99

Typedefs

- typedef long int coco::rand_t

Functions

- double coco::d_random ()
- double coco::r_random_h_eval (double t, double b, double a)
- double coco::r_random_hinv_eval (double x, double b, double a)
- double coco::r_random (double l, double u, double beta=10.0, double alpha=0.9)
- double coco::r_random (const interval &_i, double beta=10.0, double alpha=0.9)
- interval coco::i_random (double l, double r, double beta=10.0, double alpha=0.9)
- interval coco::i_random (const interval &_i, double beta=10.0, double alpha=0.9)
- interval coco::i_random ()

11.31.1 Detailed Description

Definition in file coconut_random.h.

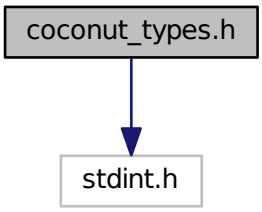
11.31.2 Define Documentation

11.31.2.1 #define INIT_SEED coconut_random()

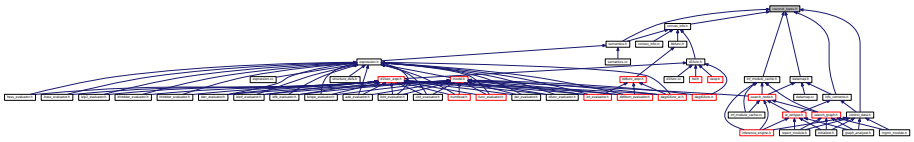
Definition at line 79 of file coconut_random.h.

11.32 coconut_types.h File Reference

`#include <stdint.h>` Include dependency graph for coconut_types.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef enum coco::tristate_e coco::tristate
- typedef uint32_t coco::search_node_id

  Type of the unique search node identifier.

Enumerations

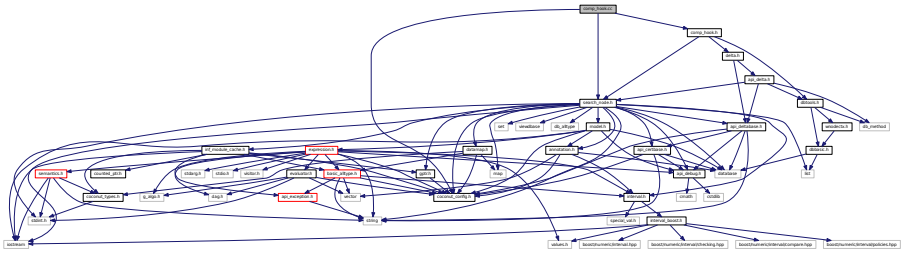
- enum coco::tristate_e { coco::t_true = 1, coco::t_false = -1, coco::t_maybe = 0 }

11.32.1 Detailed Description

Definition in file coconut_types.h.

11.33 comp_hook.cc File Reference

`#include <coconut_config.h>` `#include <search_node.h>` `#include <comp_hook.h>` Include dependency graph for comp_hook.cc:



Namespaces

- namespace coco

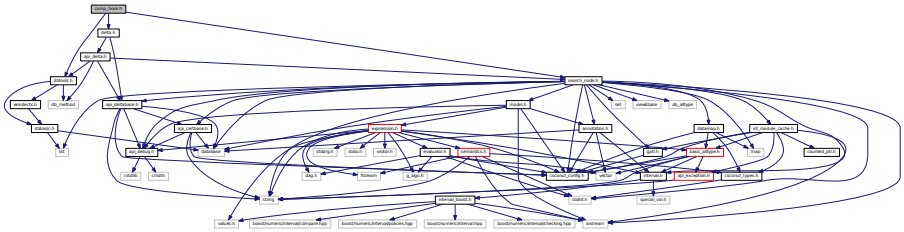
  the main namespace of the COCONUT API

11.33.1 Detailed Description

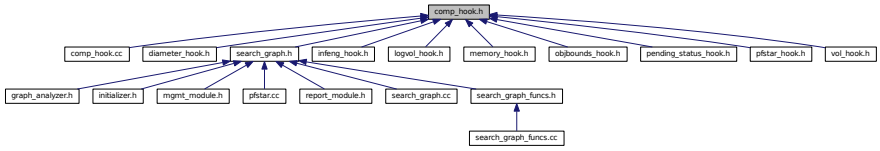
Definition in file comp_hook.cc.

11.34 comp_hook.h File Reference

`#include <dbtools.h>` `#include <search_node.h>` `#include <delta.h>` Include dependency graph for comp_hook.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::work_node_comp_hook

  The work_node_comp_hook class (work node computation hook)
- class coco::wnc_hook_base

  Base class for the work node computation hooks.

Namespaces

- namespace coco

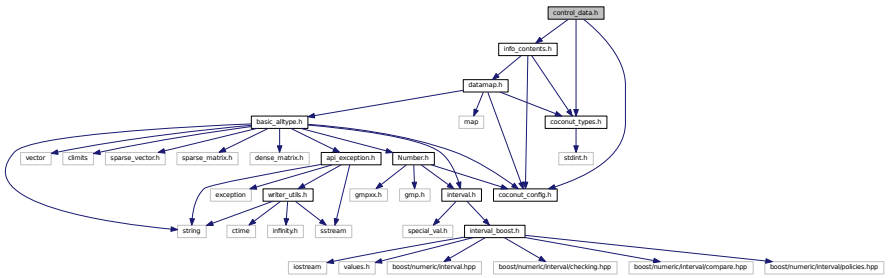
  the main namespace of the COCONUT API

11.34.1 Detailed Description

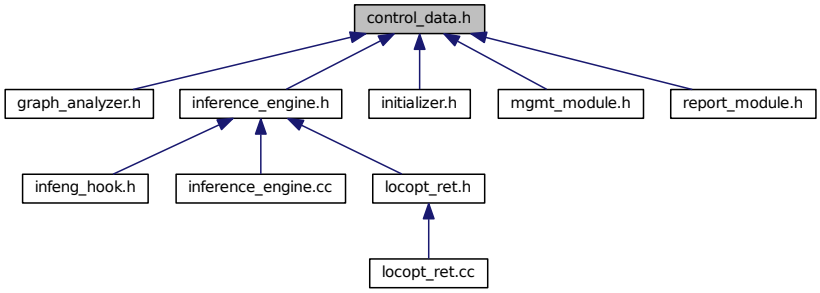
Definition in file comp_hook.h.

11.35 control_data.h File Reference

`#include <coconut_config.h> #include <coconut_types.h> #include <info-_contents.h>` Include dependency graph for control_data.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::control_data

  The class for communicating parameter information to COCONUT modules.

Namespaces

- namespace coco

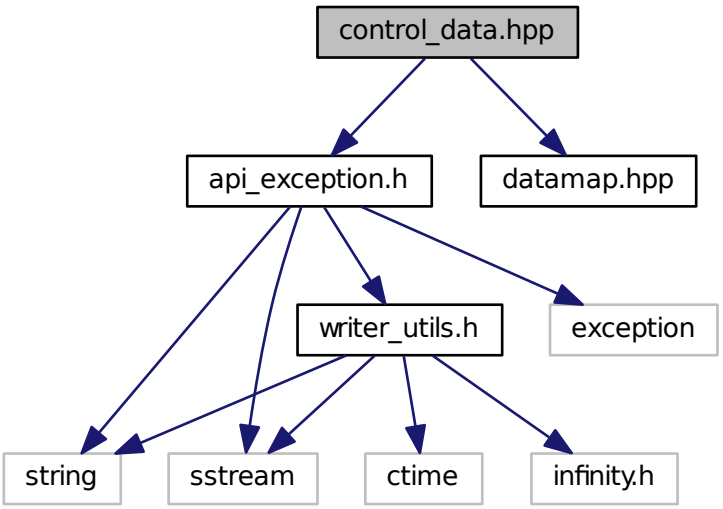
  the main namespace of the COCONUT API

11.35.1 Detailed Description

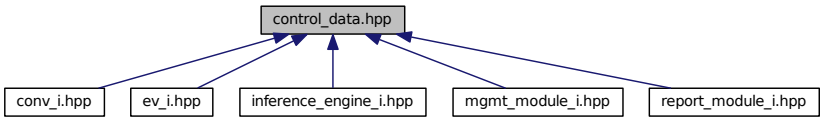
Definition in file control_data.h.

11.36 control_data.hpp File Reference

`#include <api_exception.h>` `#include <datamap.hpp>` Include dependency graph for control_data.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

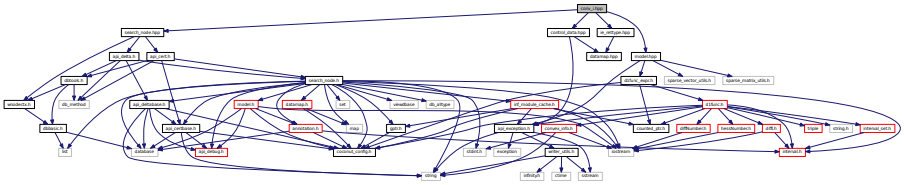
  the main namespace of the COCONUT API

11.36.1 Detailed Description

Definition in file control_data.hpp.

11.37 conv_i.hpp File Reference

`#include <control_data.hpp>` `#include <ie_rettype.hpp>` `#include <search-_node.hpp>` `#include <model.hpp>` Include dependency graph for conv_i.hpp:

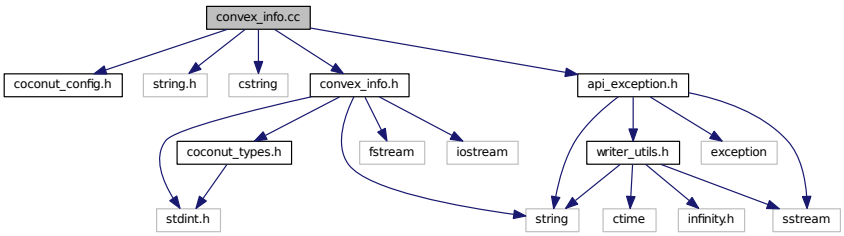


11.37.1 Detailed Description

Definition in file conv_i.hpp.

11.38 convex_info.cc File Reference

`#include <coconut_config.h>` `#include <string.h>` `#include <cstring>` `#include <convex_info.h>` `#include <api_exception.h>` Include dependency graph for convex_-info.cc:



Namespaces

- namespace coco

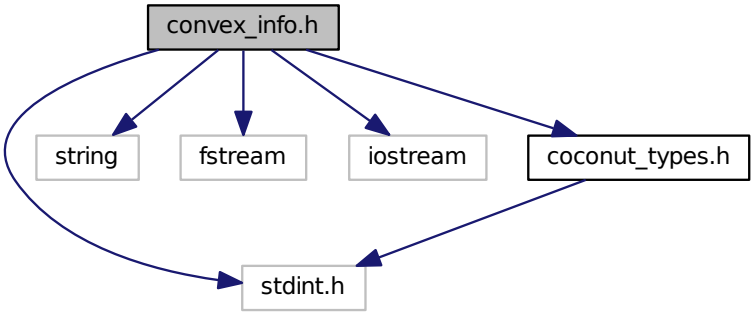
  the main namespace of the COCONUT API

11.38.1 Detailed Description

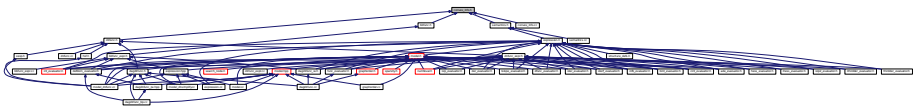
Definition in file convex_info.cc.

11.39 convex_info.h File Reference

`#include <stdint.h>` `#include <string>` `#include <fstream>` `#include <iostream>` × `#include <coconut_types.h>` Include dependency graph for convex_info.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::convex_e

  Convexity information.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Enumerations

- enum coco::convex_info { coco::c_convex = 1, coco::c_linear = 0, coco::c_concave = -1, coco::c_maybe = 2, coco::c_not = -2, coco::c_constant = 3 }

  Convexity information enum.

Functions

- bool coco::operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.
- bool coco::operator== (const convex_e &__c, const convex_info &__d)
- bool coco::operator== (const convex_info &__c, const convex_e &__d)
- bool coco::operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.
- bool coco::operator!= (const convex_e &__c, const convex_info &__d)
- bool coco::operator!= (const convex_info &__c, const convex_e &__d)
- std::ostream & coco::operator<< (std::ostream &o, const convex_e &__s)

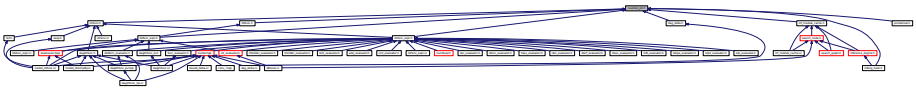
  C++ stream output operator for convex_e.

11.39.1 Detailed Description

Definition in file convex_info.h.

11.40 counted_ptr.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class counted_ptr
- class counted_ptr::counter

Defines

- #define DEBUG_COUNTED_PTR 0

11.40.1 Detailed Description

Definition in file counted_ptr.h.

11.40.2 Define Documentation

11.40.2.1 #define DEBUG_COUNTED_PTR 0

Definition at line 31 of file counted_ptr.h.

11.41 d1func.cc File Reference

`#include <d1func.h> #include <list> #include <vector> #include <NGS_step.h> #include <print_seq.h> #include <print_tuple.h> #include <sparse_vector.h>` Include dependency graph for d1func.cc:

Classes

- struct coco::cmp_point_info_i
- struct coco::cmp_point_info_s

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define N1D_DEBUG 9
- #define DIFFI_DEBUG 0
- #define D1FUNC_MAXITER 100
- #define diff_update(ret, i)

11.41.1 Detailed Description

Definition in file d1func.cc.

11.41.2 Define Documentation

11.41.2.1 #define D1FUNC_MAXITER 100

Definition at line 41 of file d1func.cc.

11.41.2.2 #define diff_update(*ret, i*)

Value:

```
do {                                                                \
    int newret = (i == 0 ? DIFF_EVAL_DISCONT : i-1);                \
    if(ret < DIFF_EVAL_DISCONT)                                     \
```

```
    ret = newret;                                                \
  else                                                           \
    ret = std::min(ret, newret);                                 \
} while(0)
```

Definition at line 2341 of file d1func.cc.

11.41.2.3 #define DIFFI_DEBUG 0

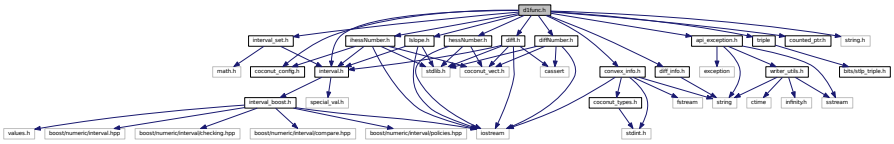
Definition at line 37 of file d1func.cc.

11.41.2.4 #define N1D_DEBUG 9

Definition at line 36 of file d1func.cc.

11.42 d1func.h File Reference

`#include <coconut_config.h>` `#include <interval.h>` `#include <interval_-set.h>` `#include <triple>` `#include <api_exception.h>` `#include <convex_-info.h>` `#include <counted_ptr.h>` `#include <string.h>` `#include <diffNumber.-h>` `#include <hessNumber.h>` `#include <diffI.h>` `#include <ihessNumber.h>` × `#include <Islope.h>` `#include <diff_info.h>` Include dependency graph for d1func.h:



This graph shows which files directly or indirectly include this file:

Classes

- class coco::d1func

  The d1func class (one dimensional function)
- struct coco::d1func::func_info

  the func_info class collects all information needed for optimal evaluation

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Enumerations

- enum coco::d1func_monotonicity_t { coco::d1fpmon_dec = -1, coco::d1fpmon_const = 0, coco::d1fpmon_inc = 1, coco::d1fpmon_unclear = 2, coco::d1fpmon_nonmon = -2 }
  The d1func_monotonicity_t enum.

Functions

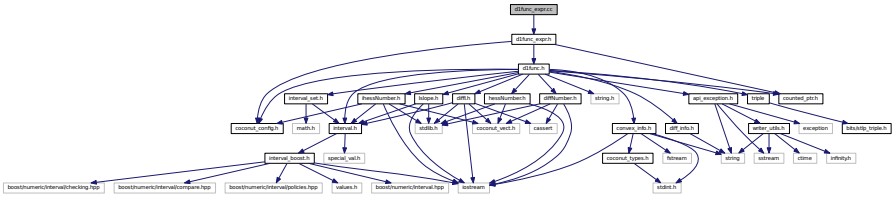
- d1func_monotonicity_t coco::operator- (d1func_monotonicity_t x)
  The d1func_monotonicity_t unary minus.

11.42.1 Detailed Description

Definition in file d1func.h.

11.43 d1func_expr.cc File Reference

`#include <d1func_expr.h>` Include dependency graph for d1func_expr.cc:



Namespaces

- namespace coco
  the main namespace of the COCONUT API

Defines

- #define N1D_DEBUG 0

11.43.1 Detailed Description

Definition in file d1func_expr.cc.

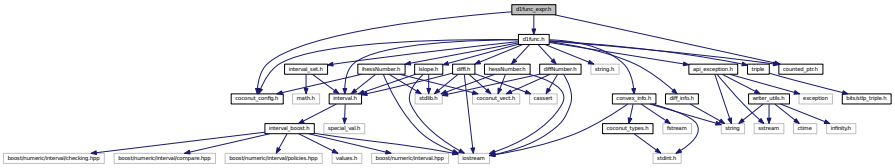
11.43.2 Define Documentation

11.43.2.1 #define N1D_DEBUG 0

Definition at line 30 of file d1func_expr.cc.

11.44 d1func_expr.h File Reference

`#include <coconut_config.h>` `#include <d1func.h>` `#include <counted_ptr.-h>` Include dependency graph for d1func_expr.h:



This graph shows which files directly or indirectly include this file:

Classes

- class coco::d1func_expr

  The d1func_expr class (one dimensional function expression)

Namespaces

- namespace coco

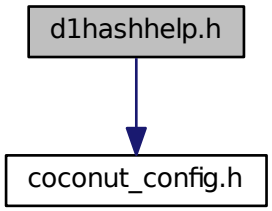
  the main namespace of the COCONUT API

11.44.1 Detailed Description

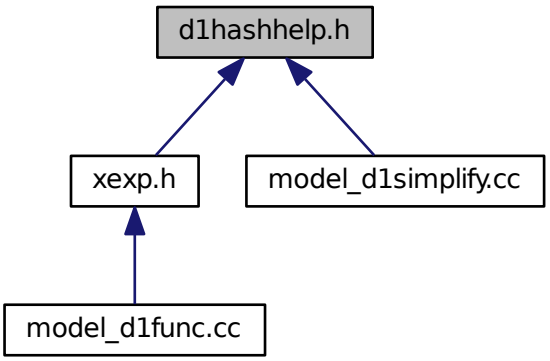
Definition in file d1func_expr.h.

11.45 d1hashhelp.h File Reference

`#include <coconut_config.h>` Include dependency graph for d1hashhelp.h:



This graph shows which files directly or indirectly include this file:



Classes

- union coco::double_to_uint64_u

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define HASH_CONSTANT_0 3UL
- #define HASH_CONSTANT_1 17UL
- #define HASH_CONSTANT_2 4259UL
- #define HASH_CONSTANT_3 8999UL
- #define HASH_CONSTANT_4 7127UL
- #define HASH_CONSTANT_5 5689UL
- #define HASH_CONSTANT_6 1619UL
- #define HASH_CONSTANT_7 2161UL
- #define HASH_CONSTANT_8 3659UL
- #define HCP_OPTYPE (-HASH_CONSTANT_0 * r.operator_type)
- #define HCP_TRANSFER(S) (HASH_CONSTANT_1 * S.thash)
- #define HCC_CTRANSFER(S) (HASH_CONSTANT_2 * S.chash)
- #define HCC_PARAMS(T) (HASH_CONSTANT_3 * double_to_uint64(r.params.T()))
- #define HCC_COEFFS(N) (HASH_CONSTANT_4 * double_to_uint64(r.coeffs[N]))
- #define HCP_SPARAMS(P) (HASH_CONSTANT_5 * r.params.P)
- #define HCP_N(N) (HASH_CONSTANT_6 * (N))
- #define HCC_DVECTOR(P) (dv_to_uint64(r.params.P()))
- #define HCC_IVECTOR(P) (iv_to_uint64(r.params.P()))
- #define HCC_NVECTOR(P) (nv_to_uint64(r.params.P()))
- #define HCC_DMATRIX(P) (dm_to_uint64(r.params.P()))
- #define HCC_IMATRIX(P) (im_to_uint64(r.params.P()))
- #define HCC_NMATRIX(P) (nm_to_uint64(r.params.P()))

Variables

- const char * coco::expr_names [ ]

11.45.1 Detailed Description

Definition in file d1hashhelp.h.

11.45.2 Define Documentation

11.45.2.1 #define HASH_CONSTANT_0 3UL

some prime numbers

Definition at line 38 of file d1hashhelp.h.

11.45.2.2 #define HASH_CONSTANT_1 17UL

Definition at line 39 of file d1hashhelp.h.

11.45.2.3 #define HASH_CONSTANT_2 4259UL

Definition at line 40 of file d1hashhelp.h.

11.45.2.4 #define HASH_CONSTANT_3 8999UL

Definition at line 41 of file d1hashhelp.h.

11.45.2.5 #define HASH_CONSTANT_4 7127UL

Definition at line 42 of file d1hashhelp.h.

11.45.2.6 #define HASH_CONSTANT_5 5689UL

Definition at line 43 of file d1hashhelp.h.

11.45.2.7 #define HASH_CONSTANT_6 1619UL

Definition at line 44 of file d1hashhelp.h.

11.45.2.8 #define HASH_CONSTANT_7 2161UL

Definition at line 45 of file d1hashhelp.h.

11.45.2.9 #define HASH_CONSTANT_8 3659UL

Definition at line 46 of file d1hashhelp.h.

11.45.2.10 #define HCC_COEFFS(*N*) (HASH_CONSTANT_4 * double_to_uint64(r.coeffs[N]))

Definition at line 252 of file d1hashhelp.h.

11.45.2.11 #define HCC_CTRANSFER(*S*) (HASH_CONSTANT_2 * S.chash)

Definition at line 250 of file d1hashhelp.h.

11.45.2.12 #define HCC_DMATRIX(*P*) (dm_to_uint64(r.params.P()))

Definition at line 258 of file d1hashhelp.h.

11.45.2.13 #define HCC_DVECTOR(*P*) (dv_to_uint64(r.params.P()))

Definition at line 255 of file d1hashhelp.h.

11.45.2.14 #define HCC_IMATRIX(*P*) (im_to_uint64(r.params.P()))

Definition at line 259 of file d1hashhelp.h.

11.45.2.15 #define HCC_IVECTOR(*P*) (iv_to_uint64(r.params.P()))

Definition at line 256 of file d1hashhelp.h.

11.45.2.16 #define HCC_NMATRIX(*P*) (nm_to_uint64(r.params.P()))

Definition at line 260 of file d1hashhelp.h.

11.45.2.17 #define HCC_NVECTOR(P) (nv_to_uint64(r.params.P()))

Definition at line 257 of file d1hashhelp.h.

11.45.2.18 #define HCC_PARAMS(T) (HASH_CONSTANT_3 * double_to_uint64(r.params.T()))

Definition at line 251 of file d1hashhelp.h.

11.45.2.19 #define HCP_N(N) (HASH_CONSTANT_6 * (N))

Definition at line 254 of file d1hashhelp.h.

11.45.2.20 #define HCP_OPTYPE (-HASH_CONSTANT_0 * r.operator_type)

Definition at line 248 of file d1hashhelp.h.

11.45.2.21 #define HCP_SPARAMS(P) (HASH_CONSTANT_5 * r.params.P)

Definition at line 253 of file d1hashhelp.h.

11.45.2.22 #define HCP_TRANSFER(S) (HASH_CONSTANT_1 * S.thash)

Definition at line 249 of file d1hashhelp.h.

11.46 d1testf.h File Reference

`#include <d1testf.h>` Include dependency graph for d1testf.h:

This graph shows which files directly or indirectly include this file:

Classes

- class coco::testpoly_func
- class coco::one_over_x_func

Namespaces

- namespace coco

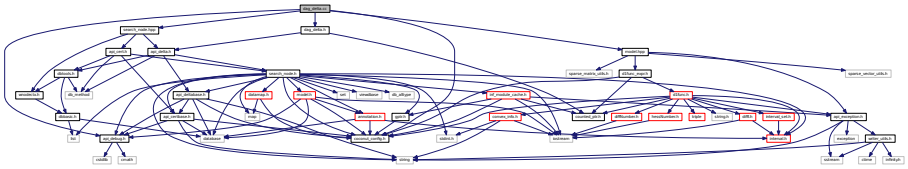
  the main namespace of the COCONUT API

11.46.1 Detailed Description

Definition in file d1testf.h.

11.47 dag_delta.cc File Reference

`#include <coconut_config.h>` `#include <dag_delta.h>` `#include <api_debug.h>` `#include <model.hpp>` `#include <search_node.hpp>` Include dependency graph for dag_delta.cc:



Namespaces

- namespace coco

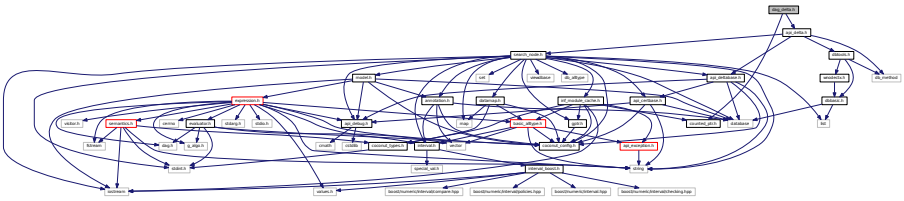
  the main namespace of the COCONUT API

11.47.1 Detailed Description

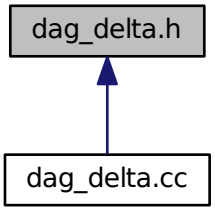
Definition in file dag_delta.cc.

11.48 dag_delta.h File Reference

`#include <api_delta.h>` `#include <counted_ptr.h>` Include dependency graph for dag-_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::dag_undelta

  The DAG undelta class for undoing changes to the DAG of a model.
- class coco::dag_undelta::__check_walkers
- class coco::dag_delta

  The DAG delta class for performing changes to the DAG of a model.
- struct coco::dag_delta::__docompare_nodes
- class coco::dag_delta::__check_nodes

Namespaces

- namespace coco

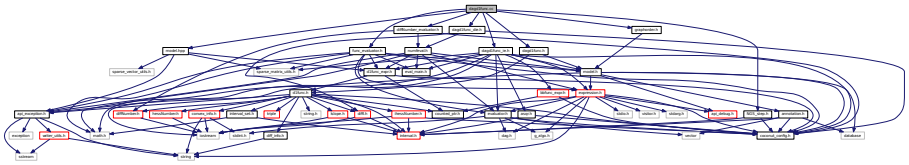
  the main namespace of the COCONUT API

11.48.1 Detailed Description

Definition in file dag_delta.h.

11.49 dagd1func.cc File Reference

`#include <dagd1func.h>` `#include <model.hpp>` `#include <func_evaluator.h>` `#include <dagd1func_ie.h>` `#include <diffNumber_evaluator.h>` `#include <dagd1func_die.h>` `#include <NGS_step.h>` `#include <graphorder.h>` Include dependency graph for dagd1func.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define N1D_DEBUG 9
- #define POINTLIST_DEBUG 2
- #define EVAL_DEBUG 2
- #define DAGD1FUNC_RELWIDTH_LIM 1.e-4

Functions

- interval coco::_internal_imperfect_eval (const interval &x, unsigned int k, const expression_const_walker &result, const model *mod, const variable_indicator &vind, const std::map< unsigned int, d1func * > *p_infos)

11.49.1 Detailed Description

Definition in file dagd1func.cc.

11.49.2 Define Documentation

11.49.2.1 #define DAGD1FUNC_RELWIDTH_LIM 1.e-4

Definition at line 43 of file dagd1func.cc.

11.49.2.2 #define EVAL_DEBUG 2

Definition at line 41 of file dagd1func.cc.

11.49.2.3 #define N1D_DEBUG 9

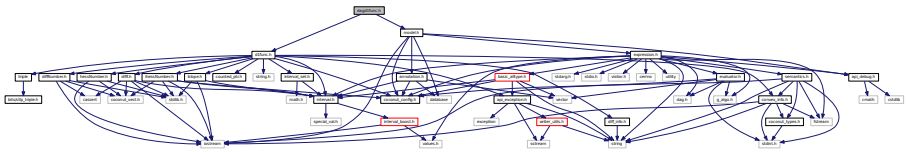
Definition at line 39 of file dagd1func.cc.

11.49.2.4 #define POINTLIST_DEBUG 2

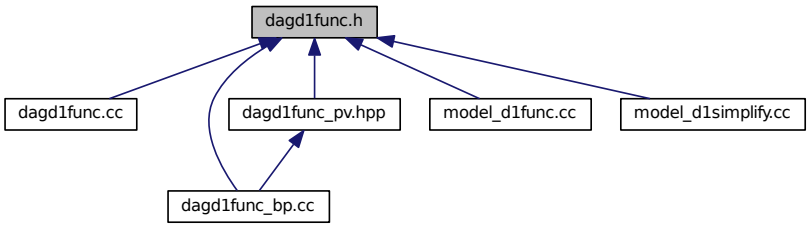
Definition at line 40 of file dagd1func.cc.

11.50 dagd1func.h File Reference

`#include <d1func.h>` `#include <model.h>` Include dependency graph for dagd1func.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::dag_d1func
  The dag_d1func class (one dimensional function)

Namespaces

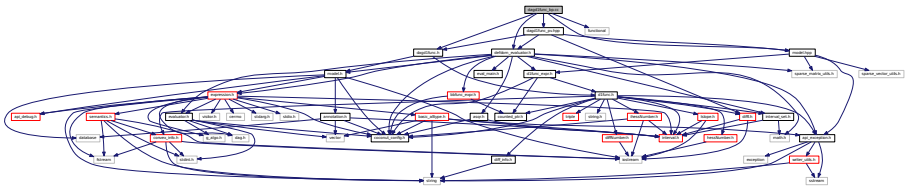
- namespace coco
  the main namespace of the COCONUT API

11.50.1 Detailed Description

Definition in file dagd1func.h.

11.51 dagd1func_bp.cc File Reference

`#include <dagd1func.h>` `#include <model.hpp>` `#include <defdom_evaluator.h>` `#include <functional>` `#include <dagd1func_pv.hpp>` Include dependency graph for dagd1func_bp.cc:



Classes

- class coco::compare_intervals

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define D1FP_EPS DBL_EPSILON
- #define N1D_BP_DEBUG 9
- #define IEPS interval(-D1FP_EPS,D1FP_EPS)

Functions

- interval_set coco::intersect_inv_hlp (d1func *d1help, const interval &xx, const interval_set &r, int order)

11.51.1 Detailed Description

Definition in file dagd1func_bp.cc.

11.51.2 Define Documentation

11.51.2.1 #define D1FP_EPS DBL_EPSILON

Definition at line 67 of file dagd1func_bp.cc.

11.51.2.2 #define IEPS interval(-D1FP_EPS,D1FP_EPS)

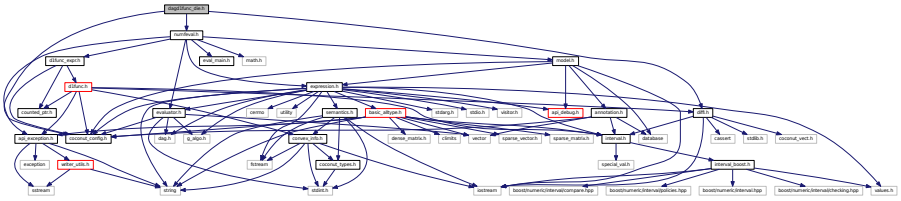
Definition at line 70 of file dagd1func_bp.cc.

11.51.2.3 #define N1D_BP_DEBUG 9

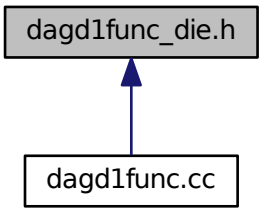
Definition at line 68 of file dagd1func_bp.cc.

11.52 dagd1func_die.h File Reference

`#include <coconut_config.h> #include <diffI.h> #include <numfeval.h>` ×
Include dependency graph for dagd1func_die.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco
  the main namespace of the COCONUT API

Defines

- #define xxxNumber_t diffI
- #define xxxNumber_name dd1f_diffI
- #define XXXNUMBER_HAS_DEGREE 1
- #define XXXNUMBER_HAS_1D 1
- #define XXXNUMBER_HAS_SQR 1
- #define XXXEVAL_HAS_BASES 1
- #define XXXNUMBER_EVAL_DEBUG 0

11.52.1 Detailed Description

Definition in file dagd1func_die.h.

11.52.2 Define Documentation

11.52.2.1 #define XXXEVAL_HAS_BASES 1

Definition at line 65 of file dagd1func_die.h.

11.52.2.2 #define XXXNUMBER_EVAL_DEBUG 0

Definition at line 70 of file dagd1func_die.h.

11.52.2.3 #define XXXNUMBER_HAS_1D 1

Definition at line 55 of file dagd1func_die.h.

11.52.2.4 #define XXXNUMBER_HAS_DEGREE 1

Definition at line 50 of file dagd1func_die.h.

11.52.2.5 #define XXXNUMBER_HAS_SQR 1

Definition at line 60 of file dagd1func_die.h.

11.52.2.6 #define xxxNumber_name dd1f_diffI

Definition at line 45 of file dagd1func_die.h.

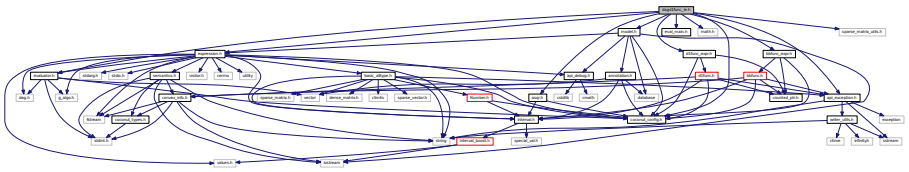
11.52.2.7 #define xxxNumber_t diffI

Definition at line 40 of file dagd1func_die.h.

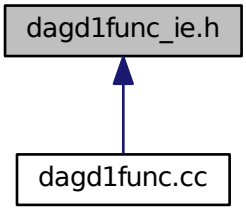
11.53 dagd1func_ie.h File Reference

```
#include <coconut_config.h> #include <evaluator.h> #include <expression.-
h> #include <model.h> #include <eval_main.h> #include <math.h> #include
```

<api_exception.h> #include <d1func_expr.h> #include <bbfunc_expr.h> #include <asqr.h> #include <sparse_matrix_utils.h> Include dependency graph for dagd1func_ie.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct coco::dd1f_interval_eval_type
  Visitor data for dd1f_interval_eval.
- class coco::dd1f_interval_eval
  Forward function range evaluation.

Namespaces

- namespace coco
  the main namespace of the COCONUT API

Typedefs

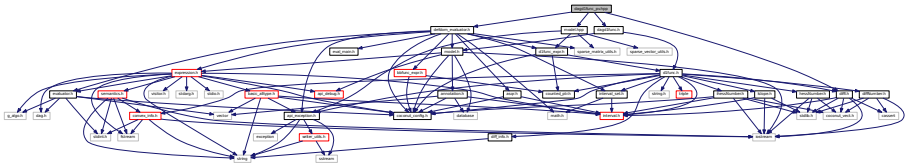
- typedef interval(∗ coco::dd1f_interval_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)

11.53.1 Detailed Description

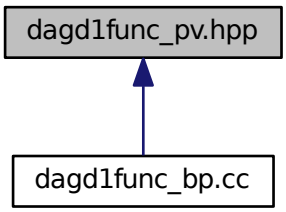
Definition in file dagd1func_ie.h.

11.54 dagd1func_pv.hpp File Reference

`#include <dagd1func.h>` `#include <model.hpp>` `#include <defdom_evaluator.h>` `#include <diffI.h>` Include dependency graph for dagd1func_pv.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct coco::polsppt_ret_type

  Return type for polsppt_eval.
- struct coco::polsppt_eval_type

  Visitor data for polsppt_eval.
- struct coco::polsppt_eval_type::help_t
- class coco::polsppt_eval

  Forward function range evaluation.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

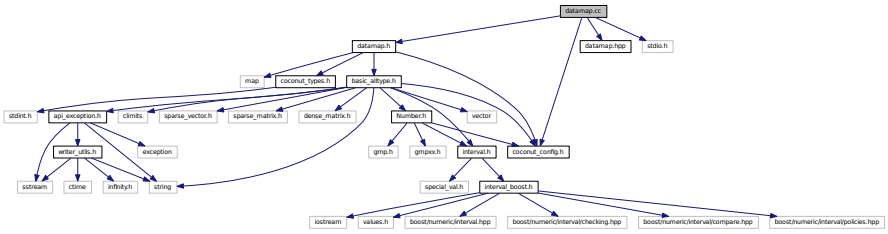
- typedef polsppt_ret_type(∗ coco::polsppt_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)
- typedef std::triple < d1func::d1func_point_t, diffI, int > coco::polsppt_info
- typedef std::multimap < unsigned int, polsppt_info > coco::polsppt_info_map
- typedef std::map< interval, polsppt_info_map, compare_intervals > coco::polsppt_interval_map
- typedef std::multimap < unsigned int, polsppt_info_map::const_iterator > coco::polsppt_map_map

11.54.1 Detailed Description

Definition in file dagd1func_pv.hpp.

11.55 datamap.cc File Reference

`#include <coconut_config.h>` `#include <datamap.h>` `#include <datamap.hpp>` × `#include <stdio.h>` Include dependency graph for datamap.cc:



Namespaces

- namespace coco

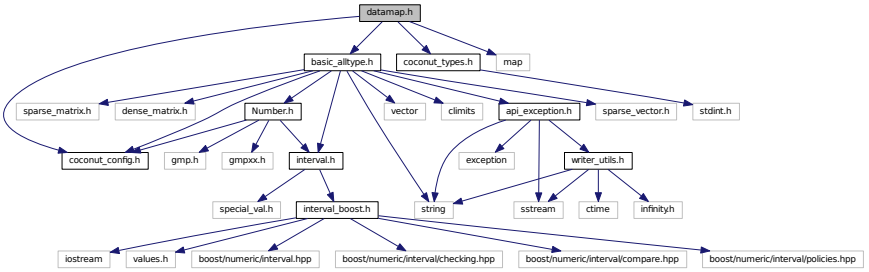
  the main namespace of the COCONUT API

11.55.1 Detailed Description

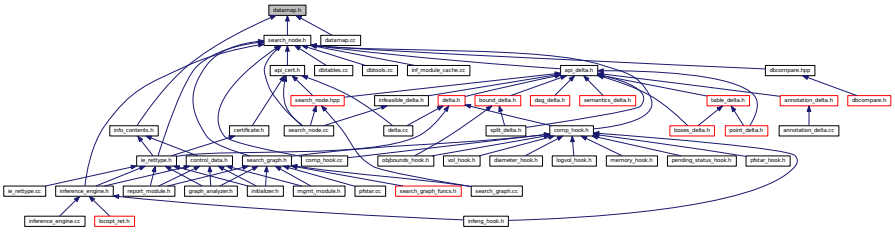
Definition in file datamap.cc.

11.56 datamap.h File Reference

`#include <coconut_config.h>` `#include <coconut_types.h>` `#include <basic_alltype.h>` `#include <map>` Include dependency graph for datamap.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::datamap

  The base class for communicating with COCONUT modules.

Namespaces

- namespace coco

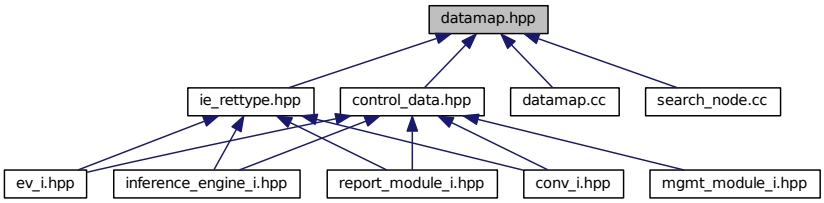
  the main namespace of the COCONUT API

11.56.1 Detailed Description

Definition in file datamap.h.

11.57 datamap.hpp File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define __DATAMAP_SEPARATOR "%"
- #define __DATAMAP_IDXSPEC "IDX"

11.57.1 Detailed Description

Definition in file datamap.hpp.

11.57.2 Define Documentation

11.57.2.1 #define __DATAMAP_IDXSPEC "IDX"

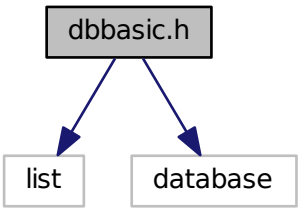
Definition at line 34 of file datamap.hpp.

11.57.2.2 #define __DATAMAP_SEPARATOR "%"

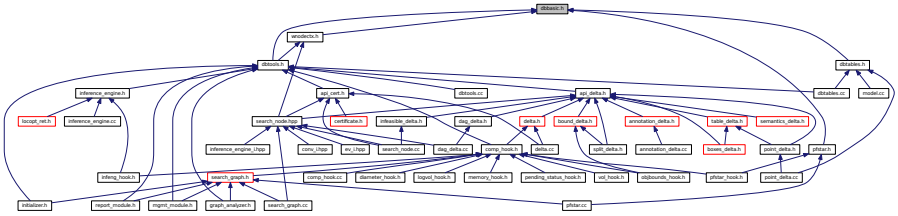
Definition at line 33 of file datamap.hpp.

11.58 dbbasic.h File Reference

`#include <list>` `#include <database>` Include dependency graph for dbbasic.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::dbt_row

  This type is used to hold one row in some table of the search database.

Namespaces

- namespace coco

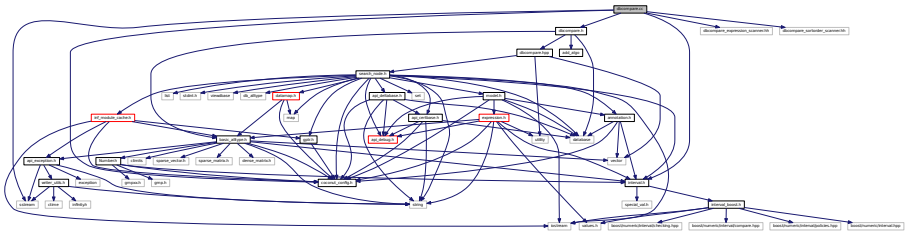
  the main namespace of the COCONUT API

11.58.1 Detailed Description

Definition in file dbbasic.h.

11.59 dbcompare.cc File Reference

`#include <coconut_config.h>` `#include <interval.h>` `#include <dbcompare.h>` `#include <sstream>` `#include "dbcompare_expression_scanner.hh"` `#include "dbcompare_sortorder_scanner.hh"` Include dependency graph for dbcompare.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

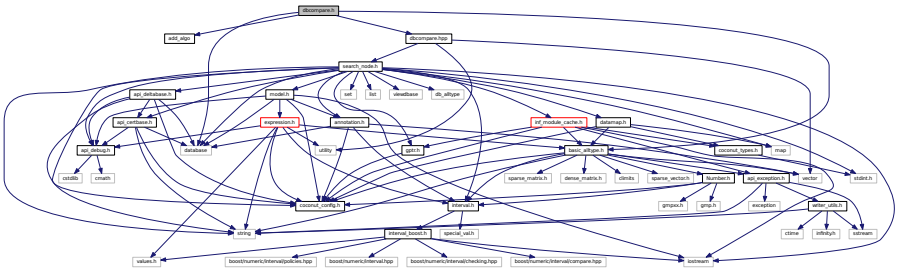
- template<typename _TR >
  bool coco::eval_in (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, _TR, const basic_alltype &I)
- template<>
  bool coco::eval_in< interval > (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, interval, const basic_alltype &I)
- template<>
  bool coco::eval_in< std::string > (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, std::string, const basic_alltype &I)
- template<typename _TR >
  bool coco::eval_s (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, _TR cmpval)
- template<>
  bool coco::eval_s< interval > (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, interval cmpval)
- template<typename _TR >
  bool coco::eval_u (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, _TR cmpval)
- bool coco::eval (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view *iv, const basic_alltype &cmpval)
- void coco::parse_dbcompare_expression (const std::string &s, std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, std::vector< std::vector< int > > &expr)
- void coco::parse_dbcompare_sortorder (const std::string &s, std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, std::vector< int > &order)

11.59.1 Detailed Description

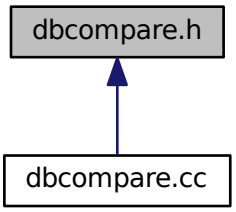
Definition in file dbcompare.cc.

11.60 dbcompare.h File Reference

`#include <add_algo>` `#include <database>` `#include <basic_alltype.h>` `#include <dbcompare.hpp>` Include dependency graph for dbcompare.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DB_COMPARE_CMP_EQ 0
- #define DB_COMPARE_CMP_NE 1

- #define DB_COMPARE_CMP_GT 2
- #define DB_COMPARE_CMP_GE 3
- #define DB_COMPARE_CMP_LT 4
- #define DB_COMPARE_CMP_LE 5
- #define DB_COMPARE_CMP_IN 6
- #define DB_COMPARE_CMP_ABSEQ 7
- #define DB_COMPARE_CMP_ABSNE 8
- #define DB_COMPARE_CMP_ABSGT 9
- #define DB_COMPARE_CMP_ABSGE 10
- #define DB_COMPARE_CMP_ABSLT 11
- #define DB_COMPARE_CMP_ABSLE 12
- #define DB_COMPARE_CMP_ABSIN 13
- #define dbc_is_cmp(X) ((X) <= DB_COMPARE_CMP_ABSIN)
- #define dbc_is_sort(X) ((X) > DB_COMPARE_CMP_ABSIN)
- #define DB_COMPARE_SORT_MIN 14
- #define DB_COMPARE_SORT_MAX 15
- #define DB_COMPARE_SORT_ABSMIN 16
- #define DB_COMPARE_SORT_ABSMAX 17
- #define DB_COMPARE_SORT_TRUE 18
- #define DB_COMPARE_SORT_FALSE 19

Enumerations

- enum coco::dbc_method { coco::dbc_cmp_eq = DB_COMPARE_CMP_EQ, coco::dbc_cmp_ne = DB_COMPARE_CMP_NE, coco::dbc_cmp_gt = DB_COMPARE_CMP_GT, coco::dbc_cmp_ge = DB_COMPARE_CMP_GE, coco::dbc_cmp_lt = DB_COMPARE_CMP_LT, coco::dbc_cmp_le = DB_COMPARE_CMP_LE, coco::dbc_cmp_in = DB_COMPARE_CMP_IN, coco::dbc_cmp_abseq = DB_COMPARE_CMP_ABSEQ, coco::dbc_cmp_absne = DB_COMPARE_CMP_ABSNE, coco::dbc_cmp_absgt = DB_COMPARE_CMP_ABSGT, coco::dbc_cmp_absge = DB_COMPARE_CMP_ABSGE, coco::dbc_cmp_abslt = DB_COMPARE_CMP_ABSLT, coco::dbc_cmp_absle = DB_COMPARE_CMP_ABSLE, coco::dbc_cmp_absin = DB_COMPARE_CMP_ABSIN, coco::dbc_sort_min = DB_COMPARE_SORT_MIN, coco::dbc_sort_max = DB_COMPARE_SORT_MAX, coco::dbc_sort_absmin = DB_COMPARE_SORT_ABSMIN, coco::dbc_sort_absmax = DB_COMPARE_SORT_ABSMAX, coco::dbc_sort_true = DB_COMPARE_SORT_TRUE, coco::dbc_sort_false = DB_COMPARE_SORT_FALSE }

Functions

- void coco::parse_dbcompare_expression (const std::string &s, std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, std::vector< std::vector< int > > &expr)
- void coco::parse_dbcompare_sortorder (const std::string &s, std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, std::vector< int > &order)
- template<typename _AT >
  void coco::filter_from_view (typename std::vector< std::pair< vdbl::rowid, _AT > > &riv, vdbl::view *iv, const std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, const std::vector< std::vector< int > > &expr)
- template<typename _AT >
  void coco::list_from_view (typename std::vector< std::pair< vdbl::rowid, _AT > > &riv, vdbl::view *iv, const std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, const std::vector< int > &sort, unsigned int n)

11.60.1 Detailed Description

Definition in file dbcompare.h.

11.60.2 Define Documentation

11.60.2.1 #define DB_COMPARE_CMP_ABSEQ 7

Definition at line 45 of file dbcompare.h.

11.60.2.2 #define DB_COMPARE_CMP_ABSGE 10

Definition at line 48 of file dbcompare.h.

11.60.2.3 #define DB_COMPARE_CMP_ABSGT 9

Definition at line 47 of file dbcompare.h.

11.60.2.4 #define DB_COMPARE_CMP_ABSIN 13

Definition at line 51 of file dbcompare.h.

11.60.2.5 #define DB_COMPARE_CMP_ABSLE 12

Definition at line 50 of file dbcompare.h.

11.60.2.6 #define DB_COMPARE_CMP_ABSLT 11

Definition at line 49 of file dbcompare.h.

11.60.2.7 #define DB_COMPARE_CMP_ABSNE 8

Definition at line 46 of file dbcompare.h.

11.60.2.8 #define DB_COMPARE_CMP_EQ 0

Definition at line 37 of file dbcompare.h.

11.60.2.9 #define DB_COMPARE_CMP_GE 3

Definition at line 40 of file dbcompare.h.

11.60.2.10 #define DB_COMPARE_CMP_GT 2

Definition at line 39 of file dbcompare.h.

11.60.2.11 #define DB_COMPARE_CMP_IN 6

Definition at line 43 of file dbcompare.h.

11.60.2.12 #define DB_COMPARE_CMP_LE 5

Definition at line 42 of file dbcompare.h.

11.60.2.13 #define DB_COMPARE_CMP_LT 4

Definition at line 41 of file dbcompare.h.

11.60.2.14 #define DB_COMPARE_CMP_NE 1

Definition at line 38 of file dbcompare.h.

11.60.2.15 #define DB_COMPARE_SORT_ABSMAX 17

Definition at line 60 of file dbcompare.h.

11.60.2.16 #define DB_COMPARE_SORT_ABSMIN 16

Definition at line 59 of file dbcompare.h.

11.60.2.17 #define DB_COMPARE_SORT_FALSE 19

Definition at line 62 of file dbcompare.h.

11.60.2.18 #define DB_COMPARE_SORT_MAX 15

Definition at line 58 of file dbcompare.h.

11.60.2.19 #define DB_COMPARE_SORT_MIN 14

Definition at line 57 of file dbcompare.h.

11.60.2.20 #define DB_COMPARE_SORT_TRUE 18

Definition at line 61 of file dbcompare.h.

11.60.2.21 #define dbc_is_cmp(*X*) ((X) <= DB_COMPARE_CMP_ABSIN)

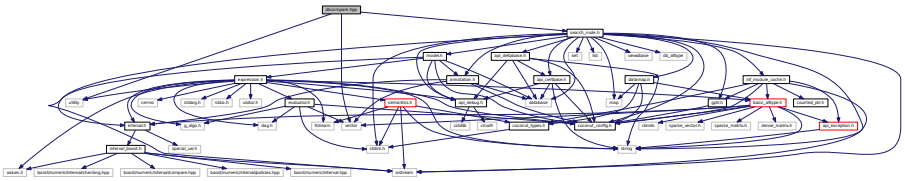
Definition at line 53 of file dbcompare.h.

11.60.2.22 #define dbc_is_sort(*X*) ((X) > DB_COMPARE_CMP_ABSIN)

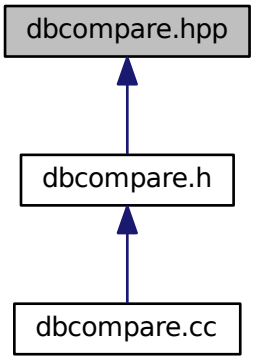
Definition at line 55 of file dbcompare.h.

11.61 dbcompare.hpp File Reference

`#include <vector> #include <utility> #include <search_node.h>` Include dependency graph for dbcompare.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::dbccmps_gt
- class coco::dbccmps_absgt
- class coco::dbccmps_lt
- class coco::dbccmps_abslt
- class coco::dbccmps_true
- class coco::dbccmps_false
- struct coco::dbccmp_true
- struct coco::dbccmp_false

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

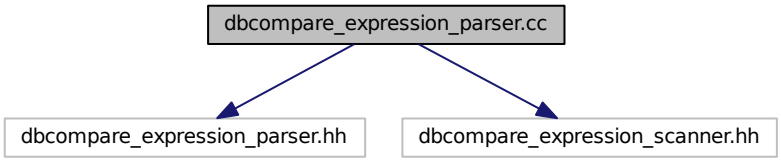
- bool coco::eval (const vdbl::rowid &rid, const std::string &colname, dbc_method tp, vdbl::view ∗iv, const basic_alltype &cmpval)
- template<typename _AT >
  void coco::filter_from_view (typename std::vector< std::pair< vdbl::rowid, _AT > > &riv, vdbl::view ∗iv, const std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, const std::vector< std::vector< int > > &expr)
- template<typename _TR >
  _TR coco::dbccmps_abs (const _TR &x)
- template<typename _TR , typename _AT >
  std::vector< std::pair < vdbl::rowid, _AT > >::iterator coco::filter_s (typename std::vector< std::pair< vdbl::rowid, _AT > > &rid, typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator rid_b, const std::string &colname, dbc_method tp, vdbl::view ∗iv, unsigned int n, _TR, double rel, double abs)
- template<typename _TR , typename _AT >
  std::vector< std::pair < vdbl::rowid, _AT > >::iterator coco::filter_u (typename std::vector< std::pair< vdbl::rowid, _AT > > &rid, typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator rid_b, const std::string &colname, dbc_method tp, vdbl::view ∗iv, unsigned int n, _TR)
- template<typename _AT >
  std::vector< std::pair < vdbl::rowid, _AT > >::iterator coco::filter (typename std::vector< std::pair< vdbl::rowid, _AT > > &rid, typename std::vector< std::pair< vdbl::rowid, _AT > >::iterator rid_b, const std::string &colname, dbc_method tp, vdbl::view ∗iv, unsigned int n, std::pair< double, double > rel_abs)
- template<typename _AT >
  void coco::list_from_view (typename std::vector< std::pair< vdbl::rowid, _AT > > &riv, vdbl::view ∗iv, const std::map< int, std::triple< std::string, dbc_method, basic_alltype > > &cols, const std::vector< int > &sort, unsigned int n)

11.61.1 Detailed Description

Definition in file dbcompare.hpp.

11.62 dbcompare_expression_parser.cc File Reference

`#include "dbcompare_expression_parser.hh"` `#include "dbcompare_expression_scanner.hh"` Include dependency graph for dbcompare_expression_parser.cc:



Namespaces

- namespace coco_api_internal

Defines

- #define yylex coco_api_internallex
- #define yylex scanner.yylex
- #define YY_(msgid) msgid
- #define YYUSE(e) ((void) (e))
- #define YYCDEBUG
- #define YY_SYMBOL_PRINT(Title, Type, Value, Location)
- #define YY_REDUCE_PRINT(Rule)
- #define YY_STACK_PRINT()
- #define YYACCEPT goto yyacceptlab
- #define YYABORT goto yyabortlab
- #define YYERROR goto yyerrorlab

11.62.1 Define Documentation

11.62.1.1 #define YY_(*msgid*) msgid

Definition at line 74 of file dbcompare_expression_parser.cc.

11.62.1.2 #define YY_REDUCE_PRINT(*Rule*)

Definition at line 114 of file dbcompare_expression_parser.cc.

11.62.1.3 #define YY_STACK_PRINT( )

Definition at line 115 of file dbcompare_expression_parser.cc.

11.62.1.4 #define YY_SYMBOL_PRINT(*Title, Type, Value, Location*)

Definition at line 113 of file dbcompare_expression_parser.cc.

11.62.1.5 #define YYABORT goto yyabortlab

Definition at line 120 of file dbcompare_expression_parser.cc.

11.62.1.6 #define YYACCEPT goto yyacceptlab

Definition at line 119 of file dbcompare_expression_parser.cc.

11.62.1.7 #define YYCDEBUG

Value:

```
for (bool yydebugcond_ = yydebug_; yydebugcond_; yydebugcond_ = false)  \
    (*yycdebug_)
```

Definition at line 82 of file dbcompare_expression_parser.cc.

11.62.1.8 #define YYERROR goto yyerrorlab

Definition at line 121 of file dbcompare_expression_parser.cc.

11.62.1.9 #define yylex coco_api_internallex

Definition at line 45 of file dbcompare_expression_parser.cc.

11.62.1.10 #define yylex scanner.yylex

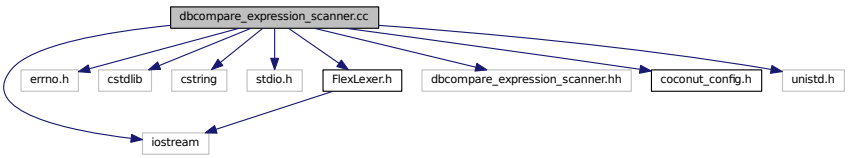
Definition at line 45 of file dbcompare_expression_parser.cc.

11.62.1.11 #define YYUSE(*e*) ((void) (e))

Definition at line 79 of file dbcompare_expression_parser.cc.

11.63 dbcompare_expression_scanner.cc File Reference

`#include <iostream>` `#include <errno.h>` `#include <cstdlib>` `#include <cstring>` × `#include <stdio.h>` `#include <FlexLexer.h>` `#include "dbcompare_expression_scanner.hh"` `#include <coconut_config.h>` `#include <unistd.h>` Include dependency graph for dbcompare_expression_scanner.cc:



Classes

- struct yy_buffer_state
- struct yy_trans_info

Defines

- #define YY_INT_ALIGNED short int
- #define FLEX_SCANNER
- #define YY_FLEX_MAJOR_VERSION 2
- #define YY_FLEX_MINOR_VERSION 5
- #define YY_FLEX_SUBMINOR_VERSION 33
- #define FLEX_BETA
- #define yyFlexLexer CoconutDBCompareExpressionFlexLexer
- #define FLEXINT_H
- #define INT8_MIN (-128)
- #define INT16_MIN (-32767-1)
- #define INT32_MIN (-2147483647-1)
- #define INT8_MAX (127)
- #define INT16_MAX (32767)
- #define INT32_MAX (2147483647)
- #define UINT8_MAX (255U)
- #define UINT16_MAX (65535U)
- #define UINT32_MAX (4294967295U)
- #define yyconst
- #define YY_NULL 0
- #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)
- #define BEGIN (yy_start) = 1 + 2 *
- #define YY_START (((yy_start) - 1) / 2)
- #define YYSTATE YY_START
- #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
- #define YY_NEW_FILE yyrestart(yyin)
- #define YY_END_OF_BUFFER_CHAR 0
- #define YY_BUF_SIZE 16384
- #define YY_STATE_BUF_SIZE ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))
- #define YY_TYPEDEF_YY_BUFFER_STATE
- #define EOB_ACT_CONTINUE_SCAN 0
- #define EOB_ACT_END_OF_FILE 1
- #define EOB_ACT_LAST_MATCH 2
- #define YY_LESS_LINENO(n)
- #define yyless(n)
- #define unput(c) yyunput(c, (yytext_ptr))
- #define YY_TYPEDEF_YY_SIZE_T
- #define YY_STRUCT_YY_BUFFER_STATE
- #define YY_BUFFER_NEW 0
- #define YY_BUFFER_NORMAL 1
- #define YY_BUFFER_EOF_PENDING 2
- #define YY_CURRENT_BUFFER
- #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]
- #define yy_new_buffer yy_create_buffer

- #define yy_set_interactive(is_interactive)
- #define yy_set_bol(at_bol)
- #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
- #define yywrap(n) 1
- #define YY_SKIP_YYWRAP
- #define yytext_ptr yytext
- #define YY_DECL int dbcompare_expression_scanner::yylex()
- #define YY_DO_BEFORE_ACTION
- #define YY_NUM_RULES 17
- #define YY_END_OF_BUFFER 18
- #define REJECT reject_used_but_not_detected
- #define yymore() yymore_used_but_not_detected
- #define YY_MORE_ADJ 0
- #define YY_RESTORE_YY_MORE_OFFSET
- #define INITIAL 0
- #define YY_EXTRA_TYPE void *
- #define YY_READ_BUF_SIZE 8192
- #define ECHO LexerOutput(yytext, yyleng)
- #define YY_INPUT(buf, result, max_size)
- #define yyterminate() return YY_NULL
- #define YY_START_STACK_INCR 25
- #define YY_FATAL_ERROR(msg) LexerError(msg)
- #define YY_USER_ACTION
- #define YY_BREAK break;
- #define YY_RULE_SETUP YY_USER_ACTION
- #define YY_EXIT_FAILURE 2
- #define yyless(n)
- #define YYTABLES_NAME "yytables"

Typedefs

- typedef signed char flex_int8_t
- typedef short int flex_int16_t
- typedef int flex_int32_t
- typedef unsigned char flex_uint8_t
- typedef unsigned short int flex_uint16_t
- typedef unsigned int flex_uint32_t
- typedef struct yy_buffer_state * YY_BUFFER_STATE
- typedef unsigned int yy_size_t
- typedef unsigned char YY_CHAR
- typedef coco_api_internal::dbcompare_expression_parser::token token

Functions

- void * CoconutDBCompareExpressionalloc (yy_size_t)
- void * CoconutDBCompareExpressionrealloc (void *, yy_size_t)
- void CoconutDBCompareExpressionfree (void *)
- if (!(yy_init))
- while (1)

Variables

- int yyleng
- YY_DECL register yy_state_type yy_current_state
- register char * yy_cp
- register char * yy_bp
- register int yy_act

11.63.1 Define Documentation

11.63.1.1 #define BEGIN (yy_start) = 1 + 2 *

Definition at line 137 of file dbcompare_expression_scanner.cc.

11.63.1.2 #define ECHO LexerOutput(yytext, yyleng)

Definition at line 502 of file dbcompare_expression_scanner.cc.

11.63.1.3 #define EOB_ACT_CONTINUE_SCAN 0

Definition at line 170 of file dbcompare_expression_scanner.cc.

11.63.1.4 #define EOB_ACT_END_OF_FILE 1

Definition at line 171 of file dbcompare_expression_scanner.cc.

11.63.1.5 #define EOB_ACT_LAST_MATCH 2

Definition at line 172 of file dbcompare_expression_scanner.cc.

11.63.1.6 #define FLEX_BETA

Definition at line 14 of file dbcompare_expression_scanner.cc.

11.63.1.7 #define FLEX_SCANNER

Definition at line 9 of file dbcompare_expression_scanner.cc.

11.63.1.8 #define FLEXINT_H

Definition at line 34 of file dbcompare_expression_scanner.cc.

11.63.1.9 #define INITIAL 0

Definition at line 469 of file dbcompare_expression_scanner.cc.

11.63.1.10 #define INT16_MAX (32767)

Definition at line 77 of file dbcompare_expression_scanner.cc.

11.63.1.11 #define INT16_MIN (-32767-1)

Definition at line 68 of file dbcompare_expression_scanner.cc.

11.63.1.12 #define INT32_MAX (2147483647)

Definition at line 80 of file dbcompare_expression_scanner.cc.

11.63.1.13 #define INT32_MIN (-2147483647-1)

Definition at line 71 of file dbcompare_expression_scanner.cc.

11.63.1.14 #define INT8_MAX (127)

Definition at line 74 of file dbcompare_expression_scanner.cc.

11.63.1.15 #define INT8_MIN (-128)

Definition at line 65 of file dbcompare_expression_scanner.cc.

11.63.1.16 #define REJECT reject_used_but_not_detected

Definition at line 426 of file dbcompare_expression_scanner.cc.

11.63.1.17 #define UINT16_MAX (65535U)

Definition at line 86 of file dbcompare_expression_scanner.cc.

11.63.1.18 #define UINT32_MAX (4294967295U)

Definition at line 89 of file dbcompare_expression_scanner.cc.

11.63.1.19 #define UINT8_MAX (255U)

Definition at line 83 of file dbcompare_expression_scanner.cc.

11.63.1.20 #define unput(*c*) yyunput(c, (yytext_ptr))

Definition at line 190 of file dbcompare_expression_scanner.cc.

11.63.1.21 #define YY_AT_BOL( ) (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)

Definition at line 309 of file dbcompare_expression_scanner.cc.

11.63.1.22 #define YY_BREAK break;

Definition at line 553 of file dbcompare_expression_scanner.cc.

11.63.1.23 #define YY_BUF_SIZE 16384

Definition at line 156 of file dbcompare_expression_scanner.cc.

11.63.1.24 #define YY_BUFFER_EOF_PENDING 2

Definition at line 263 of file dbcompare_expression_scanner.cc.

11.63.1.25 #define YY_BUFFER_NEW 0

Definition at line 251 of file dbcompare_expression_scanner.cc.

11.63.1.26 #define YY_BUFFER_NORMAL 1

Definition at line 252 of file dbcompare_expression_scanner.cc.

11.63.1.27 #define YY_CURRENT_BUFFER

Value:

```
( (yy_buffer_stack) \
                          ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
                          : NULL)
```

Definition at line 274 of file dbcompare_expression_scanner.cc.

11.63.1.28 #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]

Definition at line 281 of file dbcompare_expression_scanner.cc.

11.63.1.29 #define YY_DECL int dbcompare_expression_scanner::yylex()

Definition at line 325 of file dbcompare_expression_scanner.cc.

11.63.1.30 #define YY_DO_BEFORE_ACTION

Value:

```
(yytext_ptr) = yy_bp; \
        yyleng = (size_t) (yy_cp - yy_bp); \
        (yy_hold_char) = *yy_cp; \
        *yy_cp = '\0'; \
        (yy_c_buf_p) = yy_cp;
```

Definition at line 330 of file dbcompare_expression_scanner.cc.

11.63.1.31 #define YY_END_OF_BUFFER 18

Definition at line 338 of file dbcompare_expression_scanner.cc.

11.63.1.32 #define YY_END_OF_BUFFER_CHAR 0

Definition at line 152 of file dbcompare_expression_scanner.cc.

11.63.1.33 #define YY_EXIT_FAILURE 2

Definition at line 1564 of file dbcompare_expression_scanner.cc.

11.63.1.34 #define YY_EXTRA_TYPE void *

Definition at line 480 of file dbcompare_expression_scanner.cc.

11.63.1.35 #define YY_FATAL_ERROR(*msg*) LexerError(msg)

Definition at line 531 of file dbcompare_expression_scanner.cc.

11.63.1.36 #define YY_FLEX_MAJOR_VERSION 2

Definition at line 10 of file dbcompare_expression_scanner.cc.

11.63.1.37 #define YY_FLEX_MINOR_VERSION 5

Definition at line 11 of file dbcompare_expression_scanner.cc.

11.63.1.38 #define YY_FLEX_SUBMINOR_VERSION 33

Definition at line 12 of file dbcompare_expression_scanner.cc.

11.63.1.39 #define YY_INPUT(*buf, result, max_size*)

Value:

```
\
        if ( (result = LexerInput( (char *) buf, max_size )) < 0 ) \
                YY_FATAL_ERROR( "input in flex scanner failed" );
```

Definition at line 509 of file dbcompare_expression_scanner.cc.

11.63.1.40 #define YY_INT_ALIGNED short int

Definition at line 5 of file dbcompare_expression_scanner.cc.

11.63.1.41 #define YY_LESS_LINENO(*n*)

Definition at line 174 of file dbcompare_expression_scanner.cc.

11.63.1.42 #define YY_MORE_ADJ 0

Definition at line 428 of file dbcompare_expression_scanner.cc.

11.63.1.43 #define yy_new_buffer yy_create_buffer

Definition at line 287 of file dbcompare_expression_scanner.cc.

11.63.1.44 #define YY_NEW_FILE yyrestart(yyin)

Definition at line 150 of file dbcompare_expression_scanner.cc.

11.63.1.45 #define YY_NULL 0

Definition at line 124 of file dbcompare_expression_scanner.cc.

11.63.1.46 #define YY_NUM_RULES 17

Definition at line 337 of file dbcompare_expression_scanner.cc.

11.63.1.47 #define YY_READ_BUF_SIZE 8192

Definition at line 497 of file dbcompare_expression_scanner.cc.

11.63.1.48 #define YY_RESTORE_YY_MORE_OFFSET

Definition at line 429 of file dbcompare_expression_scanner.cc.

11.63.1.49 #define YY_RULE_SETUP YY_USER_ACTION

Definition at line 556 of file dbcompare_expression_scanner.cc.

11.63.1.50 #define YY_SC_TO_UI(*c*) ((unsigned int) (unsigned char) c)

Definition at line 131 of file dbcompare_expression_scanner.cc.

11.63.1.51 #define yy_set_bol(*at_bol*)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){\
        yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
            yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
        }
```

Definition at line 299 of file dbcompare_expression_scanner.cc.

11.63.1.52 #define yy_set_interactive(*is_interactive*)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){ \
        yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
            yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
        }
```

Definition at line 289 of file dbcompare_expression_scanner.cc.

11.63.1.53 #define YY_SKIP_YYWRAP

Definition at line 312 of file dbcompare_expression_scanner.cc.

11.63.1.54 #define YY_START (((yy_start) - 1) / 2)

Definition at line 143 of file dbcompare_expression_scanner.cc.

11.63.1.55 #define YY_START_STACK_INCR 25

Definition at line 526 of file dbcompare_expression_scanner.cc.

11.63.1.56 #define YY_STATE_BUF_SIZE ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))

Definition at line 161 of file dbcompare_expression_scanner.cc.

11.63.1.57 #define YY_STATE_EOF(*state*) (YY_END_OF_BUFFER + state + 1)

Definition at line 147 of file dbcompare_expression_scanner.cc.

11.63.1.58 #define YY_STRUCT_YY_BUFFER_STATE

Definition at line 203 of file dbcompare_expression_scanner.cc.

11.63.1.59 #define YY_TYPEDEF_YY_BUFFER_STATE

Definition at line 164 of file dbcompare_expression_scanner.cc.

11.63.1.60 #define YY_TYPEDEF_YY_SIZE_T

Definition at line 198 of file dbcompare_expression_scanner.cc.

11.63.1.61 #define YY_USER_ACTION

Definition at line 548 of file dbcompare_expression_scanner.cc.

11.63.1.62 #define yyconst

Definition at line 118 of file dbcompare_expression_scanner.cc.

11.63.1.63 #define yyFlexLexer CoconutDBCompareExpressionFlexLexer

Definition at line 23 of file dbcompare_expression_scanner.cc.

11.63.1.64 #define yyless(*n*)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                *yy_cp = (yy_hold_char); \
                YY_RESTORE_YY_MORE_OFFSET \
                (yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ;
      \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
```

Definition at line 1576 of file dbcompare_expression_scanner.cc.

11.63.1.65 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                yytext[yyleng] = (yy_hold_char); \
                (yy_c_buf_p) = yytext + yyless_macro_arg; \
                (yy_hold_char) = *(yy_c_buf_p); \
                *(yy_c_buf_p) = '\0'; \
                yyleng = yyless_macro_arg; \
                } \
        while ( 0 )
```

Definition at line 1576 of file dbcompare_expression_scanner.cc.

11.63.1.66 #define yymore( ) yymore_used_but_not_detected

Definition at line 427 of file dbcompare_expression_scanner.cc.

11.63.1.67 #define YYSTATE YY_START

Definition at line 144 of file dbcompare_expression_scanner.cc.

11.63.1.68 #define YYTABLES_NAME "yytables"

Definition at line 1638 of file dbcompare_expression_scanner.cc.

11.63.1.69 #define yyterminate( ) return YY_NULL

Definition at line 521 of file dbcompare_expression_scanner.cc.

11.63.1.70 #define yytext_ptr yytext

Definition at line 316 of file dbcompare_expression_scanner.cc.

11.63.1.71 #define yywrap(n) 1

Definition at line 311 of file dbcompare_expression_scanner.cc.

11.63.2 Typedef Documentation

11.63.2.1 typedef short int flex_int16_t

Definition at line 56 of file dbcompare_expression_scanner.cc.

11.63.2.2 typedef int flex_int32_t

Definition at line 57 of file dbcompare_expression_scanner.cc.

11.63.2.3 typedef signed char flex_int8_t

Definition at line 55 of file dbcompare_expression_scanner.cc.

11.63.2.4 typedef unsigned short int flex_uint16_t

Definition at line 59 of file dbcompare_expression_scanner.cc.

11.63.2.5 typedef unsigned int flex_uint32_t

Definition at line 60 of file dbcompare_expression_scanner.cc.

11.63.2.6 typedef unsigned char flex_uint8_t

Definition at line 58 of file dbcompare_expression_scanner.cc.

11.63.2.7 typedef coco_api_internal::dbcompare_expression_parser::token token

Definition at line 462 of file dbcompare_expression_scanner.cc.

11.63.2.8 typedef struct yy_buffer_state* YY_BUFFER_STATE

Definition at line 165 of file dbcompare_expression_scanner.cc.

11.63.2.9 typedef unsigned char YY_CHAR

Definition at line 314 of file dbcompare_expression_scanner.cc.

11.63.2.10 typedef unsigned int yy_size_t

Definition at line 199 of file dbcompare_expression_scanner.cc.

11.63.3 Function Documentation

11.63.3.1 void * CoconutDBCompareExpressionalloc (yy_size_t *size*)

Definition at line 1616 of file dbcompare_expression_scanner.cc.

11.63.3.2 void CoconutDBCompareExpressionfree (void * *ptr*)

Definition at line 1633 of file dbcompare_expression_scanner.cc.

11.63.3.3 void * CoconutDBCompareExpressionrealloc (void * *ptr*, yy_size_t *size*)

Definition at line 1621 of file dbcompare_expression_scanner.cc.

11.63.3.4 if (! *yy_init*)

Definition at line 572 of file dbcompare_expression_scanner.cc.

11.63.3.5 while (1)

Definition at line 598 of file dbcompare_expression_scanner.cc.

11.63.4 Variable Documentation

11.63.4.1 register int yy_act

Definition at line 565 of file dbcompare_expression_scanner.cc.

11.63.4.2 register char * yy_bp

Definition at line 564 of file dbcompare_expression_scanner.cc.

11.63.4.3 register char* yy_cp

Definition at line 564 of file dbcompare_expression_scanner.cc.

11.63.4.4 YY_DECL register yy_state_type yy_current_state

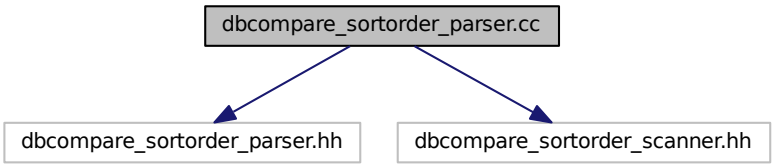
The main scanner function which does all the work.

Definition at line 563 of file dbcompare_expression_scanner.cc.

11.63.4.5 int yyleng

11.64 dbcompare_sortorder_parser.cc File Reference

`#include "dbcompare_sortorder_parser.hh"` `#include "dbcompare_sortorder_scanner.hh"` Include dependency graph for dbcompare_sortorder_parser.cc:



Namespaces

- namespace coco_api_internal

Defines

- #define yylex coco_api_internallex
- #define yylex scanner.yylex
- #define YY_(msgid) msgid
- #define YYUSE(e) ((void) (e))
- #define YYCDEBUG
- #define YY_SYMBOL_PRINT(Title, Type, Value, Location)
- #define YY_REDUCE_PRINT(Rule)
- #define YY_STACK_PRINT()
- #define YYACCEPT goto yyacceptlab
- #define YYABORT goto yyabortlab
- #define YYERROR goto yyerrorlab

11.64.1 Define Documentation

11.64.1.1 #define YY_(*msgid*) msgid

Definition at line 69 of file dbcompare_sortorder_parser.cc.

11.64.1.2 #define YY_REDUCE_PRINT(*Rule*)

Definition at line 109 of file dbcompare_sortorder_parser.cc.

11.64.1.3 #define YY_STACK_PRINT( )

Definition at line 110 of file dbcompare_sortorder_parser.cc.

11.64.1.4 #define YY_SYMBOL_PRINT(*Title, Type, Value, Location*)

Definition at line 108 of file dbcompare_sortorder_parser.cc.

11.64.1.5 #define YYABORT goto yyabortlab

Definition at line 115 of file dbcompare_sortorder_parser.cc.

11.64.1.6 #define YYACCEPT goto yyacceptlab

Definition at line 114 of file dbcompare_sortorder_parser.cc.

11.64.1.7 #define YYCDEBUG

Value:

```
for (bool yydebugcond_ = yydebug_; yydebugcond_; yydebugcond_ = false)  \
    (*yycdebug_)
```

Definition at line 77 of file dbcompare_sortorder_parser.cc.

11.64.1.8 #define YYERROR goto yyerrorlab

Definition at line 116 of file dbcompare_sortorder_parser.cc.

11.64.1.9 #define yylex coco_api_internallex

Definition at line 45 of file dbcompare_sortorder_parser.cc.

11.64.1.10 #define yylex scanner.yylex

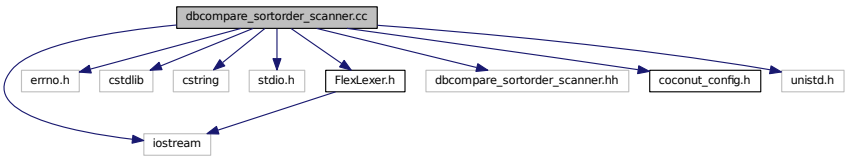
Definition at line 45 of file dbcompare_sortorder_parser.cc.

11.64.1.11 #define YYUSE(e) ((void) (e))

Definition at line 74 of file dbcompare_sortorder_parser.cc.

11.65 dbcompare_sortorder_scanner.cc File Reference

`#include <iostream>` `#include <errno.h>` `#include <cstdlib>` `#include <cstring>` × `#include <stdio.h>` `#include <FlexLexer.h>` `#include "dbcompare_sortorder_scanner.hh"` `#include <coconut_config.h>` `#include <unistd.h>` Include dependency graph for dbcompare_sortorder_scanner.cc:



Classes

- struct yy_buffer_state
- struct yy_trans_info

Defines

- #define YY_INT_ALIGNED short int
- #define FLEX_SCANNER
- #define YY_FLEX_MAJOR_VERSION 2
- #define YY_FLEX_MINOR_VERSION 5
- #define YY_FLEX_SUBMINOR_VERSION 31
- #define FLEX_BETA
- #define yyFlexLexer CoconutDBCompareSortOrderFlexLexer
- #define FLEXINT_H
- #define INT8_MIN (-128)
- #define INT16_MIN (-32767-1)
- #define INT32_MIN (-2147483647-1)
- #define INT8_MAX (127)
- #define INT16_MAX (32767)

- #define INT32_MAX (2147483647)
- #define UINT8_MAX (255U)
- #define UINT16_MAX (65535U)
- #define UINT32_MAX (4294967295U)
- #define yyconst
- #define YY_NULL 0
- #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)
- #define BEGIN (yy_start) = 1 + 2 *
- #define YY_START (((yy_start) - 1) / 2)
- #define YYSTATE YY_START
- #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
- #define YY_NEW_FILE yyrestart(yyin)
- #define YY_END_OF_BUFFER_CHAR 0
- #define YY_BUF_SIZE 16384
- #define YY_TYPEDEF_YY_BUFFER_STATE
- #define EOB_ACT_CONTINUE_SCAN 0
- #define EOB_ACT_END_OF_FILE 1
- #define EOB_ACT_LAST_MATCH 2
- #define YY_LESS_LINENO(n)
- #define yyless(n)
- #define unput(c) yyunput(c, (yytext_ptr))
- #define YY_TYPEDEF_YY_SIZE_T
- #define YY_STRUCT_YY_BUFFER_STATE
- #define YY_BUFFER_NEW 0
- #define YY_BUFFER_NORMAL 1
- #define YY_BUFFER_EOF_PENDING 2
- #define YY_CURRENT_BUFFER
- #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]
- #define yy_new_buffer yy_create_buffer
- #define yy_set_interactive(is_interactive)
- #define yy_set_bol(at_bol)
- #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
- #define yywrap() 1
- #define YY_SKIP_YYWRAP
- #define yytext_ptr yytext
- #define YY_DECL int dbcompare_sortorder_scanner::yylex()
- #define YY_DO_BEFORE_ACTION
- #define YY_NUM_RULES 11
- #define YY_END_OF_BUFFER 12
- #define REJECT reject_used_but_not_detected
- #define yymore() yymore_used_but_not_detected
- #define YY_MORE_ADJ 0
- #define YY_RESTORE_YY_MORE_OFFSET
- #define INITIAL 0
- #define YY_EXTRA_TYPE void *
- #define YY_READ_BUF_SIZE 8192
- #define ECHO LexerOutput(yytext, yyleng)
- #define YY_INPUT(buf, result, max_size)
- #define yyterminate() return YY_NULL
- #define YY_START_STACK_INCR 25

- #define YY_FATAL_ERROR(msg) LexerError(msg)
- #define YY_USER_ACTION
- #define YY_BREAK break;
- #define YY_RULE_SETUP YY_USER_ACTION
- #define YY_EXIT_FAILURE 2
- #define yyless(n)
- #define YYTABLES_NAME "yytables"

Typedefs

- typedef signed char flex_int8_t
- typedef short int flex_int16_t
- typedef int flex_int32_t
- typedef unsigned char flex_uint8_t
- typedef unsigned short int flex_uint16_t
- typedef unsigned int flex_uint32_t
- typedef struct yy_buffer_state * YY_BUFFER_STATE
- typedef unsigned int yy_size_t
- typedef unsigned char YY_CHAR
- typedef coco_api_internal::dbcompare_sortorder_parser::token token

Functions

- void * CoconutDBCompareSortOrderalloc (yy_size_t)
- void * CoconutDBCompareSortOrderrealloc (void *, yy_size_t)
- void CoconutDBCompareSortOrderfree (void *)

Variables

- int yyleng

11.65.1 Define Documentation

11.65.1.1 #define BEGIN (yy_start) = 1 + 2 *

Definition at line 126 of file dbcompare_sortorder_scanner.cc.

11.65.1.2 #define ECHO LexerOutput(yytext, yyleng)

Definition at line 499 of file dbcompare_sortorder_scanner.cc.

11.65.1.3 #define EOB_ACT_CONTINUE_SCAN 0

Definition at line 155 of file dbcompare_sortorder_scanner.cc.

11.65.1.4 #define EOB_ACT_END_OF_FILE 1

Definition at line 156 of file dbcompare_sortorder_scanner.cc.

11.65.1.5 #define EOB_ACT_LAST_MATCH 2

Definition at line 157 of file dbcompare_sortorder_scanner.cc.

11.65.1.6 #define FLEX_BETA

Definition at line 14 of file dbcompare_sortorder_scanner.cc.

11.65.1.7 #define FLEX_SCANNER

Definition at line 9 of file dbcompare_sortorder_scanner.cc.

11.65.1.8 #define FLEXINT_H

Definition at line 31 of file dbcompare_sortorder_scanner.cc.

11.65.1.9 #define INITIAL 0

Definition at line 466 of file dbcompare_sortorder_scanner.cc.

11.65.1.10 #define INT16_MAX (32767)

Definition at line 66 of file dbcompare_sortorder_scanner.cc.

11.65.1.11 #define INT16_MIN (-32767-1)

Definition at line 57 of file dbcompare_sortorder_scanner.cc.

11.65.1.12 #define INT32_MAX (2147483647)

Definition at line 69 of file dbcompare_sortorder_scanner.cc.

11.65.1.13 #define INT32_MIN (-2147483647-1)

Definition at line 60 of file dbcompare_sortorder_scanner.cc.

11.65.1.14 #define INT8_MAX (127)

Definition at line 63 of file dbcompare_sortorder_scanner.cc.

11.65.1.15 #define INT8_MIN (-128)

Definition at line 54 of file dbcompare_sortorder_scanner.cc.

11.65.1.16 #define REJECT reject_used_but_not_detected

Definition at line 426 of file dbcompare_sortorder_scanner.cc.

11.65.1.17 #define UINT16_MAX (65535U)

Definition at line 75 of file dbcompare_sortorder_scanner.cc.

11.65.1.18 #define UINT32_MAX (4294967295U)

Definition at line 78 of file dbcompare_sortorder_scanner.cc.

11.65.1.19 #define UINT8_MAX (255U)

Definition at line 72 of file dbcompare_sortorder_scanner.cc.

11.65.1.20 #define unput(*c*) yyunput(c, (yytext_ptr))

Definition at line 175 of file dbcompare_sortorder_scanner.cc.

11.65.1.21 #define YY_AT_BOL( ) (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)

Definition at line 294 of file dbcompare_sortorder_scanner.cc.

11.65.1.22 #define YY_BREAK break;

Definition at line 550 of file dbcompare_sortorder_scanner.cc.

11.65.1.23 #define YY_BUF_SIZE 16384

Definition at line 145 of file dbcompare_sortorder_scanner.cc.

11.65.1.24 #define YY_BUFFER_EOF_PENDING 2

Definition at line 248 of file dbcompare_sortorder_scanner.cc.

11.65.1.25 #define YY_BUFFER_NEW 0

Definition at line 236 of file dbcompare_sortorder_scanner.cc.

11.65.1.26 #define YY_BUFFER_NORMAL 1

Definition at line 237 of file dbcompare_sortorder_scanner.cc.

11.65.1.27 #define YY_CURRENT_BUFFER

Value:

```
( (yy_buffer_stack) \
                          ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
                          : NULL)
```

Definition at line 259 of file dbcompare_sortorder_scanner.cc.

11.65.1.28 #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]

Definition at line 266 of file dbcompare_sortorder_scanner.cc.

11.65.1.29 #define YY_DECL int dbcompare_sortorder_scanner::yylex()

Definition at line 310 of file dbcompare_sortorder_scanner.cc.

11.65.1.30 #define YY_DO_BEFORE_ACTION

Value:

```
(yytext_ptr) = yy_bp; \
        yyleng = (size_t) (yy_cp - yy_bp); \
        (yy_hold_char) = *yy_cp; \
        *yy_cp = '\0'; \
        (yy_c_buf_p) = yy_cp;
```

Definition at line 315 of file dbcompare_sortorder_scanner.cc.

11.65.1.31 #define YY_END_OF_BUFFER 12

Definition at line 323 of file dbcompare_sortorder_scanner.cc.

11.65.1.32 #define YY_END_OF_BUFFER_CHAR 0

Definition at line 141 of file dbcompare_sortorder_scanner.cc.

11.65.1.33 #define YY_EXIT_FAILURE 2

11.65.1.34 #define YY_EXTRA_TYPE void *

Definition at line 477 of file dbcompare_sortorder_scanner.cc.

11.65.1.35 #define YY_FATAL_ERROR(*msg*) LexerError(msg)

Definition at line 528 of file dbcompare_sortorder_scanner.cc.

11.65.1.36 #define YY_FLEX_MAJOR_VERSION 2

Definition at line 10 of file dbcompare_sortorder_scanner.cc.

11.65.1.37 #define YY_FLEX_MINOR_VERSION 5

Definition at line 11 of file dbcompare_sortorder_scanner.cc.

11.65.1.38 #define YY_FLEX_SUBMINOR_VERSION 31

Definition at line 12 of file dbcompare_sortorder_scanner.cc.

11.65.1.39 #define YY_INPUT(*buf, result, max_size*)

Value:

```
\
        if ( (result = LexerInput( (char *) buf, max_size )) < 0 ) \
                YY_FATAL_ERROR( "input in flex scanner failed" );
```

Definition at line 506 of file dbcompare_sortorder_scanner.cc.

11.65.1.40 #define YY_INT_ALIGNED short int

Definition at line 5 of file dbcompare_sortorder_scanner.cc.

11.65.1.41 #define YY_LESS_LINENO(*n*)

Definition at line 159 of file dbcompare_sortorder_scanner.cc.

11.65.1.42 #define YY_MORE_ADJ 0

Definition at line 428 of file dbcompare_sortorder_scanner.cc.

11.65.1.43 #define yy_new_buffer yy_create_buffer

Definition at line 272 of file dbcompare_sortorder_scanner.cc.

11.65.1.44 #define YY_NEW_FILE yyrestart(yyin)

Definition at line 139 of file dbcompare_sortorder_scanner.cc.

11.65.1.45 #define YY_NULL 0

Definition at line 113 of file dbcompare_sortorder_scanner.cc.

11.65.1.46 #define YY_NUM_RULES 11

Definition at line 322 of file dbcompare_sortorder_scanner.cc.

11.65.1.47 #define YY_READ_BUF_SIZE 8192

Definition at line 494 of file dbcompare_sortorder_scanner.cc.

11.65.1.48 #define YY_RESTORE_YY_MORE_OFFSET

Definition at line 429 of file dbcompare_sortorder_scanner.cc.

11.65.1.49 #define YY_RULE_SETUP YY_USER_ACTION

Definition at line 553 of file dbcompare_sortorder_scanner.cc.

11.65.1.50 #define YY_SC_TO_UI(*c*) ((unsigned int) (unsigned char) c)

Definition at line 120 of file dbcompare_sortorder_scanner.cc.

11.65.1.51 #define yy_set_bol(*at_bol*)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){\
        yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
            yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
        }
```

Definition at line 284 of file dbcompare_sortorder_scanner.cc.

11.65.1.52 #define yy_set_interactive(*is_interactive*)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){ \
        yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
            yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
        }
```

Definition at line 274 of file dbcompare_sortorder_scanner.cc.

11.65.1.53 #define YY_SKIP_YYWRAP

Definition at line 297 of file dbcompare_sortorder_scanner.cc.

11.65.1.54 #define YY_START (((yy_start) - 1) / 2)

Definition at line 132 of file dbcompare_sortorder_scanner.cc.

11.65.1.55 #define YY_START_STACK_INCR 25

Definition at line 523 of file dbcompare_sortorder_scanner.cc.

11.65.1.56 #define YY_STATE_EOF(*state*) (YY_END_OF_BUFFER + state + 1)

Definition at line 136 of file dbcompare_sortorder_scanner.cc.

11.65.1.57 #define YY_STRUCT_YY_BUFFER_STATE

Definition at line 188 of file dbcompare_sortorder_scanner.cc.

11.65.1.58 #define YY_TYPEDEF_YY_BUFFER_STATE

Definition at line 149 of file dbcompare_sortorder_scanner.cc.

11.65.1.59 #define YY_TYPEDEF_YY_SIZE_T

Definition at line 183 of file dbcompare_sortorder_scanner.cc.

11.65.1.60 #define YY_USER_ACTION

Definition at line 545 of file dbcompare_sortorder_scanner.cc.

11.65.1.61 #define yyconst

Definition at line 107 of file dbcompare_sortorder_scanner.cc.

11.65.1.62 #define yyFlexLexer CoconutDBCompareSortOrderFlexLexer

Definition at line 20 of file dbcompare_sortorder_scanner.cc.

11.65.1.63 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                *yy_cp = (yy_hold_char); \
                YY_RESTORE_YY_MORE_OFFSET \
                (yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ;
      \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
```

Definition at line 162 of file dbcompare_sortorder_scanner.cc.

11.65.1.64 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                yytext[yyleng] = (yy_hold_char); \
                (yy_c_buf_p) = yytext + yyless_macro_arg; \
                (yy_hold_char) = *(yy_c_buf_p); \
                *(yy_c_buf_p) = '\0'; \
                yyleng = yyless_macro_arg; \
                } \
        while ( 0 )
```

Definition at line 162 of file dbcompare_sortorder_scanner.cc.

11.65.1.65 #define yymore( ) yymore_used_but_not_detected

Definition at line 427 of file dbcompare_sortorder_scanner.cc.

11.65.1.66 #define YYSTATE YY_START

Definition at line 133 of file dbcompare_sortorder_scanner.cc.

11.65.1.67 #define YYTABLES_NAME "yytables"

11.65.1.68 #define yyterminate( ) return YY_NULL

Definition at line 518 of file dbcompare_sortorder_scanner.cc.

11.65.1.69 #define yytext_ptr yytext

Definition at line 301 of file dbcompare_sortorder_scanner.cc.

11.65.1.70 #define yywrap( ) 1

Definition at line 296 of file dbcompare_sortorder_scanner.cc.

11.65.2 Typedef Documentation

11.65.2.1 typedef short int flex_int16_t

Definition at line 45 of file dbcompare_sortorder_scanner.cc.

11.65.2.2 typedef int flex_int32_t

Definition at line 46 of file dbcompare_sortorder_scanner.cc.

11.65.2.3 typedef signed char flex_int8_t

Definition at line 44 of file dbcompare_sortorder_scanner.cc.

11.65.2.4 typedef unsigned short int flex_uint16_t

Definition at line 48 of file dbcompare_sortorder_scanner.cc.

11.65.2.5 typedef unsigned int flex_uint32_t

Definition at line 49 of file dbcompare_sortorder_scanner.cc.

11.65.2.6 typedef unsigned char flex_uint8_t

Definition at line 47 of file dbcompare_sortorder_scanner.cc.

11.65.2.7 typedef coco_api_internal::dbcompare_sortorder_parser::token token

Definition at line 459 of file dbcompare_sortorder_scanner.cc.

11.65.2.8 typedef struct yy_buffer_state* YY_BUFFER_STATE

Definition at line 150 of file dbcompare_sortorder_scanner.cc.

11.65.2.9 typedef unsigned char YY_CHAR

Definition at line 299 of file dbcompare_sortorder_scanner.cc.

11.65.2.10 typedef unsigned int yy_size_t

Definition at line 184 of file dbcompare_sortorder_scanner.cc.

11.65.3 Function Documentation

11.65.3.1 void* CoconutDBCompareSortOrderalloc (yy_size_t)

11.65.3.2 void CoconutDBCompareSortOrderfree (void *)

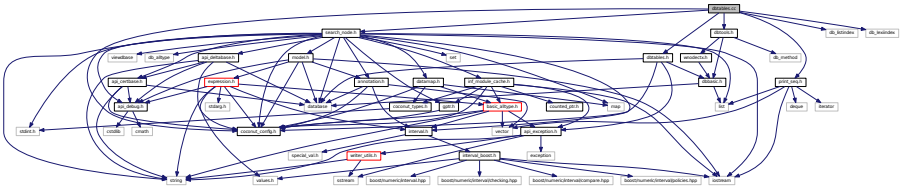
11.65.3.3 void* CoconutDBCompareSortOrderrealloc (void *, yy_size_t)

11.65.4 Variable Documentation

11.65.4.1 int yyleng

11.66 dbtables.cc File Reference

`#include <coconut_config.h>` `#include <search_node.h>` `#include <dbtools.-h>` `#include <dbtables.h>` `#include <print_seq.h>` `#include <db_listindex>` × `#include <db_lexiindex>` Include dependency graph for dbtables.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

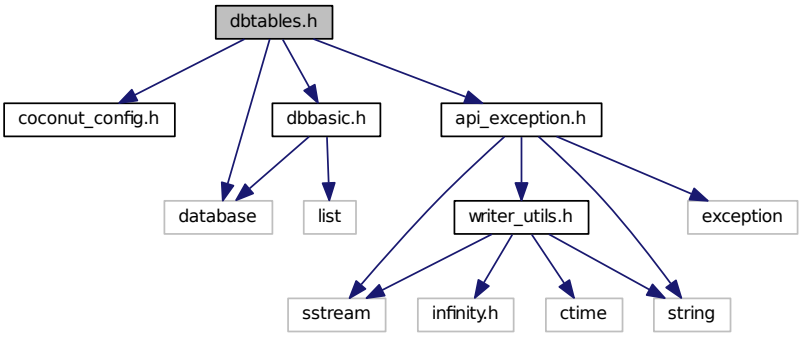
- tableid coco::get_point_table (database ∗ptr, standard_table ∗&ptb, userid _uid)

11.66.1 Detailed Description

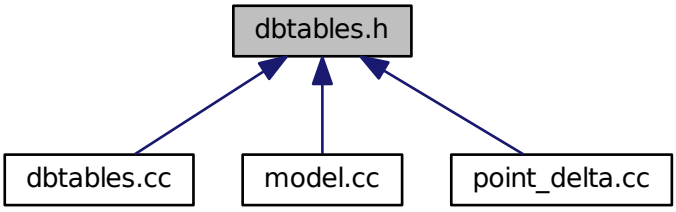
Definition in file dbtables.cc.

11.67 dbtables.h File Reference

`#include <coconut_config.h> #include <database> #include <dbbasic.h>` ×
`#include <api_exception.h>` Include dependency graph for dbtables.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- vdbl::tableid coco::get_point_table (vdbl::database ∗ptr, vdbl::standard_table ∗&ptb, vdbl::userid)

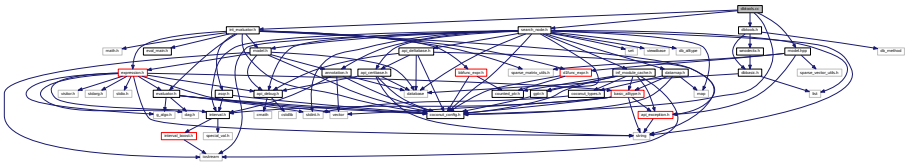
  Get the point from the search database.

11.67.1 Detailed Description

Definition in file dbtables.h.

11.68 dbtools.cc File Reference

`#include <coconut_config.h>` `#include <search_node.h>` `#include <dbtools.-h>` `#include <int_evaluator.h>` `#include <model.hpp>` Include dependency graph for dbtools.cc:



Namespaces

- namespace coco

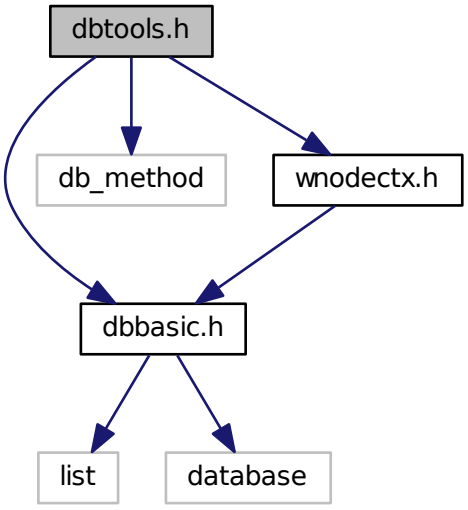
  the main namespace of the COCONUT API

11.68.1 Detailed Description

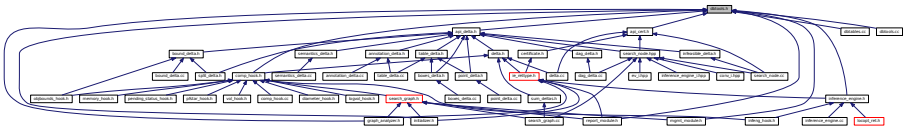
Definition in file dbtools.cc.

11.69 dbtools.h File Reference

`#include <dbbasic.h>` `#include <db_method>` `#include <wnodectx.h>` Include dependency graph for dbtools.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::point_check_feasibility

  Stored procedure checking the feasibility of a point.
- class coco::box_check_intersection

  Stored procedure checking whether a box intersects the work_node's box.

Namespaces

- namespace coco

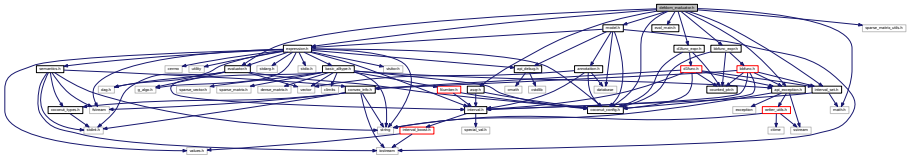
  the main namespace of the COCONUT API

11.69.1 Detailed Description

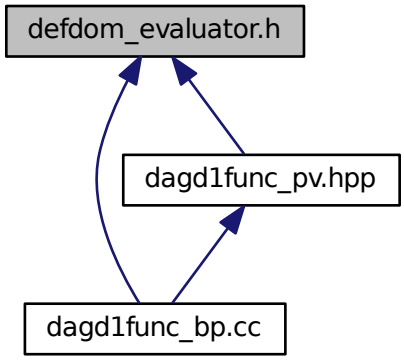
Definition in file dbtools.h.

11.70 defdom_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <math.h>` `#include <api_exception.h>` `#include <d1func_expr.h>` `#include <bbfunc_expr.h>` `#include <asqr.h>` `#include <sparse_matrix_utils.h>` `#include <interval_set.h>` Include dependency graph for defdom_evaluator.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct coco::defdom_ret_type

  Return type for defdom_eval.
- struct coco::defdom_problem_point
- struct coco::defdom_eval_type

Visitor data for defdom_eval.

- class coco::defdom_eval

  Forward function range evaluation.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

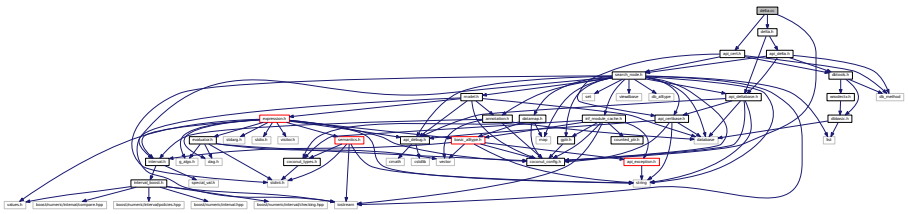
- typedef std::multimap < unsigned int, defdom_problem_point > coco::defdom_map
- typedef defdom_ret_type(∗ coco::defdom_evaluator)(const std::vector< interval > ∗__x, defdom_map &__m, const variable_indicator &__v)

11.70.1 Detailed Description

Definition in file defdom_evaluator.h.

11.71 delta.cc File Reference

`#include <coconut_config.h> #include <delta.h> #include <api_cert.h>` ×
Include dependency graph for delta.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

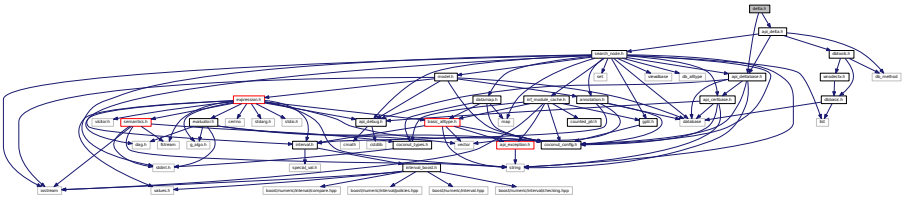
11.71.1 Detailed Description

This is an internal header file, which is not intended for use outside the API.

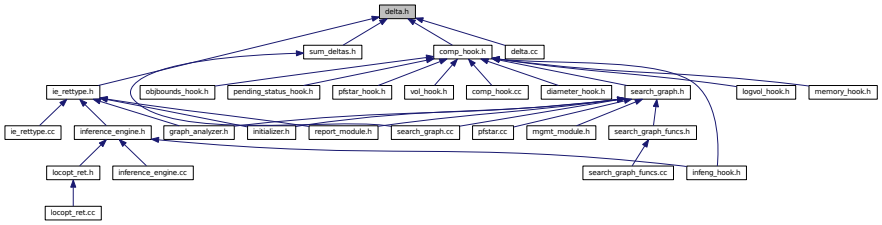
Definition in file delta.cc.

11.72 delta.h File Reference

`#include <api_deltabase.h>` `#include <api_delta.h>` Include dependency graph for delta.h:



This graph shows which files directly or indirectly include this file:

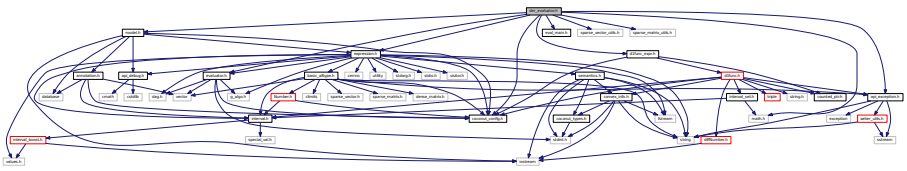


11.72.1 Detailed Description

Definition in file delta.h.

11.73 der_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <sparse_vector_-utils.h>` `#include <sparse_matrix_utils.h>` `#include <math.h>` `#include <d1func-_expr.h>` `#include <api_exception.h>` Include dependency graph for der_evaluator.h:



Classes

- class coco::prep_d_eval

  Preparation Evaluator for derivatives.
- struct coco::func_d_eval_type

  Visitor data for func_d_eval.
- class coco::func_d_eval

  Forward function evaluation with preparation of derivative data.
- struct coco::der_eval_type

  Visitor data for der_eval.
- class coco::der_eval

  Backward gradient evaluation with prepared derivative data.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

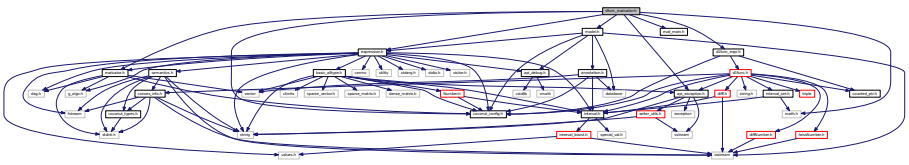
- typedef bool(∗ coco::prep_d_evaluator)()
- typedef double(∗ coco::func_d_evaluator)(const std::vector< double > ∗__x, const variable_indicator &__v, std::vector< double > &__d_data)
- typedef std::vector< double > &(∗ coco::der_evaluator)(const std::vector< double > &__d_dat, const variable_indicator &__v)

11.73.1 Detailed Description

Definition in file der_evaluator.h.

11.74 dfunc_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <math.h>` `#include <d1func_expr.h>` `#include <api_exception.h>` Include dependency graph for dfunc_evaluator.h-:



Classes

- struct coco::dfunc_eval_rettype
- struct coco::dfunc_eval_type
- class coco::dfunc_eval

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

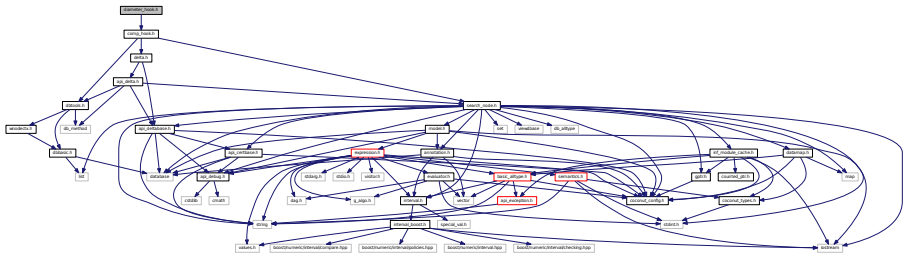
- typedef double(∗ coco::func_evaluator)(const std::vector< double > ∗__x, const variable_indicator &__v)

11.74.1 Detailed Description

Definition in file dfunc_evaluator.h.

11.75 diameter_hook.h File Reference

`#include <comp_hook.h>` Include dependency graph for diameter_hook.h:



Classes

- class coco::diameter_comp_hook

  The log-volume computation hook (work node computation hook)

Namespaces

- namespace coco

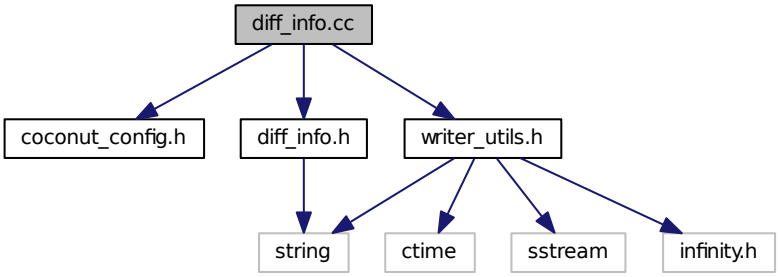
  the main namespace of the COCONUT API

11.75.1 Detailed Description

Definition in file diameter_hook.h.

11.76 diff_info.cc File Reference

`#include <coconut_config.h>` `#include <diff_info.h>` `#include <writer_utils.-h>` Include dependency graph for diff_info.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

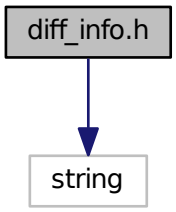
- std::string coco::get_diff_class (int i)

11.76.1 Detailed Description

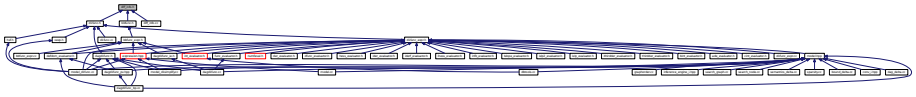
Definition in file diff_info.cc.

11.77 diff_info.h File Reference

`#include <string>` Include dependency graph for diff_info.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DIFF_EVAL_ANALYTIC -4
- #define DIFF_EVAL_C_INFINITY -3
- #define DIFF_EVAL_DISCONT -2
- #define DIFF_EVAL_PART_UNDEF -1

Functions

- std::string coco::get_diff_class (int i)

11.77.1 Detailed Description

Definition in file diff_info.h.

11.77.2 Define Documentation

11.77.2.1 #define DIFF_EVAL_ANALYTIC -4

Definition at line 35 of file diff_info.h.

11.77.2.2 #define DIFF_EVAL_C_INFINITY -3

Definition at line 36 of file diff_info.h.

11.77.2.3 #define DIFF_EVAL_DISCONT -2

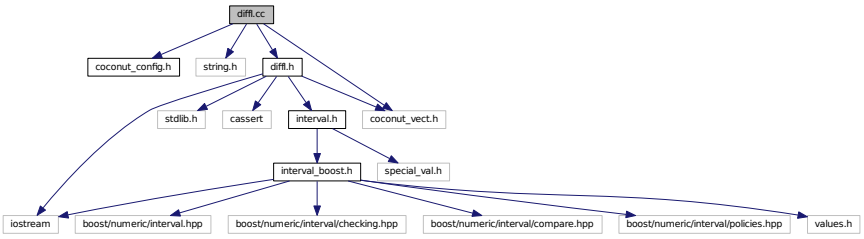
Definition at line 37 of file diff_info.h.

11.77.2.4 #define DIFF_EVAL_PART_UNDEF -1

Definition at line 38 of file diff_info.h.

11.78 diffI.cc File Reference

`#include <coconut_config.h>` `#include <string.h>` `#include <diffI.h>` `#include <coconut_vect.h>` Include dependency graph for diffI.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DEBUG_EVAL 0

Functions

- diffI coco::operator+ (const diffI &a, const diffI &b)
- diffI coco::operator+ (const double &a, const diffI &b)

- diffI coco::operator+ (const diffI &a, const double &b)
- diffI coco::operator+ (const interval &a, const diffI &b)
- diffI coco::operator+ (const diffI &a, const interval &b)
- diffI coco::operator- (const diffI &a, const diffI &b)
- diffI coco::operator- (const double &a, const diffI &b)
- diffI coco::operator- (const diffI &a, const double &b)
- diffI coco::operator- (const interval &a, const diffI &b)
- diffI coco::operator- (const diffI &a, const interval &b)
- diffI coco::operator- (const diffI &a)
- diffI coco::operator∗ (const double &a, const diffI &b)
- diffI coco::operator∗ (const diffI &a, const double &b)
- diffI coco::operator/ (const diffI &a, const double &b)
- diffI coco::operator∗ (const interval &a, const diffI &b)
- diffI coco::operator∗ (const diffI &a, const interval &b)
- diffI coco::operator/ (const diffI &a, const interval &b)
- diffI coco::operator∗ (const diffI &a, const diffI &b)
- diffI coco::operator/ (const diffI &a, const diffI &b)
- diffI coco::operator/ (const double &a, const diffI &b)
- diffI coco::operator/ (const interval &a, const diffI &b)
- diffI coco::exp (const diffI &a)
- diffI coco::sin (const diffI &a)
- diffI coco::cos (const diffI &a)
- diffI coco::sqrt (const diffI &a)
- diffI coco::abs (const diffI &a)
- diffI coco::power (const diffI &a, int n)
- diffI coco::pow (const diffI &a, const diffI &b)
- diffI coco::max (const diffI &a, const diffI &b)
- diffI coco::min (const diffI &a, const diffI &b)
- diffI coco::log (const diffI &a)
- diffI coco::sinh (const diffI &a)
- diffI coco::cosh (const diffI &a)
- diffI coco::atan2 (const diffI &a, const diffI &b)
- diffI coco::atan (const diffI &a)
- diffI coco::sqr (const diffI &a)
- std::ostream & coco::operator<< (std::ostream &s, const diffI &a)

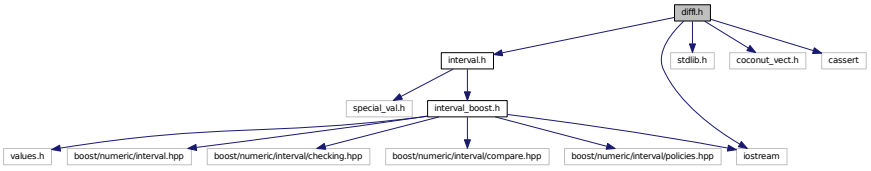
11.78.1 Define Documentation

11.78.1.1 #define DEBUG_EVAL 0

Definition at line 12 of file diffI.cc.

11.79 diffI.h File Reference

`#include "interval.h"` `#include <stdlib.h>` `#include <iostream>` `#include <coconut-_vect.h>` `#include <cassert>` Include dependency graph for diffI.h:



This graph shows which files directly or indirectly include this file:

Classes

- class coco::diffI

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- diffI coco::operator+ (const diffI &a, const diffI &b)
- diffI coco::operator+ (const double &a, const diffI &b)
- diffI coco::operator+ (const diffI &a, const double &b)
- diffI coco::operator+ (const interval &a, const diffI &b)
- diffI coco::operator+ (const diffI &a, const interval &b)
- diffI coco::operator- (const diffI &a, const diffI &b)
- diffI coco::operator- (const double &a, const diffI &b)
- diffI coco::operator- (const diffI &a, const double &b)
- diffI coco::operator- (const interval &a, const diffI &b)
- diffI coco::operator- (const diffI &a, const interval &b)
- diffI coco::operator- (const diffI &a)
- diffI coco::operator∗ (const diffI &a, const diffI &b)
- diffI coco::operator∗ (const double &a, const diffI &b)
- diffI coco::operator∗ (const diffI &a, const double &b)

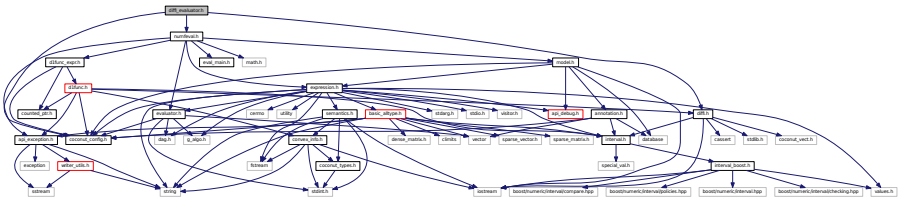
- diffI coco::operator∗ (const interval &a, const diffI &b)
- diffI coco::operator∗ (const diffI &a, const interval &b)
- diffI coco::operator/ (const diffI &a, const diffI &b)
- diffI coco::operator/ (const double &a, const diffI &b)
- diffI coco::operator/ (const diffI &a, const double &b)
- diffI coco::operator/ (const interval &a, const diffI &b)
- diffI coco::operator/ (const diffI &a, const interval &b)
- std::ostream & coco::operator<< (std::ostream &s, const diffI &a)
- diffI coco::exp (const diffI &a)
- diffI coco::sin (const diffI &a)
- diffI coco::cos (const diffI &a)
- diffI coco::sqrt (const diffI &a)
- diffI coco::abs (const diffI &a)
- diffI coco::power (const diffI &a, int n)
- diffI coco::pow (const diffI &a, const diffI &b)
- diffI coco::max (const diffI &a, const diffI &b)
- diffI coco::min (const diffI &a, const diffI &b)
- diffI coco::log (const diffI &a)
- diffI coco::sinh (const diffI &a)
- diffI coco::cosh (const diffI &a)
- diffI coco::atan2 (const diffI &a, const diffI &b)
- diffI coco::atan (const diffI &a)
- diffI coco::sqr (const diffI &a)

Variables

- const int coco::STANDARD_MAXDEG_DIFFINTERVAL = 4

11.80 diffI_evaluator.h File Reference

`#include <coconut_config.h> #include <diffI.h> #include <numfeval.h>` ×
Include dependency graph for diffI_evaluator.h:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define xxxNumber_t diffI
- #define xxxNumber_name xxxNumber_t
- #define XXXNUMBER_HAS_DEGREE 1
- #define XXXNUMBER_HAS_1D 1
- #define XXXNUMBER_HAS_SQR 1
- #define XXXEVAL_HAS_BASES 0
- #define XXXNUMBER_EVAL_DEBUG 0

11.80.1 Detailed Description

Definition in file diffI_evaluator.h.

11.80.2 Define Documentation

11.80.2.1 #define XXXEVAL_HAS_BASES 0

Definition at line 65 of file diffI_evaluator.h.

11.80.2.2 #define XXXNUMBER_EVAL_DEBUG 0

Definition at line 70 of file diffI_evaluator.h.

11.80.2.3 #define XXXNUMBER_HAS_1D 1

Definition at line 55 of file diffI_evaluator.h.

11.80.2.4 #define XXXNUMBER_HAS_DEGREE 1

Definition at line 50 of file diffI_evaluator.h.

11.80.2.5 #define XXXNUMBER_HAS_SQR 1

Definition at line 60 of file diffI_evaluator.h.

11.80.2.6 #define xxxNumber_name xxxNumber_t

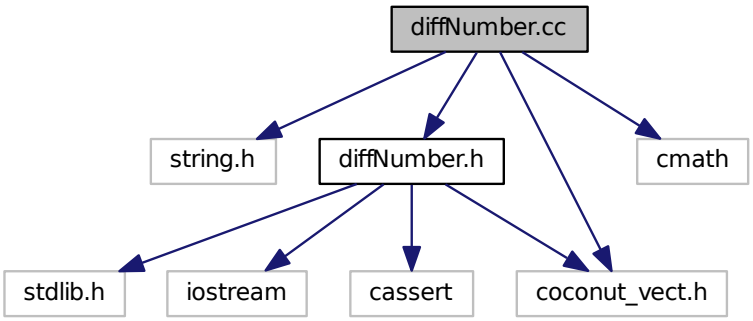
Definition at line 45 of file diffI_evaluator.h.

11.80.2.7 #define xxxNumber_t diffI

Definition at line 40 of file diffI_evaluator.h.

11.81 diffNumber.cc File Reference

`#include <string.h>` `#include <diffNumber.h>` `#include <coconut_vect.h>` × `#include <cmath>` Include dependency graph for diffNumber.cc:



Namespaces

- namespace coco

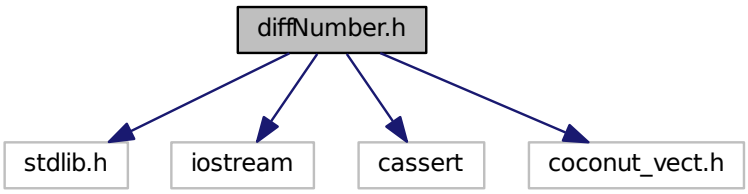
  the main namespace of the COCONUT API

Functions

- diffNumber coco::operator∗ (const diffNumber &a, const diffNumber &b)
- diffNumber coco::operator/ (const diffNumber &a, const diffNumber &b)
- diffNumber coco::operator/ (const double &a, const diffNumber &b)
- diffNumber coco::exp (const diffNumber &a)
- diffNumber coco::sin (const diffNumber &a)
- diffNumber coco::cos (const diffNumber &a)
- diffNumber coco::sqrt (const diffNumber &a)
- diffNumber coco::abs (const diffNumber &a)
- diffNumber coco::max (const diffNumber &a, const diffNumber &b)
- diffNumber coco::min (const diffNumber &a, const diffNumber &b)
- diffNumber coco::power (const diffNumber &a, int n)
- diffNumber coco::pow (const diffNumber &a, const diffNumber &b)
- diffNumber coco::log (const diffNumber &a)
- diffNumber coco::sinh (const diffNumber &a)
- diffNumber coco::cosh (const diffNumber &a)
- diffNumber coco::atan2 (const diffNumber &a, const diffNumber &b)
- diffNumber coco::atan (const diffNumber &a)
- diffNumber coco::square (const diffNumber &a)
- std::ostream & coco::operator<< (std::ostream &s, const diffNumber &a)

11.82 diffNumber.h File Reference

`#include <stdlib.h>` `#include <iostream>` `#include <cassert>` `#include <coconut-_vect.h>` Include dependency graph for diffNumber.h:



This graph shows which files directly or indirectly include this file:

Classes

- class coco::diffNumber

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- diffNumber coco::operator+ (const diffNumber &a, const diffNumber &b)
- diffNumber coco::operator+ (const double &a, const diffNumber &b)
- diffNumber coco::operator+ (const diffNumber &a, const double &b)
- diffNumber coco::operator- (const diffNumber &a, const diffNumber &b)
- diffNumber coco::operator- (const double &a, const diffNumber &b)
- diffNumber coco::operator- (const diffNumber &a, const double &b)
- diffNumber coco::operator- (const diffNumber &a)
- diffNumber coco::operator∗ (const diffNumber &a, const diffNumber &b)
- diffNumber coco::operator∗ (const double &a, const diffNumber &b)

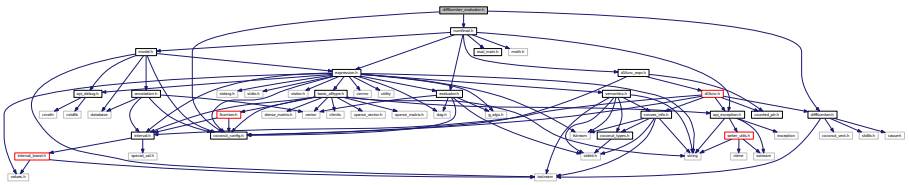
- diffNumber coco::operator∗ (const diffNumber &a, const double &b)
- diffNumber coco::operator/ (const diffNumber &a, const diffNumber &b)
- diffNumber coco::operator/ (const diffNumber &a, const double &b)
- diffNumber coco::operator/ (const double &a, const diffNumber &b)
- std::ostream & coco::operator<< (std::ostream &s, const diffNumber &a)
- diffNumber coco::exp (const diffNumber &a)
- diffNumber coco::sin (const diffNumber &a)
- diffNumber coco::cos (const diffNumber &a)
- diffNumber coco::sqrt (const diffNumber &a)
- diffNumber coco::abs (const diffNumber &a)
- diffNumber coco::power (const diffNumber &a, int n)
- diffNumber coco::pow (const diffNumber &a, const diffNumber &b)
- diffNumber coco::max (const diffNumber &a, const diffNumber &b)
- diffNumber coco::min (const diffNumber &a, const diffNumber &b)
- diffNumber coco::log (const diffNumber &a)
- diffNumber coco::sinh (const diffNumber &a)
- diffNumber coco::cosh (const diffNumber &a)
- diffNumber coco::atan2 (const diffNumber &a, const diffNumber &b)
- diffNumber coco::atan (const diffNumber &a)
- diffNumber coco::square (const diffNumber &a)

Variables

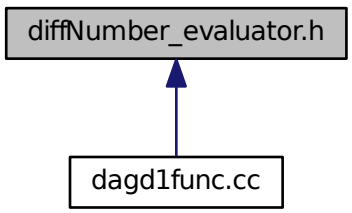
- const int coco::STANDARD_MAXDEG_DIFFNUMBER = 4

11.83 diffNumber_evaluator.h File Reference

`#include <coconut_config.h>` `#include <diffNumber.h>` `#include <numfeval.-h>` Include dependency graph for diffNumber_evaluator.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco
  the main namespace of the COCONUT API

Defines

- #define xxxNumber_t diffNumber
- #define xxxNumber_name xxxNumber_t
- #define XXXNUMBER_HAS_DEGREE 1
- #define XXXNUMBER_HAS_1D 0
- #define XXXNUMBER_HAS_SQR 0
- #define XXXEVAL_HAS_BASES 0
- #define XXXNUMBER_EVAL_DEBUG 0

11.83.1 Detailed Description

Definition in file diffNumber_evaluator.h.

11.83.2 Define Documentation

11.83.2.1 #define XXXEVAL_HAS_BASES 0

Definition at line 65 of file diffNumber_evaluator.h.

11.83.2.2 #define XXXNUMBER_EVAL_DEBUG 0

Definition at line 70 of file diffNumber_evaluator.h.

11.83.2.3 #define XXXNUMBER_HAS_1D 0

Definition at line 55 of file diffNumber_evaluator.h.

11.83.2.4 #define XXXNUMBER_HAS_DEGREE 1

Definition at line 50 of file diffNumber_evaluator.h.

11.83.2.5 #define XXXNUMBER_HAS_SQR 0

Definition at line 60 of file diffNumber_evaluator.h.

11.83.2.6 #define xxxNumber_name xxxNumber_t

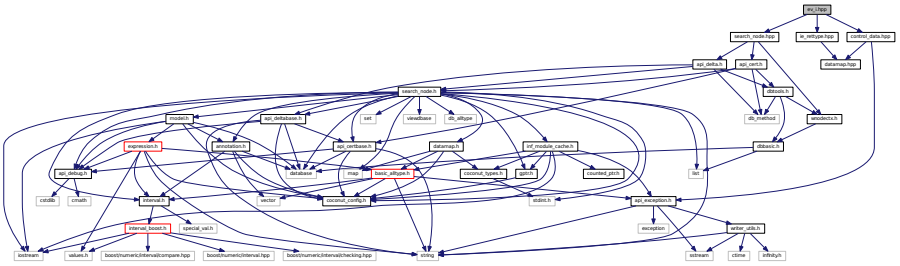
Definition at line 45 of file diffNumber_evaluator.h.

11.83.2.7 #define xxxNumber_t diffNumber

Definition at line 40 of file diffNumber_evaluator.h.

11.84 ev_i.hpp File Reference

`#include <control_data.hpp>` `#include <ie_rettype.hpp>` `#include <search_node.hpp>` Include dependency graph for ev_i.hpp:

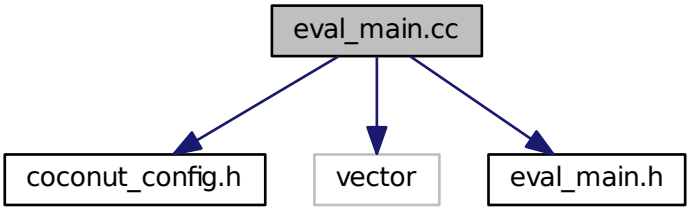


11.84.1 Detailed Description

Definition in file ev_i.hpp.

11.85 eval_main.cc File Reference

`#include <coconut_config.h> #include <vector> #include <eval_main.h>` ×
Include dependency graph for eval_main.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- std::vector< std::vector< void * > > coco::standard_evaluators (NUM_EVALUATORS)

11.85.1 Detailed Description

Definition in file eval_main.cc.

11.86 eval_main.h File Reference

This graph shows which files directly or indirectly include this file:

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define FUNC_EVALUATOR 0
- #define FUNC_D_EVALUATOR 1
- #define FUNC_RANGE 2
- #define FUNC_D_RANGE 3
- #define DER_EVALUATOR 4
- #define DER_RANGE 5
- #define REDUCE_RANGE 6
- #define PRINT 7
- #define FUNC_FW_PREP 8
- #define FUNC_ISLP_EVALUATOR 9
- #define ISLP_EVALUATOR 10
- #define FUNC_ID_EVALUATOR 11
- #define IDER_EVALUATOR 12
- #define FUNC_CRANGE 13
- #define FUNC_BRANGE 14
- #define FUNC_ANALYTICD 15
- #define FUNC_INFB 16
- #define CONVEX_EVALUATOR 17
- #define DIFF_EVALUATOR 18
- #define FUNC_IDER_FORWARD 19
- #define NUM_EVALUATORS 20

Variables

- std::vector< std::vector< void * > > coco::standard_evaluators

11.86.1 Detailed Description

Definition in file eval_main.h.

11.86.2 Define Documentation

11.86.2.1 #define CONVEX_EVALUATOR 17

function convexity evaluation

Definition at line 54 of file eval_main.h.

11.86.2.2 #define DER_EVALUATOR 4

derivative evaluation

Definition at line 39 of file eval_main.h.

11.86.2.3 #define DER_RANGE 5

derivative range evaluation

Definition at line 40 of file eval_main.h.

11.86.2.4 #define DIFF_EVALUATOR 18

function differentiability evaluation

Definition at line 55 of file eval_main.h.

11.86.2.5 #define FUNC_ANALYTICD 15

function analytic range

Definition at line 52 of file eval_main.h.

11.86.2.6 #define FUNC_BRANGE 14

function bounded range

Definition at line 51 of file eval_main.h.

11.86.2.7 #define FUNC_CRANGE 13

function complex range

Definition at line 50 of file eval_main.h.

11.86.2.8 #define FUNC_D_EVALUATOR 1

function evaluation with partial derivative

Definition at line 34 of file eval_main.h.

11.86.2.9 #define FUNC_D_RANGE 3

function range evaluation with partial derivative

Definition at line 37 of file eval_main.h.

11.86.2.10 #define FUNC_EVALUATOR 0

function evaluation

Definition at line 33 of file eval_main.h.

11.86.2.11 #define FUNC_FW_PREP 8

function evaluation

Definition at line 43 of file eval_main.h.

11.86.2.12 #define FUNC_ID_EVALUATOR 11

function range evaluation with interval derivative

Definition at line 47 of file eval_main.h.

11.86.2.13 #define FUNC_IDER_FORWARD 19

function range evaluation

Definition at line 57 of file eval_main.h.

11.86.2.14 #define FUNC_INFB 16

function bounds at infinity

Definition at line 53 of file eval_main.h.

11.86.2.15 #define FUNC_ISLP_EVALUATOR 9

function evaluation with partial slope

Definition at line 44 of file eval_main.h.

11.86.2.16 #define FUNC_RANGE 2

function range evaluation

Definition at line 36 of file eval_main.h.

11.86.2.17 #define IDER_EVALUATOR 12

interval derivative evaluation

Definition at line 49 of file eval_main.h.

11.86.2.18 #define ISLP_EVALUATOR 10

slope evaluation

Definition at line 46 of file eval_main.h.

11.86.2.19 #define NUM_EVALUATORS 20

number of evaluators for user defined nodes

Definition at line 59 of file eval_main.h.

11.86.2.20 #define PRINT 7

print function

Definition at line 42 of file eval_main.h.

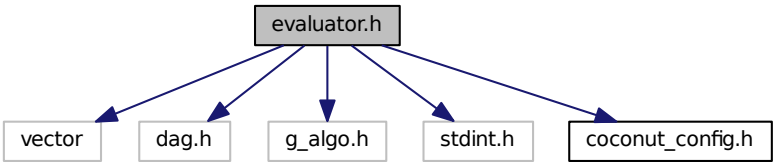
11.86.2.21 #define REDUCE_RANGE 6

function backwards reduction

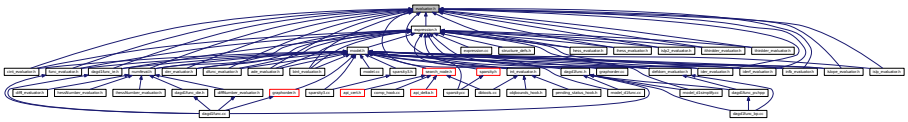
Definition at line 41 of file eval_main.h.

11.87 evaluator.h File Reference

`#include <vector>` `#include <dag.h>` `#include <g_algo.h>` `#include <stdint.-h>` `#include <coconut_config.h>` Include dependency graph for evaluator.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::variable_indicator

  Bitmap class used to indicate variable occurrence.
- class coco::_evaluator_base

  Base class of all evaluators.
- class coco::evaluator_base

  Base class of all (non-caching) evaluators.
- class coco::cached_evaluator_base

  Base class of all caching evaluators.
- class coco::forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class coco::backward_evaluator_base

  Base class of all (non-caching) backward evaluators.
- class coco::cached_forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class coco::cached_backward_evaluator_base

  Base class of all caching backward evaluators.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- template<class _Walker , class _Visitor >
  _Visitor::return_value coco::recursive_short_cut_walk (_Walker __w, _Visitor __f)

  Perform a recursive graph walk with possible caching and short-cuts.
- template<class _Walker , class _Visitor >
  _Visitor::return_value coco::_recursive_short_cut_walk (_Walker __w, _Visitor __f)

  Perform a recursive graph walk with possible caching and short-cuts (internal)
- template<class _Visitor , class _Walker >
  _Visitor::return_value coco::evaluate (_Visitor __v, _Walker __start)

  Evaluate an evaluator on a DAG.

11.87.1 Detailed Description

Definition in file evaluator.h.

11.88 expression.cc File Reference

`#include <coconut_config.h>` `#include <expression.h>` `#include <expression.-hpp>` `#include <print_seq.h>` `#include <print_matrix.h>` `#include <writer-_utils.h>` Include dependency graph for expression.cc:

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

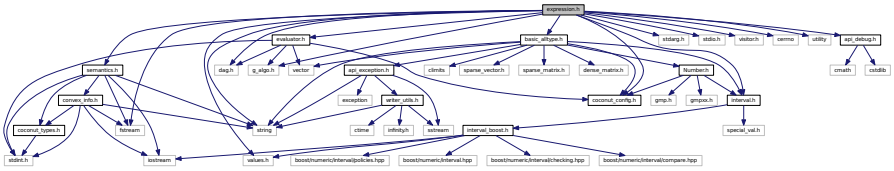
- std::ostream & coco::operator<< (std::ostream &o, const expression_node &__x)

11.88.1 Detailed Description

Definition in file expression.cc.

11.89 expression.h File Reference

`#include <coconut_config.h>` `#include <stdarg.h>` `#include <stdio.h>` `#include <interval.h>` `#include <dag.h>` `#include <visitor.h>` `#include <semantics.h>` `#include <evaluator.h>` `#include <basic_alltype.h>` `#include <g_algo.h>` `#include <fstream>` `#include <cerrno>` `#include <string>` `#include <utility>` `#include <values.h>` `#include <api_debug.h>` Include dependency graph for expression.h:



This graph shows which files directly or indirectly include this file:

Classes

- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- class coco::num::Number
- class coco::num::number_exception
- class coco::coco::api_exception

  API exception class.
- class coco::coco::nyi_exception

  Not Yet Implemented exception class.
- class coco::coco::basic_alltype

  The basic alltype which can hold any of a number of basic types.
- class coco::expression_node

  The base class for a node in the expression DAGs.

Namespaces

- namespace coco

  the main namespace of the COCONUT API
- namespace coco::coco
- namespace coco::num

Defines

- #define EXPR_LASTARG NULL
- #define EXPRINFO_GHOST 0
- #define EXPRINFO_CONSTANT -1
- #define EXPRINFO_VARIABLE -2
- #define EXPRINFO_SUM -3
- #define EXPRINFO_MEAN -4
- #define EXPRINFO_PROD -5
- #define EXPRINFO_MAX -6
- #define EXPRINFO_MIN -7
- #define EXPRINFO_MONOME -8
- #define EXPRINFO_SCPROD -9
- #define EXPRINFO_NORM -10
- #define EXPRINFO_INVERT -11
- #define EXPRINFO_SQUARE -12
- #define EXPRINFO_SQROOT -13
- #define EXPRINFO_ABS -14
- #define EXPRINFO_INTPOWER -15
- #define EXPRINFO_EXP -16
- #define EXPRINFO_LOG -17
- #define EXPRINFO_SIN -18
- #define EXPRINFO_COS -19
- #define EXPRINFO_SINH -20
- #define EXPRINFO_COSH -21
- #define EXPRINFO_GAUSS -22
- #define EXPRINFO_POLY -23
- #define EXPRINFO_POW -24
- #define EXPRINFO_DIV -25
- #define EXPRINFO_ATAN2 -26
- #define EXPRINFO_LIN -27
- #define EXPRINFO_QUAD -28
- #define EXPRINFO_IQUAD -29
- #define EXPRINFO_WSUMSQR -30
- #define EXPRINFO_LINSQR -31
- #define EXPRINFO_RE -32
- #define EXPRINFO_IM -33
- #define EXPRINFO_ARG -34
- #define EXPRINFO_CPLXCONJ -35
- #define EXPRINFO_LOOKUP -36
- #define EXPRINFO_PWLIN -37
- #define EXPRINFO_SPLINE -38
- #define EXPRINFO_PWCONSTLC -39

- #define EXPRINFO_PWCONSTRC -40
- #define EXPRINFO_IN -41
- #define EXPRINFO_IF -42
- #define EXPRINFO_AND -43
- #define EXPRINFO_OR -44
- #define EXPRINFO_NOT -45
- #define EXPRINFO_IMPLIES -46
- #define EXPRINFO_COUNT -47
- #define EXPRINFO_ALLDIFF -48
- #define EXPRINFO_HISTOGRAM -49
- #define EXPRINFO_LEVEL -50
- #define EXPRINFO_NEIGHBOR -51
- #define EXPRINFO_NOGOOD -52
- #define EXPRINFO_EXPECTATION -53
- #define EXPRINFO_INTEGRAL -54
- #define EXPRINFO_DET -55
- #define EXPRINFO_COND -56
- #define EXPRINFO_PSD -57
- #define EXPRINFO_MPROD -58
- #define EXPRINFO_FEM -59
- #define EXPRINFO_CMPROD -60
- #define EXPRINFO_CGFEM -61
- #define EXPRINFO_ASQR -62
- #define EXPRINFO_D1FUNC -63
- #define EXPRINFO_BLACKBOX -64
- #define EXPRINFO_UNDEFINED -65
- #define EXPRINFO_NUMOFPREDEF -(EXPRINFO_UNDEFINED)
- #define EXPR_LASTARG NULL

Operator types of standard expressions

These definitions define the standard operator types for the expression_node class. In the operator descriptions y_i is used for the value of the ith child of the node, while $x_i = c_i y_i$ stands for the value of the ith child of the node multiplied by the ith entry of the `coeffs` *array.*

- #define EXPRINFO_GHOST 0
- #define EXPRINFO_CONSTANT -1
- #define EXPRINFO_VARIABLE -2
- #define EXPRINFO_SUM -3
- #define EXPRINFO_MEAN -4
- #define EXPRINFO_PROD -5
- #define EXPRINFO_MAX -6
- #define EXPRINFO_MIN -7
- #define EXPRINFO_MONOME -8
- #define EXPRINFO_SCPROD -9
- #define EXPRINFO_NORM -10
- #define EXPRINFO_INVERT -11
- #define EXPRINFO_SQUARE -12
- #define EXPRINFO_SQROOT -13
- #define EXPRINFO_ABS -14
- #define EXPRINFO_INTPOWER -15
- #define EXPRINFO_EXP -16
- #define EXPRINFO_LOG -17
- #define EXPRINFO_SIN -18

- #define EXPRINFO_COS -19
- #define EXPRINFO_SINH -20
- #define EXPRINFO_COSH -21
- #define EXPRINFO_GAUSS -22
- #define EXPRINFO_POLY -23
- #define EXPRINFO_POW -24
- #define EXPRINFO_DIV -25
- #define EXPRINFO_ATAN2 -26
- #define EXPRINFO_LIN -27
- #define EXPRINFO_QUAD -28
- #define EXPRINFO_IQUAD -29
- #define EXPRINFO_WSUMSQR -30
- #define EXPRINFO_LINSQR -31
- #define EXPRINFO_RE -32
- #define EXPRINFO_IM -33
- #define EXPRINFO_ARG -34
- #define EXPRINFO_CPLXCONJ -35
- #define EXPRINFO_LOOKUP -36
- #define EXPRINFO_PWLIN -37
- #define EXPRINFO_SPLINE -38
- #define EXPRINFO_PWCONSTLC -39
- #define EXPRINFO_PWCONSTRC -40
- #define EXPRINFO_IN -41
- #define EXPRINFO_IF -42
- #define EXPRINFO_AND -43
- #define EXPRINFO_OR -44
- #define EXPRINFO_NOT -45
- #define EXPRINFO_IMPLIES -46
- #define EXPRINFO_COUNT -47
- #define EXPRINFO_ALLDIFF -48
- #define EXPRINFO_HISTOGRAM -49
- #define EXPRINFO_LEVEL -50
- #define EXPRINFO_NEIGHBOR -51
- #define EXPRINFO_NOGOOD -52
- #define EXPRINFO_EXPECTATION -53
- #define EXPRINFO_INTEGRAL -54
- #define EXPRINFO_DET -55
- #define EXPRINFO_COND -56
- #define EXPRINFO_PSD -57
- #define EXPRINFO_MPROD -58
- #define EXPRINFO_FEM -59
- #define EXPRINFO_CMPROD -60
- #define EXPRINFO_CGFEM -61
- #define EXPRINFO_ASQR -62
- #define EXPRINFO_D1FUNC -63
- #define EXPRINFO_BLACKBOX -64
- #define EXPRINFO_UNDEFINED -65
- #define EXPRINFO_NUMOFPREDEF -(EXPRINFO_UNDEFINED)

Typedefs

- typedef interval coco::rhs_t
- typedef std::vector< void * > coco::evaluator_v
- typedef boost::numeric::interval_lib::policies < my_rounded_math< double > , checking_my< double >> coco::coco::my_policies
- typedef mpq_t coco::num::rat
- typedef MP_RAT * coco::num::prat
- typedef const MP_RAT * coco::num::cprat
- typedef coco::interval coco::num::intv
- typedef std::string coco::num::str

Enumerations

- enum coco::e_expression_type { coco::ex_bound = 1, coco::ex_linear = 1<<1, coco::ex_quadratic = 1<<2, coco::ex_polynomial = 1<<3, coco::ex_other = 1<<4, coco::ex_kj = 1<<7, coco::ex_org = 1<<8, coco::ex_redundant = 1<<9, coco::ex_notredundant = 1<<10, coco::ex_active_lo = 1<<11, coco::ex_inactive_lo = 1<<12, coco::ex_active_hi = 1<<13, coco::ex_inactive_hi = 1<<14, coco::ex_active = ex_active_lo|ex_active_hi, coco::ex_inactive = ex_inactive_lo|ex_inactive_hi, coco::ex_integer = 1<<15, coco::ex_exists = 1<<16, coco::ex_forall = 1<<17, coco::ex_free = 1<<18, coco::ex_stochastic = 1<<19, coco::ex_convex = 1<<20, coco::ex_concave = 1<<21, coco::ex_inequality = 1<<28, coco::ex_equality = 1<<29, coco::ex_leftbound = 1<<30, coco::ex_rightbound = 1<<31, coco::ex_atmlin = ex_bound|ex_linear, coco::ex_atmquad = ex_atmlin|ex_quadratic, coco::ex_atmpoly = ex_atmquad|ex_polynomial, coco::ex_nonlin = ex_quadratic|ex_polynomial|ex_other, coco::ex_nonbnd = ex_linear|ex_nonlin, coco::ex_any = ex_atmlin|ex_nonlin, coco::ex_bothbound = ex_leftbound|ex_rightbound }
- enum coco::num::numtypes { coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL }
- enum coco::num::number_exception_type { coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR }
- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

  Enum for classifying api_exceptions.

Functions

- double coco::coco::safeguarded_mid (const interval &__i)
- interval coco::coco::ipow (const interval &x, int n)
- interval coco::coco::gauss (const interval &x)
- interval coco::coco::atan2 (const interval &y, const interval &x)
- double coco::coco::absmin (const interval &__i)
- double coco::coco::gainfactor (const interval &_old, const interval &_new)
- template<class _TC >
  bool coco::coco::operator== (const interval &__i, const _TC &__d)
- template<class _TC >
  bool coco::coco::operator!= (const interval &__i, const _TC &__d)
- bool coco::coco::operator== (const interval &a, double b)
- bool coco::coco::operator!= (const interval &a, double b)
- bool coco::coco::operator< (const interval &a, double b)
- bool coco::coco::operator< (double a, const interval &b)
- interval coco::coco::operator+ (const interval &a, double b)
- interval coco::coco::operator+ (double b, const interval &a)
- interval coco::coco::operator- (const interval &a, double b)
- interval coco::coco::operator- (double b, const interval &a)
- interval coco::coco::cancel (const interval &a, const interval &b)
- interval coco::coco::operator* (const interval &a, double b)
- interval coco::coco::operator* (double b, const interval &a)
- interval coco::coco::operator/ (const interval &a, double b)
- interval coco::coco::operator/ (double b, const interval &a)
- std::ostream & coco::coco::operator<< (std::ostream &s, const interval &a)
- double coco::coco::mid (const interval &)
- double coco::coco::diam (const interval &)
- double coco::coco::width (const interval &)
- double coco::coco::relDiam (const interval &)
- double coco::coco::rad (const interval &)
- double coco::coco::mig (const interval &)
- double coco::coco::mag (const interval &)
- double coco::coco::dist (const interval &x, const interval &y)
- interval coco::coco::round_to_integer (const interval &x)
- interval coco::coco::acos (const interval &x)
- interval coco::coco::abs (const interval &x)
- interval coco::coco::acosh (const interval &x)
- interval coco::coco::acot (const interval &x)
- interval coco::coco::acoth (const interval &x)
- interval coco::coco::asin (const interval &x)
- interval coco::coco::asinh (const interval &x)
- interval coco::coco::atan (const interval &x)
- interval coco::coco::atanh (const interval &x)
- interval coco::coco::cos (const interval &x)
- interval coco::coco::cosh (const interval &x)
- interval coco::coco::cot (const interval &x)
- interval coco::coco::coth (const interval &x)
- interval coco::coco::exp (const interval &x)
- interval coco::coco::exp10 (const interval &x)

- interval coco::coco::exp2 (const interval &x)
- interval coco::coco::expm1 (const interval &x)
- interval coco::coco::log (const interval &x)
- interval coco::coco::log10 (const interval &x)
- interval coco::coco::log1p (const interval &x)
- interval coco::coco::log2 (const interval &x)
- interval coco::coco::power (const interval &x, int n)
- interval coco::coco::pow (const interval &x, const interval &y)
- interval coco::coco::sin (const interval &x)
- interval coco::coco::sinh (const interval &x)
- interval coco::coco::sqr (const interval &x)
- interval coco::coco::sqrt (const interval &x)
- interval coco::coco::tan (const interval &x)
- interval coco::coco::tanh (const interval &x)
- interval coco::coco::imax (const interval &x, const interval &y)
- interval coco::coco::imin (const interval &x, const interval &y)
- interval coco::coco::division_part2 (const interval &x, const interval &y, bool b=true)
- template<class _TH >
  std::string coco::coco::convert_to_str (const _TH &_h)

  Convert an object of any printable class to a string.
- std::string coco::coco::i2a (int x)

  converter int -> string
- std::string coco::coco::localdatetime ()

  converter date & time into std::string
- std::string coco::coco::datetime (const struct tm *timeptr)

  local time as C++ string
- std::string coco::coco::e2D (double d)

  Fortran real as C++ string.
- const basic_alltype & coco::coco::basic_alltype_empty ()
- std::ostream & coco::coco::operator<< (std::ostream &os, const basic_alltype &b)
- bool coco::coco::less_than (const basic_alltype &a, const basic_alltype &b)
- bool coco::coco::less_equal (const basic_alltype &a, const basic_alltype &b)

- bool coco::coco::operator== (const interval &a, const interval &b)

- bool coco::coco::operator!= (const interval &a, const interval &b)

- bool coco::coco::operator< (const interval &a, const interval &b)

- interval coco::coco::operator+ (const interval &a, const interval &b)

- interval coco::coco::operator- (const interval &a, const interval &b)
- interval coco::coco::operator∗ (const interval &a, const interval &b)
- interval coco::coco::operator/ (const interval &a, const interval &b)
- interval coco::coco::division_part1 (const interval &x, const interval &y, bool &b)

11.89.1 Detailed Description

Definition in file expression.h.

11.89.2 Define Documentation

11.89.2.1 #define EXPR_LASTARG NULL

Definition at line 395 of file search_graph.cc.

11.89.2.2 #define EXPR_LASTARG NULL

11.89.2.3 #define EXPRINFO_ABS -14

11.89.2.4 #define EXPRINFO_ALLDIFF -48

11.89.2.5 #define EXPRINFO_AND -43

11.89.2.6 #define EXPRINFO_ARG -34

11.89.2.7 #define EXPRINFO_ASQR -62

11.89.2.8 #define EXPRINFO_ATAN2 -26

11.89.2.9 #define EXPRINFO_BLACKBOX -64

11.89.2.10 #define EXPRINFO_CGFEM -61

11.89.2.11 #define EXPRINFO_CMPROD -60

11.89.2.12 #define EXPRINFO_COND -56

11.89.2.13 #define EXPRINFO_CONSTANT -1

11.89.2.14 #define EXPRINFO_COS -19

11.89.2.15 #define EXPRINFO_COSH -21

11.89.2.16 #define EXPRINFO_COUNT -47

11.89.2.17 #define EXPRINFO_CPLXCONJ -35

11.89.2.18 #define EXPRINFO_D1FUNC -63

11.89.2.19 #define EXPRINFO_DET -55

11.89.2.20 #define EXPRINFO_DIV -25

11.89.2.21 #define EXPRINFO_EXP -16

11.89.2.22 #define EXPRINFO_EXPECTATION -53

11.89.2.23 #define EXPRINFO_FEM -59

11.89.2.24 #define EXPRINFO_GAUSS -22

11.89.2.25 #define EXPRINFO_GHOST 0

11.89.2.26 #define EXPRINFO_HISTOGRAM -49

11.89.2.27 #define EXPRINFO_IF -42

11.89.2.28 #define EXPRINFO_IM -33

11.89.2.29 #define EXPRINFO_IMPLIES -46

11.89.2.30 #define EXPRINFO_IN -41

11.89.2.31 #define EXPRINFO_INTEGRAL -54

11.89.2.32 #define EXPRINFO_INTPOWER -15

11.89.2.33 #define EXPRINFO_INVERT -11

11.89.2.34 #define EXPRINFO_IQUAD -29

11.89.2.35 #define EXPRINFO_LEVEL -50

11.89.2.36 #define EXPRINFO_LIN -27

11.89.2.37 #define EXPRINFO_LINSQR -31

11.89.2.38 #define EXPRINFO_LOG -17

11.89.2.39 #define EXPRINFO_LOOKUP -36

11.89.2.40 #define EXPRINFO_MAX -6

11.89.2.41 #define EXPRINFO_MEAN -4

11.89.2.42 #define EXPRINFO_MIN -7

11.89.2.43 #define EXPRINFO_MONOME -8

11.89.2.44 #define EXPRINFO_MPROD -58

11.89.2.45 #define EXPRINFO_NEIGHBOR -51

11.89.2.46 #define EXPRINFO_NOGOOD -52

11.89.2.47 #define EXPRINFO_NORM -10

11.89.2.48 #define EXPRINFO_NOT -45

11.89.2.49 #define EXPRINFO_NUMOFPREDEF -(EXPRINFO_UNDEFINED)

11.89.2.50 #define EXPRINFO_OR -44

11.89.2.51 #define EXPRINFO_POLY -23

11.89.2.52 #define EXPRINFO_POW -24

11.89.2.53 #define EXPRINFO_PROD -5

11.89.2.54 #define EXPRINFO_PSD -57

11.89.2.55 #define EXPRINFO_PWCONSTLC -39

11.89.2.56 #define EXPRINFO_PWCONSTRC -40

11.89.2.57 #define EXPRINFO_PWLIN -37

11.89.2.58 #define EXPRINFO_QUAD -28

11.89.2.59 #define EXPRINFO_RE -32

11.89.2.60 #define EXPRINFO_SCPROD -9

11.89.2.61 #define EXPRINFO_SIN -18

11.89.2.62 #define EXPRINFO_SINH -20

11.89.2.63 #define EXPRINFO_SPLINE -38

11.89.2.64 #define EXPRINFO_SQROOT -13

11.89.2.65 #define EXPRINFO_SQUARE -12

11.89.2.66 #define EXPRINFO_SUM -3

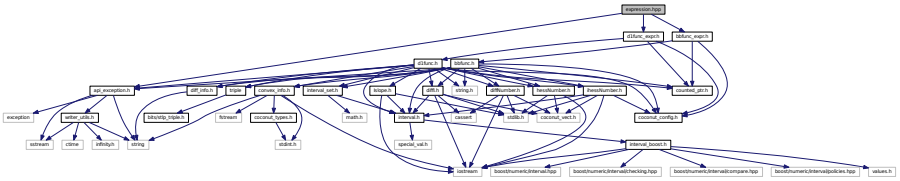
11.89.2.67 #define EXPRINFO_UNDEFINED -65

11.89.2.68 #define EXPRINFO_VARIABLE -2

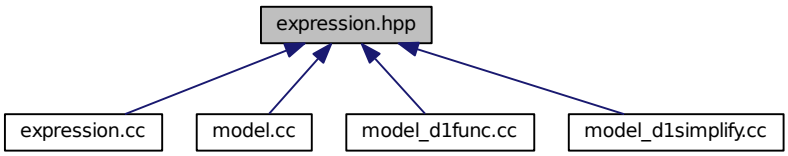
11.89.2.69 #define EXPRINFO_WSUMSQR -30

11.90 expression.hpp File Reference

`#include <api_exception.h> #include <d1func_expr.h> #include <bbfunc_-expr.h>` Include dependency graph for expression.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::expression_node::parents_compare
- class coco::expression_node::children_compare
- class coco::expression_node::parents_compare_eq
- class coco::expression_print_visitor

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

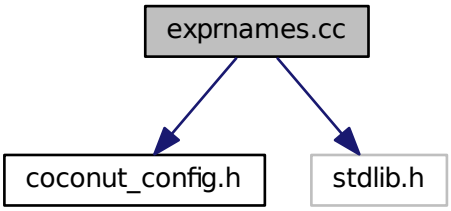
- std::ostream & coco::__wr_interval (std::ostream &o, const interval &__i)

11.90.1 Detailed Description

Definition in file expression.hpp.

11.91 exprnames.cc File Reference

`#include <coconut_config.h>` `#include <stdlib.h>` Include dependency graph for exprnames.cc-
:



Namespaces

- namespace coco

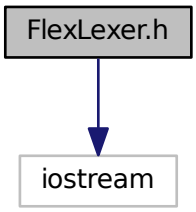
  the main namespace of the COCONUT API

11.91.1 Detailed Description

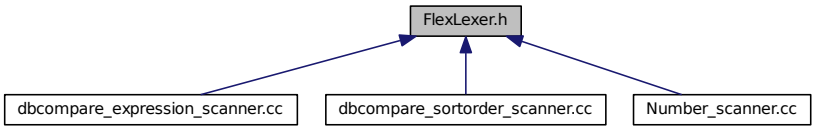
Definition in file exprnames.cc.

11.92 FlexLexer.h File Reference

`#include <iostream>` Include dependency graph for FlexLexer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class FlexLexer
- class yyFlexLexer

Defines

- #define FLEX_STD std::
- #define yyFlexLexerOnce

Typedefs

- typedef int yy_state_type

11.92.1 Define Documentation

11.92.1.1 #define FLEX_STD std::

Definition at line 53 of file FlexLexer.h.

11.92.1.2 #define yyFlexLexerOnce

Definition at line 108 of file FlexLexer.h.

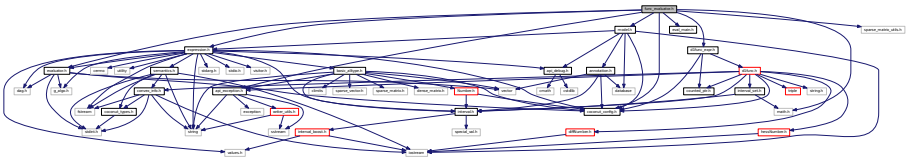
11.92.2 Typedef Documentation

11.92.2.1 typedef int yy_state_type

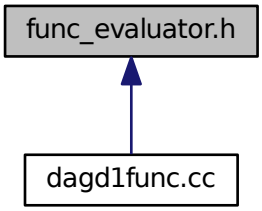
Definition at line 58 of file FlexLexer.h.

11.93 func_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <math.h>` `#include <d1func_expr.h>` `#include <api_exception.h>` `#include <sparse_matrix_utils.-h>` Include dependency graph for func_evaluator.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct coco::func_eval_type

  Visitor data for func_eval.
- class coco::func_eval

  Forward function evaluation.

Namespaces

- namespace coco

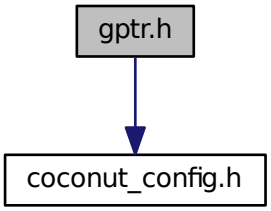
  the main namespace of the COCONUT API

11.93.1 Detailed Description

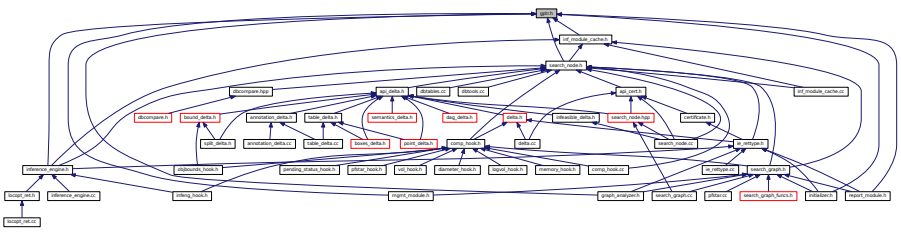
Definition in file func_evaluator.h.

11.94 gptr.h File Reference

`#include <coconut_config.h>` Include dependency graph for gptr.h:



This graph shows which files directly or indirectly include this file:



Classes

- class gptr

  Global pointer class.
- class ptr

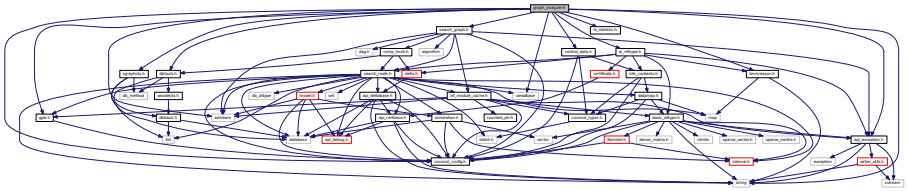
  Local global pointer class.

11.94.1 Detailed Description

Definition in file gptr.h.

11.95 graph_analyzer.h File Reference

`#include <iostream>` `#include <coconut_config.h>` `#include <search_graph.h>` `#include <termreason.h>` `#include <gptr.h>` `#include <string>` `#include <dbtools.h>` `#include <viewdbase>` `#include <ie_statistic.h>` `#include <control_data.h>` `#include <ie_rettype.h>` `#include <sgraphctx.h>` `#include <api_exception.h>` Include dependency graph for graph_analyzer.h:



Classes

- class coco::graph_analyzer_exception

  Graph analyzer exception class.
- class coco::graph_analyzer

  Graph analyzer base class.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef ie_return_type coco::ga_return_type

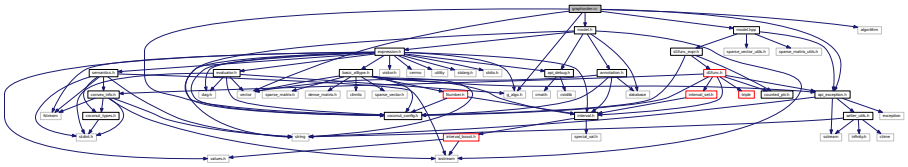
  Graph analyzer return type.

11.95.1 Detailed Description

Definition in file graph_analyzer.h.

11.96 graphorder.cc File Reference

`#include <coconut_config.h>` `#include <model.h>` `#include <vector>` `#include <algorithm>` `#include <g_algo.h>` `#include <api_exception.h>` `#include <model.hpp>` Include dependency graph for graphorder.cc:



Classes

- class coco::graphorder_visitor

  This visitor class is used for computing a graph order.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- void coco::_internal_graphorder (const model &DAG, std::vector< unsigned int > &order, std::vector< unsigned int > *inv_order)
- void coco::graphorder (const model &DAG, std::vector< unsigned int > &order)

  Compute a graph order.
- void coco::graphorder (const model &DAG, std::vector< unsigned int > &order, std::vector< unsigned int > &inv_order)

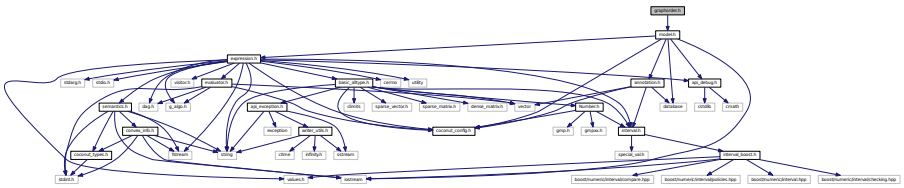
  Compute a graph order.

11.96.1 Detailed Description

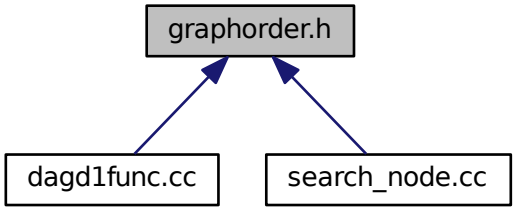
Definition in file graphorder.cc.

11.97 graphorder.h File Reference

`#include <model.h>` Include dependency graph for graphorder.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- void coco::graphorder (const model &DAG, std::vector< unsigned int > &order)

  Compute a graph order.
- void coco::graphorder (const model &DAG, std::vector< unsigned int > &order, std::vector< unsigned int > &inv_order)

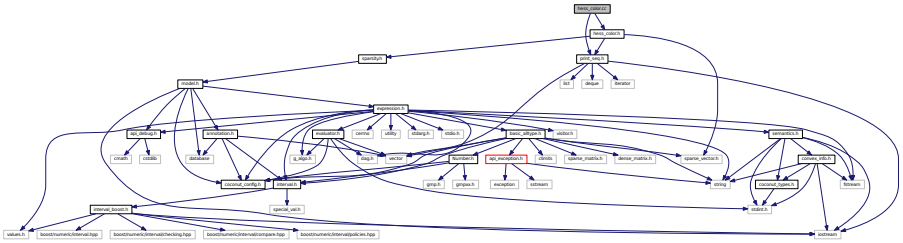
  Compute a graph order.

11.97.1 Detailed Description

Definition in file graphorder.h.

11.98 hess_color.cc File Reference

`#include <hess_color.h>` `#include <print_seq.h>` Include dependency graph for hess-_color.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define HESS_COLOR_DEBUG 0

Functions

- void coco::hess_star_color1 (const unsigned int n, const vmtl::sparse_matrix< int > &H, std::vector< std::vector< unsigned int > > &parts, std::vector< vmtl::sparse_vector< unsigned int > > &pro-vides)

11.98.1 Detailed Description

Definition in file hess_color.cc.

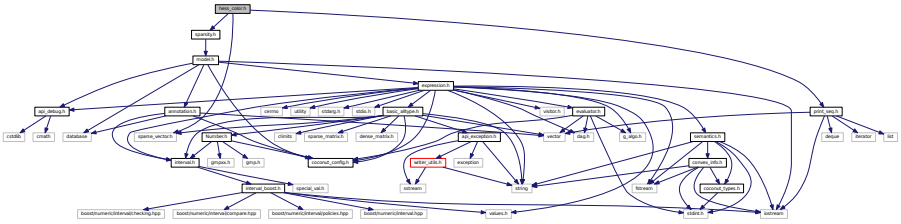
11.98.2 Define Documentation

11.98.2.1 #define HESS_COLOR_DEBUG 0

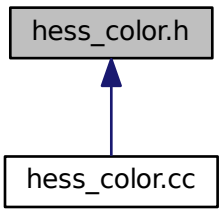
Definition at line 33 of file hess_color.cc.

11.99 hess_color.h File Reference

`#include <sparsity.h> #include <sparse_vector.h> #include <print_seq.-h>` Include dependency graph for hess_color.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define HESS_VEC_THRESHOLD 0.9

Functions

- void coco::hess_star_color1 (const unsigned int n, const vmtl::sparse_matrix< int > &H, std::vector< std::vector< unsigned int > > &parts, std::vector< vmtl::sparse_vector< unsigned int > > &provides)

11.99.1 Detailed Description

Definition in file hess_color.h.

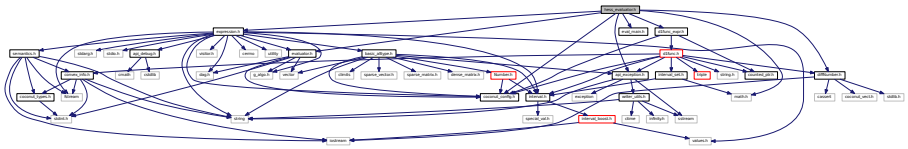
11.99.2 Define Documentation

11.99.2.1 #define HESS_VEC_THRESHOLD 0.9

Definition at line 40 of file hess_color.h.

11.100 hess_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <eval_main.h>` `#include <math.h>` `#include <api_exception.h>` × `#include <d1func_expr.h>` `#include <diffNumber.h>` Include dependency graph for hess-_evaluator.h:



Classes

- struct coco::hessPreparationEvaluatorType
- class coco::hessPreparationEvaluator
- struct coco::hessForwardEvaluatorReturnValue
- struct coco::hessForwardEvaluatorType
- class coco::hessForwardEvaluator
- struct coco::hessBackwardEvaluatorType
- class coco::hessBackwardEvaluator

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DEBUG_HESS_EVALUATOR 0
- #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

11.100.1 Detailed Description

Definition in file hess_evaluator.h.

11.100.2 Define Documentation

11.100.2.1 #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

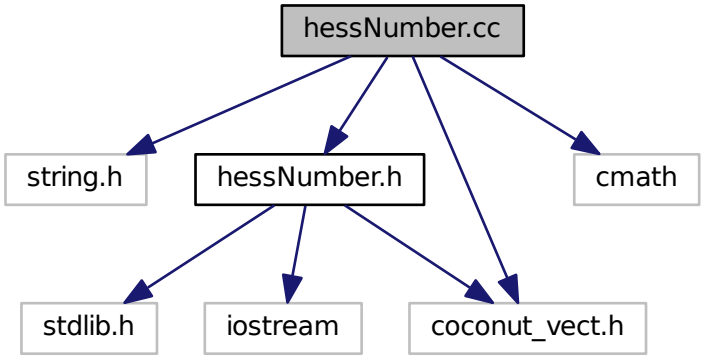
Definition at line 31 of file hess_evaluator.h.

11.100.2.2 #define DEBUG_HESS_EVALUATOR 0

Definition at line 30 of file hess_evaluator.h.

11.101 hessNumber.cc File Reference

`#include <string.h>` `#include <hessNumber.h>` `#include <coconut_vect.h>` × `#include <cmath>` Include dependency graph for hessNumber.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

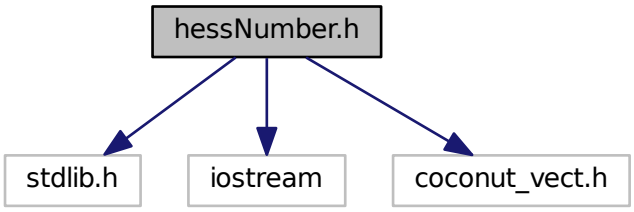
Functions

- hessNumber coco::operator∗ (const hessNumber &a, const hessNumber &b)
- hessNumber coco::operator/ (const hessNumber &a, const hessNumber &b)

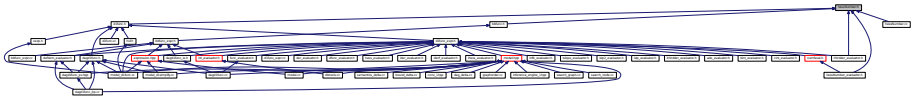
- hessNumber coco::operator/ (const double &a, const hessNumber &b)
- hessNumber coco::exp (const hessNumber &a)
- hessNumber coco::sin (const hessNumber &a)
- hessNumber coco::cos (const hessNumber &a)
- hessNumber coco::sqrt (const hessNumber &a)
- hessNumber coco::abs (const hessNumber &a)
- hessNumber coco::power (const hessNumber &a, int n)
- hessNumber coco::pow (const hessNumber &a, const hessNumber &b)
- hessNumber coco::log (const hessNumber &a)
- hessNumber coco::sinh (const hessNumber &a)
- hessNumber coco::cosh (const hessNumber &a)
- hessNumber coco::atan2 (const hessNumber &a, const hessNumber &b)
- hessNumber coco::atan (const hessNumber &a)
- hessNumber coco::square (const hessNumber &a)
- std::ostream & coco::operator<< (std::ostream &s, const hessNumber &a)

11.102 hessNumber.h File Reference

`#include <stdlib.h>` `#include <iostream>` `#include <coconut_vect.h>` Include dependency graph for hessNumber.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::hessNumber

Namespaces

- namespace coco

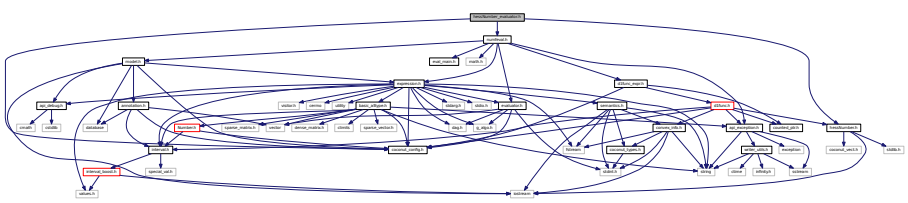
  the main namespace of the COCONUT API

Functions

- hessNumber coco::operator+ (const hessNumber &a, const hessNumber &b)
- hessNumber coco::operator+ (const double &a, const hessNumber &b)
- hessNumber coco::operator+ (const hessNumber &a, const double &b)
- hessNumber coco::operator- (const hessNumber &a, const hessNumber &b)
- hessNumber coco::operator- (const double &a, const hessNumber &b)
- hessNumber coco::operator- (const hessNumber &a, const double &b)
- hessNumber coco::operator- (const hessNumber &a)
- hessNumber coco::operator∗ (const hessNumber &a, const hessNumber &b)
- hessNumber coco::operator∗ (const double &a, const hessNumber &b)
- hessNumber coco::operator∗ (const hessNumber &a, const double &b)
- hessNumber coco::operator/ (const hessNumber &a, const hessNumber &b)
- hessNumber coco::operator/ (const hessNumber &a, const double &b)
- hessNumber coco::operator/ (const double &a, const hessNumber &b)
- std::ostream & coco::operator<< (std::ostream &s, const hessNumber &a)
- hessNumber coco::exp (const hessNumber &a)
- hessNumber coco::sin (const hessNumber &a)
- hessNumber coco::cos (const hessNumber &a)
- hessNumber coco::sqrt (const hessNumber &a)
- hessNumber coco::abs (const hessNumber &a)
- hessNumber coco::power (const hessNumber &a, int n)
- hessNumber coco::pow (const hessNumber &a, const hessNumber &b)
- hessNumber coco::max (const hessNumber &a, const hessNumber &b)
- hessNumber coco::min (const hessNumber &a, const hessNumber &b)
- hessNumber coco::log (const hessNumber &a)
- hessNumber coco::sinh (const hessNumber &a)
- hessNumber coco::cosh (const hessNumber &a)
- hessNumber coco::atan2 (const hessNumber &a, const hessNumber &b)
- hessNumber coco::atan (const hessNumber &a)
- hessNumber coco::square (const hessNumber &a)

11.103 hessNumber_evaluator.h File Reference

`#include <coconut_config.h>` `#include <hessNumber.h>` `#include <numfeval.-h>` Include dependency graph for hessNumber_evaluator.h:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define xxxNumber_t hessNumber
- #define xxxNumber_name xxxNumber_t
- #define XXXNUMBER_HAS_DEGREE 0
- #define XXXNUMBER_HAS_1D 0
- #define XXXNUMBER_HAS_SQR 0
- #define XXXEVAL_HAS_BASES 0
- #define XXXNUMBER_EVAL_DEBUG 0

11.103.1 Detailed Description

Definition in file hessNumber_evaluator.h.

11.103.2 Define Documentation

11.103.2.1 #define XXXEVAL_HAS_BASES 0

Definition at line 65 of file hessNumber_evaluator.h.

11.103.2.2 #define XXXNUMBER_EVAL_DEBUG 0

Definition at line 70 of file hessNumber_evaluator.h.

11.103.2.3 #define XXXNUMBER_HAS_1D 0

Definition at line 55 of file hessNumber_evaluator.h.

11.103.2.4 #define XXXNUMBER_HAS_DEGREE 0

Definition at line 50 of file hessNumber_evaluator.h.

11.103.2.5 #define XXXNUMBER_HAS_SQR 0

Definition at line 60 of file hessNumber_evaluator.h.

11.103.2.6 #define xxxNumber_name xxxNumber_t

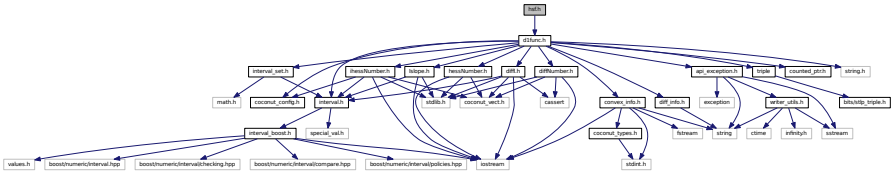
Definition at line 45 of file hessNumber_evaluator.h.

11.103.2.7 #define xxxNumber_t hessNumber

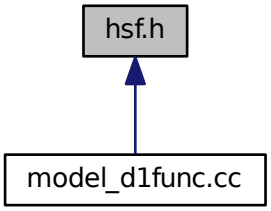
Definition at line 40 of file hessNumber_evaluator.h.

11.104 hsf.h File Reference

`#include <d1func.h>` Include dependency graph for hsf.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::hsf_func

  The hsf_func class (one dimensional function)

Namespaces

- namespace coco

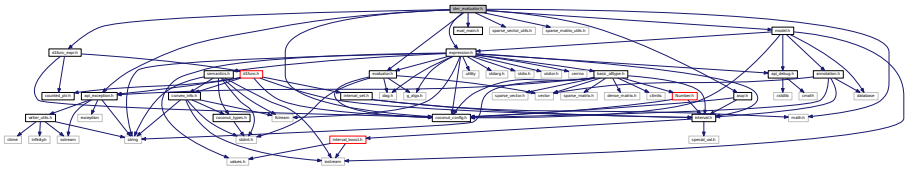
  the main namespace of the COCONUT API

11.104.1 Detailed Description

Definition in file hsf.h.

11.105 ider_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.h>` `#include <model.h>` `#include <eval_main.h>` `#include <sparse_vector_utils.h>` `#include <sparse_matrix_utils.h>` `#include <math.h>` `#include <api_exception.h>` `#include <d1func_expr.h>` `#include <asqr.h>` Include dependency graph for ider_evaluator.h:



Classes

- class coco::prep_id_eval

  Preparation Evaluator for interval derivatives.
- struct coco::func_id_eval_type

  Visitor data for func_id_eval.
- class coco::func_id_eval

  Forward function range evaluation with preparation of interval derivative data.
- struct coco::ider_eval_type

  Visitor data for ider_eval.
- class coco::ider_eval

  Backward interval gradient evaluation with prepared interval derivative data.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

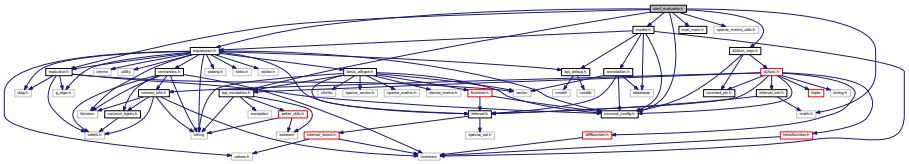
- typedef bool(∗ coco::prep_id_evaluator)()
- typedef interval(∗ coco::func_id_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v, std::vector< interval > &__id_data)
- typedef std::vector< interval > &(∗ coco::ider_evaluator)(const std::vector< interval > &__d_dat, const variable_indicator &__v)

11.105.1 Detailed Description

Definition in file ider_evaluator.h.

11.106 iderf_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <sparse_matrix_-utils.h>` `#include <math.h>` `#include <d1func_expr.h>` `#include <api_exception.-h>` Include dependency graph for iderf_evaluator.h:



Classes

- struct coco::iderf_ret_type

  Visitor data for iderf_eval return value.
- struct coco::iderf_eval_type

  Visitor data for iderf_eval.
- class coco::iderf_eval

  Forward function and derivative range evaluation.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

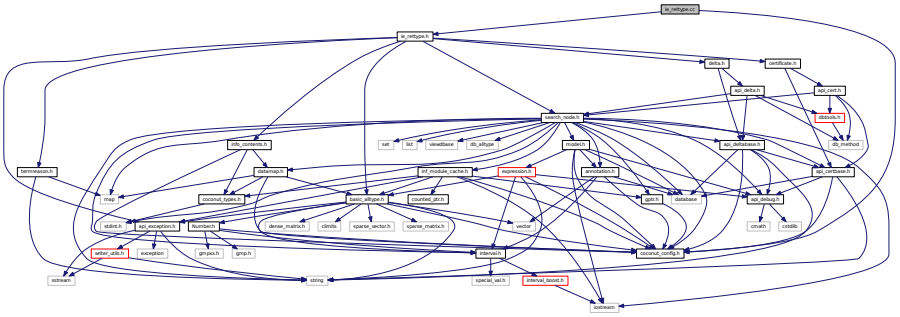
- typedef iderf_ret_type(∗ coco::iderf_evaluator)(const std::vector< interval > ∗__x, const variable-_indicator &__v)

11.106.1 Detailed Description

Definition in file iderf_evaluator.h.

11.107 ie_rettype.cc File Reference

`#include <coconut_config.h>` `#include <ie_rettype.h>` Include dependency graph for ie_rettype.cc:



Namespaces

- namespace coco

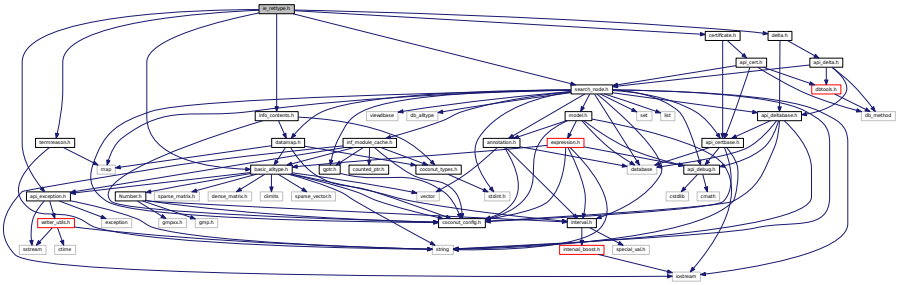
  the main namespace of the COCONUT API

11.107.1 Detailed Description

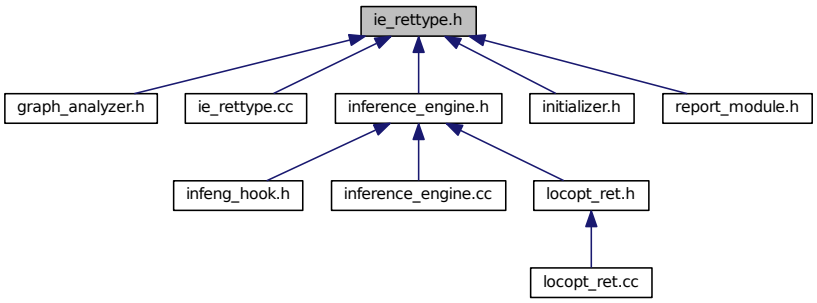
Definition in file ie_rettype.cc.

11.108 ie_rettype.h File Reference

`#include <basic_alltype.h>` `#include <termreason.h>` `#include <info_contents.h>` `#include <search_node.h>` `#include <delta.h>` `#include <api_exception.h>` `#include <certificate.h>` Include dependency graph for ie_rettype.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::ie_return_type

  The return class of all inference engines.

Namespaces

- namespace coco

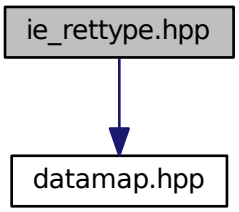
  the main namespace of the COCONUT API

11.108.1 Detailed Description

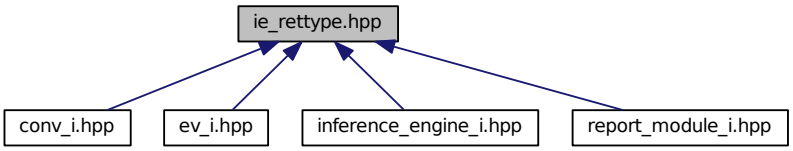
Definition in file ie_rettype.h.

11.109 ie_rettype.hpp File Reference

`#include <datamap.hpp>` Include dependency graph for ie_rettype.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

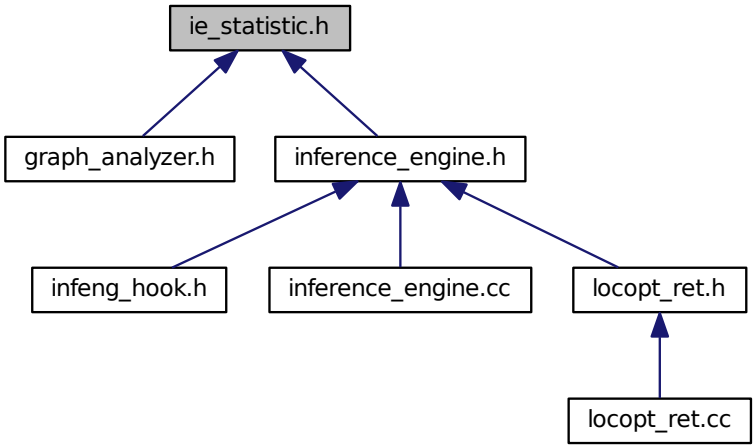
  the main namespace of the COCONUT API

11.109.1 Detailed Description

Definition in file ie_rettype.hpp.

11.110 ie_statistic.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class coco::statistic_info

  Base class for all inference engine statistics classes.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

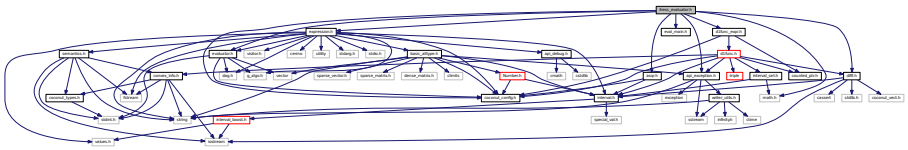
11.110.1 Detailed Description

Definition in file ie_statistic.h.

11.111 ihess_evaluator.h File Reference

```
#include <coconut_config.h> #include <evaluator.h> #include <expression.-
h> #include <eval_main.h> #include <math.h> #include <api_exception.h> ×
```

`#include <d1func_expr.h>` `#include <diffI.h>` `#include <asqr.h>` Include dependency graph for ihess_evaluator.h:



Classes

- struct coco::ihessPreparationEvaluatorType
- class coco::ihessPreparationEvaluator
- struct coco::ihessForwardEvaluatorReturnValue
- struct coco::ihessForwardEvaluatorType
- class coco::ihessForwardEvaluator
- struct coco::ihessBackwardEvaluatorType
- class coco::ihessBackwardEvaluator

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DEBUG_IHESS_EVALUATOR 0
- #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

11.111.1 Detailed Description

Definition in file ihess_evaluator.h.

11.111.2 Define Documentation

11.111.2.1 #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

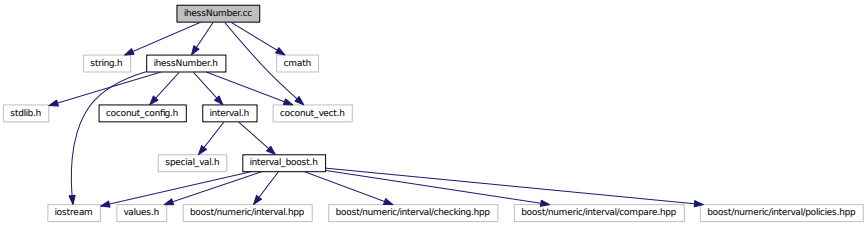
Definition at line 31 of file ihess_evaluator.h.

11.111.2.2 #define DEBUG_IHESS_EVALUATOR 0

Definition at line 30 of file ihess_evaluator.h.

11.112 ihessNumber.cc File Reference

`#include <string.h>` `#include <ihessNumber.h>` `#include <coconut_vect.h>` × `#include <cmath>` Include dependency graph for ihessNumber.cc:



Namespaces

- namespace coco

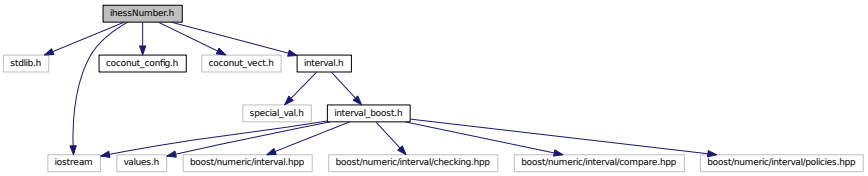
  the main namespace of the COCONUT API

Functions

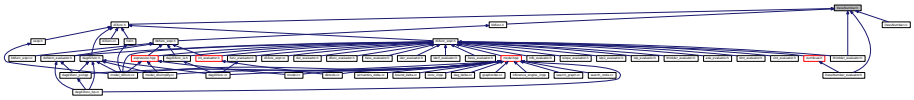
- ihessNumber coco::operator∗ (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::operator/ (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::operator/ (const double &a, const ihessNumber &b)
- ihessNumber coco::operator/ (const interval &a, const ihessNumber &b)
- ihessNumber coco::exp (const ihessNumber &a)
- ihessNumber coco::sin (const ihessNumber &a)
- ihessNumber coco::cos (const ihessNumber &a)
- ihessNumber coco::sqrt (const ihessNumber &a)
- ihessNumber coco::abs (const ihessNumber &a)
- ihessNumber coco::power (const ihessNumber &a, int n)
- ihessNumber coco::pow (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::max (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::min (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::log (const ihessNumber &a)
- ihessNumber coco::sinh (const ihessNumber &a)
- ihessNumber coco::cosh (const ihessNumber &a)
- ihessNumber coco::atan2 (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::atan (const ihessNumber &a)
- ihessNumber coco::sqr (const ihessNumber &a)
- std::ostream & coco::operator<< (std::ostream &s, const ihessNumber &a)

11.113 ihessNumber.h File Reference

`#include <stdlib.h>` `#include <iostream>` `#include <coconut_config.h>` `#include <coconut_vect.h>` `#include <interval.h>` Include dependency graph for ihessNumber.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::ihessNumber

Namespaces

- namespace coco

  the main namespace of the COCONUT API

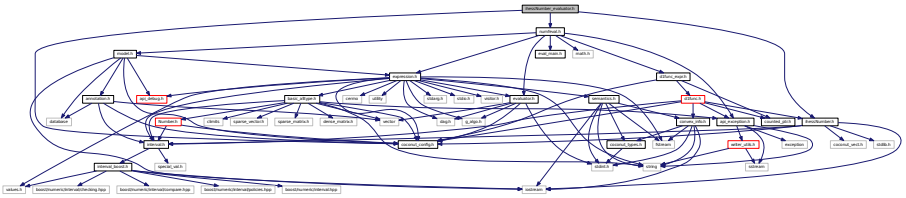
Functions

- ihessNumber coco::operator+ (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::operator+ (const double &a, const ihessNumber &b)
- ihessNumber coco::operator+ (const interval &a, const ihessNumber &b)
- ihessNumber coco::operator+ (const ihessNumber &a, const double &b)
- ihessNumber coco::operator+ (const ihessNumber &a, const interval &b)
- ihessNumber coco::operator- (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::operator- (const double &a, const ihessNumber &b)
- ihessNumber coco::operator- (const interval &a, const ihessNumber &b)
- ihessNumber coco::operator- (const ihessNumber &a, const double &b)
- ihessNumber coco::operator- (const ihessNumber &a, const interval &b)
- ihessNumber coco::operator- (const ihessNumber &a)
- ihessNumber coco::operator∗ (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::operator∗ (const double &a, const ihessNumber &b)

- ihessNumber coco::operator∗ (const interval &a, const ihessNumber &b)
- ihessNumber coco::operator∗ (const ihessNumber &a, const double &b)
- ihessNumber coco::operator∗ (const ihessNumber &a, const interval &b)
- ihessNumber coco::operator/ (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::operator/ (const ihessNumber &a, const double &b)
- ihessNumber coco::operator/ (const ihessNumber &a, const interval &b)
- ihessNumber coco::operator/ (const double &a, const ihessNumber &b)
- ihessNumber coco::operator/ (const interval &a, const ihessNumber &b)
- std::ostream & coco::operator<< (std::ostream &s, const ihessNumber &a)
- ihessNumber coco::exp (const ihessNumber &a)
- ihessNumber coco::sin (const ihessNumber &a)
- ihessNumber coco::cos (const ihessNumber &a)
- ihessNumber coco::sqrt (const ihessNumber &a)
- ihessNumber coco::abs (const ihessNumber &a)
- ihessNumber coco::power (const ihessNumber &a, int n)
- ihessNumber coco::pow (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::max (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::min (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::log (const ihessNumber &a)
- ihessNumber coco::sinh (const ihessNumber &a)
- ihessNumber coco::cosh (const ihessNumber &a)
- ihessNumber coco::atan2 (const ihessNumber &a, const ihessNumber &b)
- ihessNumber coco::atan (const ihessNumber &a)
- ihessNumber coco::sqr (const ihessNumber &a)

11.114 ihessNumber_evaluator.h File Reference

`#include <coconut_config.h>` `#include <ihessNumber.h>` `#include <numfeval.-h>` Include dependency graph for ihessNumber_evaluator.h:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define xxxNumber_t ihessNumber
- #define xxxNumber_name xxxNumber_t
- #define XXXNUMBER_HAS_DEGREE 0
- #define XXXNUMBER_HAS_1D 1
- #define XXXNUMBER_HAS_SQR 1
- #define XXXEVAL_HAS_BASES 0
- #define XXXNUMBER_EVAL_DEBUG 0

11.114.1 Detailed Description

Definition in file ihessNumber_evaluator.h.

11.114.2 Define Documentation

11.114.2.1 #define XXXEVAL_HAS_BASES 0

Definition at line 65 of file ihessNumber_evaluator.h.

11.114.2.2 #define XXXNUMBER_EVAL_DEBUG 0

Definition at line 70 of file ihessNumber_evaluator.h.

11.114.2.3 #define XXXNUMBER_HAS_1D 1

Definition at line 55 of file ihessNumber_evaluator.h.

11.114.2.4 #define XXXNUMBER_HAS_DEGREE 0

Definition at line 50 of file ihessNumber_evaluator.h.

11.114.2.5 #define XXXNUMBER_HAS_SQR 1

Definition at line 60 of file ihessNumber_evaluator.h.

11.114.2.6 #define xxxNumber_name xxxNumber_t

Definition at line 45 of file ihessNumber_evaluator.h.

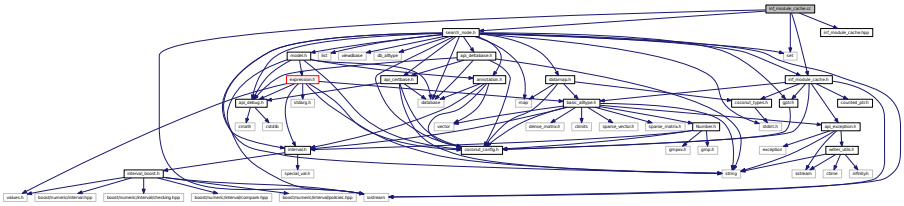
11.114.2.7 #define xxxNumber_t ihessNumber

Definition at line 40 of file ihessNumber_evaluator.h.

11.115 inf_module_cache.cc File Reference

`#include <iostream>` `#include <set>` `#include <inf_module_cache.h>` `#include <inf_module_cache.hpp>` `#include <search_node.h>` Include dependency graph for inf-

_module_cache.cc:



Namespaces

- namespace coco

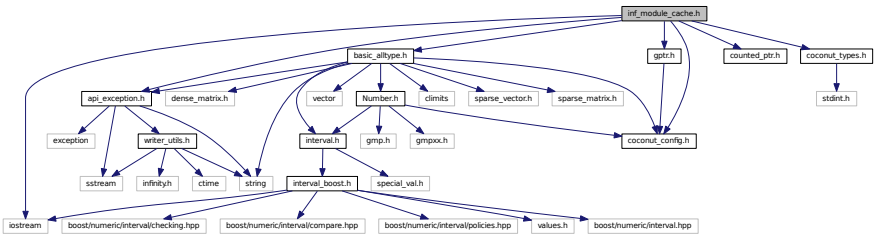
  the main namespace of the COCONUT API

11.115.1 Detailed Description

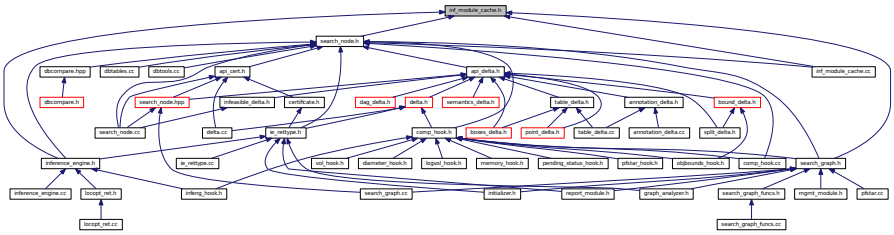
Definition in file inf_module_cache.cc.

11.116 inf_module_cache.h File Reference

`#include <iostream>` `#include <gptr.h>` `#include <counted_ptr.h>` `#include <coconut_config.h>` `#include <coconut_types.h>` `#include <api_exception.h>` `#include <basic_alltype.h>` Include dependency graph for inf_module_cache.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::inference_module_cache_autogen

  Inference module cache auto-generate method base class.
- class coco::inference_module_cache

  Inference module cache class.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define INFMODCACHE_PERSISTENT 0
- #define INFMODCACHE_SUBBOXES 1
- #define INFMODCACHE_THISBOX 2
- #define INFMODCACHE_PERSISTENT 0
- #define INFMODCACHE_SUBBOXES 1
- #define INFMODCACHE_THISBOX 2

11.116.1 Detailed Description

Definition in file inf_module_cache.h.

11.116.2 Define Documentation

11.116.2.1 #define INFMODCACHE_PERSISTENT 0

Definition at line 41 of file search_graph.cc.

11.116.2.2 #define INFMODCACHE_PERSISTENT 0

11.116.2.3 #define INFMODCACHE_SUBBOXES 1

11.116.2.4 #define INFMODCACHE_SUBBOXES 1

Definition at line 42 of file search_graph.cc.

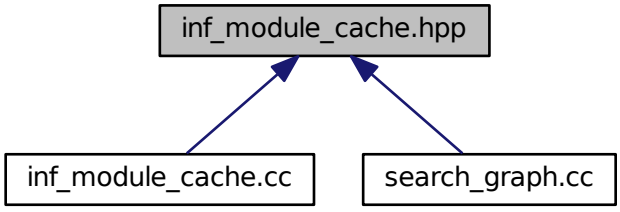
11.116.2.5 #define INFMODCACHE_THISBOX 2

Definition at line 43 of file search_graph.cc.

11.116.2.6 #define INFMODCACHE_THISBOX 2

11.117 inf_module_cache.hpp File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

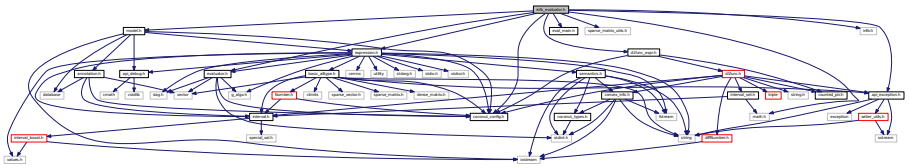
11.117.1 Detailed Description

Definition in file inf_module_cache.hpp.

11.118 infb_evaluator.h File Reference

```
#include <coconut_config.h> #include <evaluator.h> #include <expression.-
h> #include <model.h> #include <eval_main.h> #include <sparse_matrix_-
```

utils.h> #include <math.h> #include <infb.h> #include <d1func_expr.h> × #include <api_exception.h> Include dependency graph for infb_evaluator.h:



Classes

- struct coco::infbound_eval_type
- class coco::infbound_eval

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

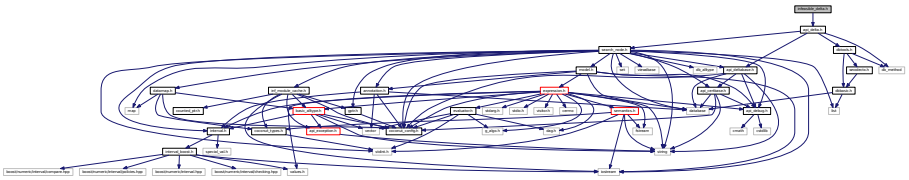
- typedef infbound(∗ coco::infbound_evaluator)(const std::vector< infbound > ∗__x, const variable_indicator &__v)

11.118.1 Detailed Description

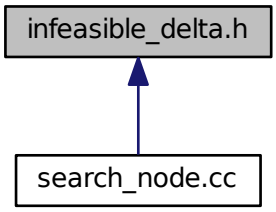
Definition in file infb_evaluator.h.

11.119 infeasible_delta.h File Reference

#include <api_delta.h> Include dependency graph for infeasible_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::infeasible_undelta

  The infeasible undelta class for undoing changes to the feasibility of a model.
- class coco::infeasible_delta

  The infeasible delta class for marking a model as infeasible.

Namespaces

- namespace coco

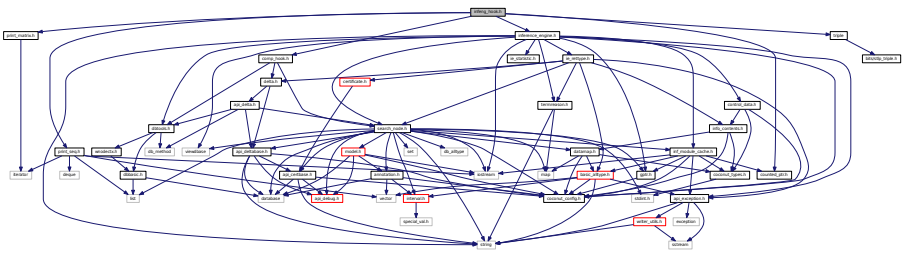
  the main namespace of the COCONUT API

11.119.1 Detailed Description

Definition in file infeasible_delta.h.

11.120 infeng_hook.h File Reference

`#include <comp_hook.h> #include <inference_engine.h> #include <triple> × #include <print_seq.h> #include <print_matrix.h> #include <counted_ptr.-h>` Include dependency graph for infeng_hook.h:



Classes

- class coco::inference_engine_comp_hook

  The inference engine meta computation hook (work node computation hook)

Namespaces

- namespace coco

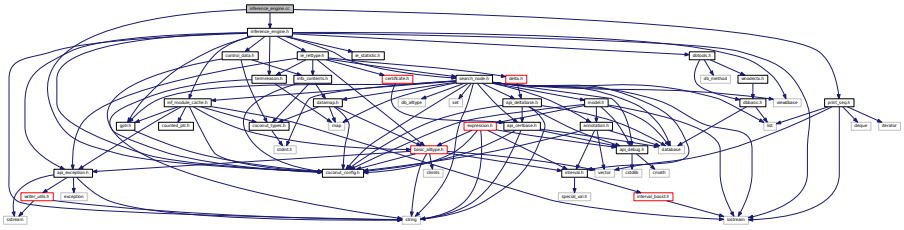
  the main namespace of the COCONUT API

11.120.1 Detailed Description

Definition in file infeng_hook.h.

11.121 inference_engine.cc File Reference

`#include <coconut_config.h>` `#include <inference_engine.h>` `#include <print-_seq.h>` Include dependency graph for inference_engine.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

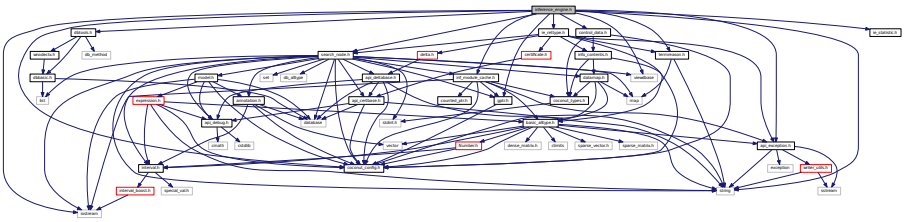
11.121.1 Detailed Description

Definition in file inference_engine.cc.

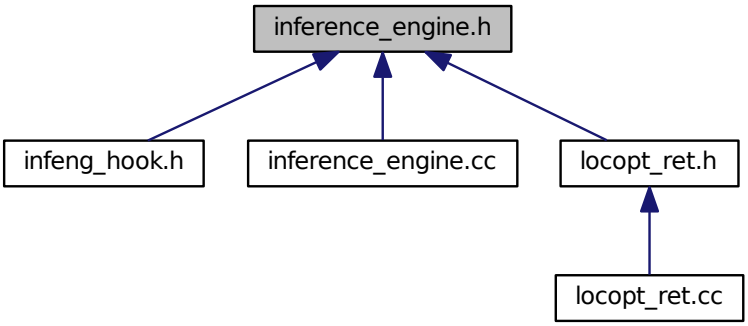
11.122 inference_engine.h File Reference

`#include <iostream>` `#include <coconut_config.h>` `#include <search_node.-h>` `#include <termreason.h>` `#include <gptr.h>` `#include <string>` `#include`

<dbtools.h> #include <viewdbase> #include <ie_statistic.h> #include <control-_data.h> #include <ie_rettype.h> #include <api_exception.h> #include <inf-_module_cache.h> Include dependency graph for inference_engine.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::inference_engine_exception
  Inference engine exception class.
- class coco::inference_engine
  Inference engine base class.

Namespaces

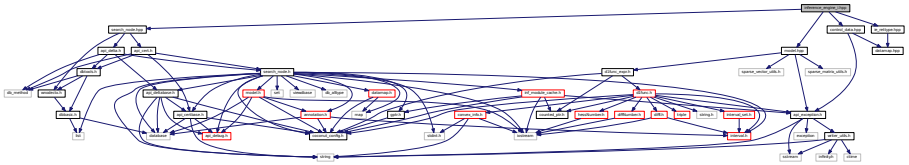
- namespace coco
  the main namespace of the COCONUT API

11.122.1 Detailed Description

Definition in file inference_engine.h.

11.123 inference_engine_i.hpp File Reference

`#include <search_node.hpp>` `#include <control_data.hpp>` `#include <ie_rettype.hpp>` `#include <model.hpp>` Include dependency graph for inference_engine_i.hpp:

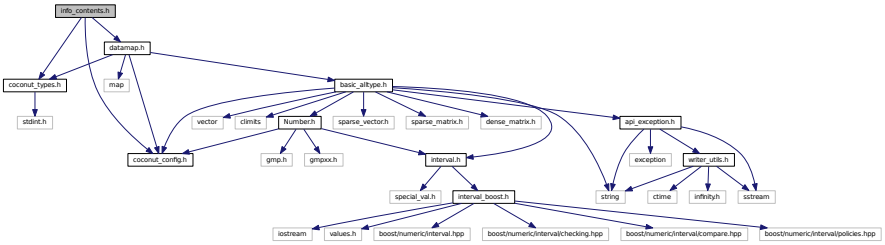


11.123.1 Detailed Description

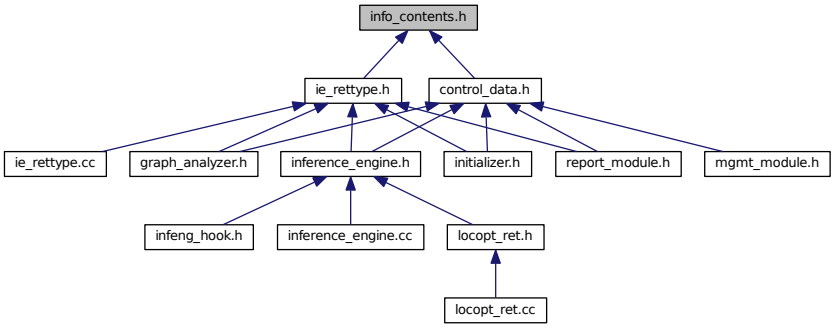
Definition in file inference_engine_i.hpp.

11.124 info_contents.h File Reference

`#include <coconut_config.h>` `#include <coconut_types.h>` `#include <datamap.h>` Include dependency graph for info_contents.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::info_contents

  The class for returning additional information from inference modules.

Namespaces

- namespace coco

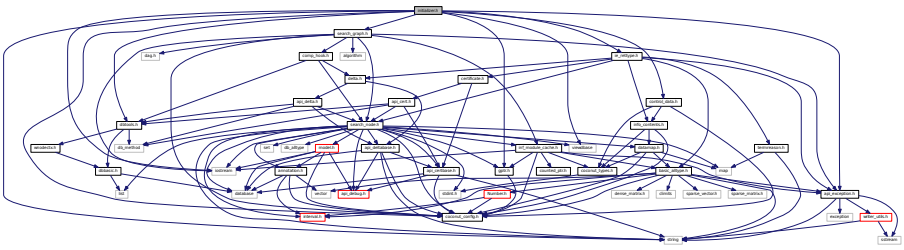
  the main namespace of the COCONUT API

11.124.1 Detailed Description

Definition in file info_contents.h.

11.125 initializer.h File Reference

`#include <iostream> #include <coconut_config.h> #include <search_graph.-h> #include <gptr.h> #include <string> #include <dbtools.h> #include <viewdbase> × #include <control_data.h> #include <ie_rettype.h> #include <api_exception.-h>` Include dependency graph for initializer.h:



Classes

- class coco::initializer_exception

  Initializer exception class.
- class coco::initializer

  Initializer base class.

Namespaces

- namespace coco

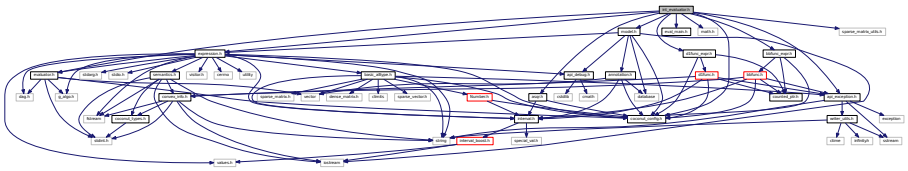
  the main namespace of the COCONUT API

11.125.1 Detailed Description

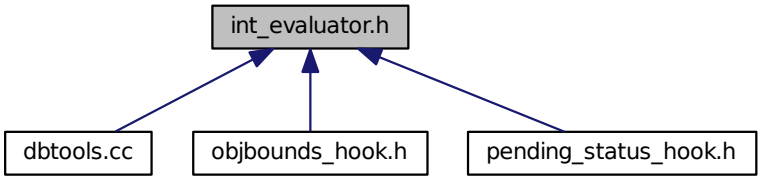
Definition in file initializer.h.

11.126 int_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <math.h>` `#include <api_exception.h>` `#include <d1func_expr.h>` `#include <bbfunc_expr.h>` `#include <asqr.h>` `#include <sparse_matrix_utils.h>` Include dependency graph for int_evaluator.h-:



This graph shows which files directly or indirectly include this file:



Classes

- struct coco::interval_eval_type

  Visitor data for interval_eval.
- class coco::interval_eval

  Forward function range evaluation.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

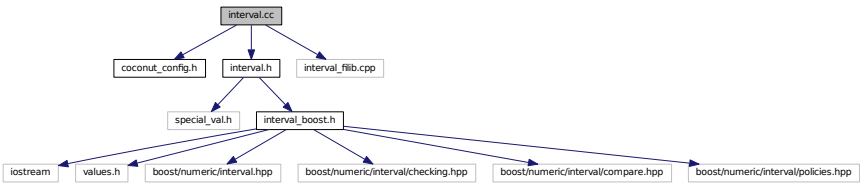
- typedef interval(∗ coco::interval_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v)

11.126.1 Detailed Description

Definition in file int_evaluator.h.

11.127 interval.cc File Reference

`#include <coconut_config.h> #include <interval.h> #include <interval_filib.cpp>` Include dependency graph for interval.cc:

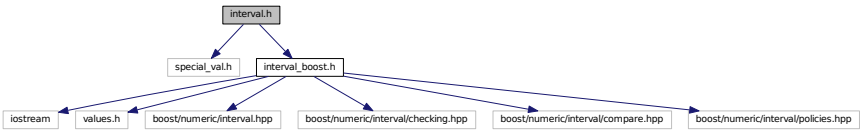


11.127.1 Detailed Description

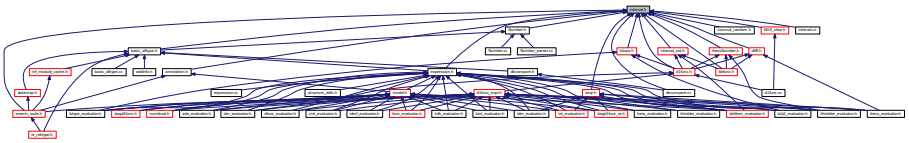
Definition in file interval.cc.

11.128 interval.h File Reference

`#include <special_val.h>` `#include <interval_boost.h>` Include dependency graph for interval.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))

- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))
- #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)
- #define get_integer(x) ((int) rint((x)))

11.128.1 Detailed Description

Definition in file interval.h.

11.128.2 Define Documentation

11.128.2.1 #define get_integer(*x*) ((int) rint((x)))

11.128.2.2 #define get_integer(*x*) ((int) rint((x)))

11.128.2.3 #define get_integer(*x*) ((int) rint((x)))

11.128.2.4 #define get_integer(*x*) ((int) rint((x)))

11.128.2.5 #define get_integer(*x*) ((int) rint((x)))

11.128.2.6 #define get_integer(*x*) ((int) rint((x)))

11.128.2.7 #define get_integer(*x*) ((int) rint((x)))

11.128.2.8 #define get_integer(*x*) ((int) rint((x)))

11.128.2.9 #define get_integer(*x*) ((int) rint((x)))

This macro returns the integer part of a double (as integer).

Definition at line 57 of file search_graph.cc.

11.128.2.10 #define get_integer(*x*) ((int) rint((x)))

11.128.2.11 #define get_integer(*x*) ((int) rint((x)))

11.128.2.12 #define get_integer(*x*) ((int) rint((x)))

11.128.2.13 #define get_integer(*x*) ((int) rint((x)))

11.128.2.14 #define get_integer(x) ((int) rint((x)))

11.128.2.15 #define get_integer(x) ((int) rint((x)))

11.128.2.16 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.17 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.18 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

This macro checks whether a double number represents an integer.

Definition at line 53 of file search_graph.cc.

11.128.2.19 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.20 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.21 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.22 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.23 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.24 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.25 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.26 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.27 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.28 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

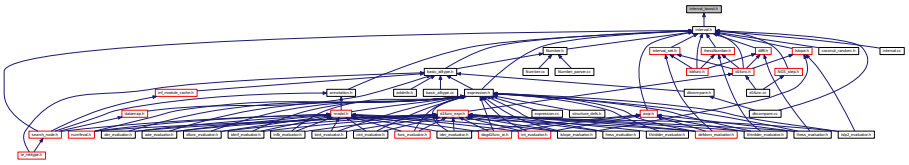
11.128.2.29 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.128.2.30 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

11.129 interval_boost.h File Reference

`#include <iostream>` `#include <values.h>` `#include <boost/numeric/interval.-hpp>` `#include <boost/numeric/interval/checking.hpp>` `#include <boost/numeric/interval/hpp>` `#include <boost/numeric/interval/policies.hpp>` Include dependency graph for interval_boost.h:

This graph shows which files directly or indirectly include this file:



Classes

- struct coco::checking_my
- struct coco::my_rounded_math
- struct coco::interval_st

  Constructor-free interval.
- class coco::interval

  Interval wrapper class.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()
- #define coconut_init_interval()

Typedefs

- typedef boost::numeric::interval_lib::policies < my_rounded_math< double > , checking_my< double >> coco::my_policies

Functions

- double coco::safeguarded_mid (const interval &__i)
- interval coco::ipow (const interval &x, int n)
- interval coco::gauss (const interval &x)
- interval coco::atan2 (const interval &y, const interval &x)
- double coco::absmin (const interval &__i)
- double coco::gainfactor (const interval &_old, const interval &_new)
- bool coco::operator== (const interval &a, const interval &b)
- bool coco::operator== (const interval &a, double b)
- bool coco::operator!= (const interval &a, const interval &b)
- bool coco::operator!= (const interval &a, double b)
- bool coco::operator< (const interval &a, const interval &b)
- interval coco::operator+ (const interval &a, const interval &b)
- interval coco::operator- (const interval &a, const interval &b)
- interval coco::cancel (const interval &a, const interval &b)
- interval coco::operator∗ (const interval &a, const interval &b)
- interval coco::operator/ (const interval &a, const interval &b)
- std::ostream & coco::operator<< (std::ostream &s, const interval &a)
- double coco::mid (const interval &)
- double coco::diam (const interval &)
- double coco::width (const interval &)
- double coco::relDiam (const interval &)
- double coco::rad (const interval &)
- double coco::mig (const interval &)
- double coco::mag (const interval &)
- double coco::dist (const interval &x, const interval &y)
- interval coco::round_to_integer (const interval &x)
- interval coco::acos (const interval &x)
- interval coco::abs (const interval &x)
- interval coco::acosh (const interval &x)
- interval coco::acot (const interval &x)
- interval coco::acoth (const interval &x)
- interval coco::asin (const interval &x)
- interval coco::asinh (const interval &x)
- interval coco::atan (const interval &x)
- interval coco::atanh (const interval &x)
- interval coco::cos (const interval &x)
- interval coco::cosh (const interval &x)
- interval coco::cot (const interval &x)
- interval coco::coth (const interval &x)
- interval coco::exp (const interval &x)
- interval coco::exp10 (const interval &x)
- interval coco::exp2 (const interval &x)
- interval coco::expm1 (const interval &x)
- interval coco::log (const interval &x)
- interval coco::log10 (const interval &x)
- interval coco::log1p (const interval &x)
- interval coco::log2 (const interval &x)
- interval coco::power (const interval &x, int n)

- interval coco::pow (const interval &x, const interval &y)
- interval coco::sin (const interval &x)
- interval coco::sinh (const interval &x)
- interval coco::sqr (const interval &x)
- interval coco::sqrt (const interval &x)
- interval coco::tan (const interval &x)
- interval coco::tanh (const interval &x)
- interval coco::imax (const interval &x, const interval &y)
- interval coco::imin (const interval &x, const interval &y)
- interval coco::division_part1 (const interval &x, const interval &y, bool &b)

- template<class _TC >
  bool coco::operator== (const interval &__i, const _TC &__d)

- template<class _TC >
  bool coco::operator!= (const interval &__i, const _TC &__d)

- bool coco::operator< (const interval &a, double b)
- bool coco::operator< (double a, const interval &b)

- interval coco::operator+ (const interval &a, double b)
- interval coco::operator+ (double b, const interval &a)

- interval coco::operator- (const interval &a, double b)
- interval coco::operator- (double b, const interval &a)

- interval coco::operator* (const interval &a, double b)
- interval coco::operator* (double b, const interval &a)

- interval coco::operator/ (const interval &a, double b)
- interval coco::operator/ (double b, const interval &a)

- interval coco::division_part2 (const interval &x, const interval &y, bool b=true)

11.129.1 Detailed Description

Definition in file interval_boost.h.

11.129.2 Define Documentation

11.129.2.1 #define coconut_init_interval( )

11.129.2.2 #define coconut_init_interval( )

11.129.2.3 #define coconut_init_interval( )

11.129.2.4 #define coconut_init_interval( )

11.129.2.5 #define coconut_init_interval( )

11.129.2.6 #define coconut_init_interval( )

11.129.2.7 #define coconut_init_interval( )

11.129.2.8 #define coconut_init_interval( )

11.129.2.9 #define coconut_init_interval( )

11.129.2.10 #define coconut_init_interval( )

11.129.2.11 #define coconut_init_interval( )

11.129.2.12 #define coconut_init_interval( )

11.129.2.13 #define coconut_init_interval( )

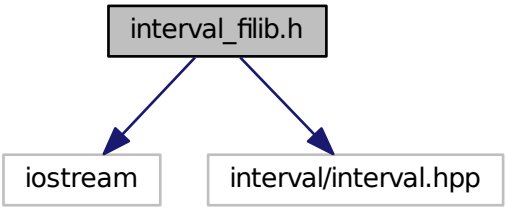
11.129.2.14 #define coconut_init_interval( )

Definition at line 46 of file search_graph.cc.

11.129.2.15 #define coconut_init_interval( )

11.130 interval_filib.h File Reference

`#include <iostream>` `#include <interval/interval.hpp>` Include dependency graph for interval_filib.h:



Classes

- struct coco::interval_st

  Constructor-free interval.
- class coco::interval

  Interval wrapper class.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define FILIB_USE_MACROS 0
- #define FILIB_EXTENDED 1
- #define coconut_init_interval() filib::fp_traits<double>::setup()

Functions

- double coco::safeguarded_mid (const interval &__i)
- interval coco::ipow (const interval &x, int n)
- interval coco::gauss (const interval &x)
- interval coco::atan2 (const interval &y, const interval &x)
- double coco::absmin (const interval &__i)
- double coco::gainfactor (const interval &_old, const interval &_new)
- bool coco::operator== (const interval &a, const interval &b)
- bool coco::operator!= (const interval &a, const interval &b)

- double coco::downward_plus (const double &a, const double &b)
- double coco::upward_plus (const double &a, const double &b)
- double coco::downward_minus (const double &a, const double &b)
- double coco::upward_minus (const double &a, const double &b)
- double coco::downward_multiplies (const double &a, const double &b)
- double coco::upward_multiplies (const double &a, const double &b)
- double coco::downward_divides (const double &a, const double &b)
- double coco::upward_divides (const double &a, const double &b)
- bool coco::operator< (const interval &a, const interval &b)
- interval coco::operator+ (const interval &a, const interval &b)
- interval coco::operator- (const interval &a, const interval &b)
- interval coco::cancel (const interval &a, const interval &b)
- interval coco::operator* (const interval &a, const interval &b)
- interval coco::operator/ (const interval &a, const interval &b)
- interval coco::division_part1 (const interval &x, const interval &y, bool &b)
- double coco::mid (const interval &)
- double coco::diam (const interval &)
- double coco::width (const interval &)
- double coco::relDiam (const interval &)
- double coco::rad (const interval &)
- double coco::mig (const interval &)
- double coco::mag (const interval &)
- double coco::dist (const interval &x, const interval &y)
- interval coco::round_to_integer (const interval &x)
- interval coco::acos (const interval &x)
- interval coco::abs (const interval &x)
- interval coco::acosh (const interval &x)
- interval coco::acot (const interval &x)
- interval coco::acoth (const interval &x)
- interval coco::asin (const interval &x)
- interval coco::asinh (const interval &x)
- interval coco::atan (const interval &x)
- interval coco::atanh (const interval &x)
- interval coco::cos (const interval &x)
- interval coco::cosh (const interval &x)
- interval coco::cot (const interval &x)
- interval coco::coth (const interval &x)
- interval coco::exp (const interval &x)
- interval coco::exp10 (const interval &x)
- interval coco::exp2 (const interval &x)
- interval coco::expm1 (const interval &x)
- interval coco::log (const interval &x)
- interval coco::log10 (const interval &x)
- interval coco::log1p (const interval &x)
- interval coco::log2 (const interval &x)
- interval coco::power (const interval &x, int n)
- interval coco::pow (const interval &x, const interval &y)
- interval coco::sin (const interval &x)
- interval coco::sinh (const interval &x)
- interval coco::sqr (const interval &x)

- interval coco::sqrt (const interval &x)
- interval coco::tan (const interval &x)
- interval coco::tanh (const interval &x)
- interval coco::imax (const interval &x, const interval &y)
- interval coco::imin (const interval &x, const interval &y)
- interval coco::intersect (const interval &__x, const interval &__y)
- interval coco::hull (const interval &__x, const interval &__y)
- interval coco::hulldiff (const interval &__x, const interval &__y)
- std::ostream & coco::operator<< (std::ostream &s, const interval &a)

- template<class _TC >
  bool coco::operator== (const interval &__i, const _TC &__d)

- template<class _TC >
  bool coco::operator!= (const interval &__i, const _TC &__d)

- bool coco::operator< (const interval &a, double b)
- bool coco::operator< (double a, const interval &b)

- bool coco::operator> (const interval &a, const interval &b)
- bool coco::operator> (const interval &a, double b)
- bool coco::operator> (double a, const interval &b)

- bool coco::operator<= (const interval &a, const interval &b)
- bool coco::operator<= (const interval &a, double b)
- bool coco::operator<= (double a, const interval &b)

- bool coco::operator>= (const interval &a, const interval &b)
- bool coco::operator>= (const interval &a, double b)
- bool coco::operator>= (double a, const interval &b)

- bool coco::lexicographic_lt (const interval &a, const interval &b)

- bool coco::lexicographic_le (const interval &a, const interval &b)

- interval coco::operator+ (const interval &a, double b)
- interval coco::operator+ (double b, const interval &a)

- interval coco::operator- (const interval &a, double b)
- interval coco::operator- (double b, const interval &a)

- interval coco::operator∗ (const interval &a, double b)
- interval coco::operator∗ (double b, const interval &a)

- interval coco::operator/ (const interval &a, double b)
- interval coco::operator/ (double b, const interval &a)

- interval coco::division_part2 (const interval &x, const interval &y, bool b=true)

11.130.1 Detailed Description

Definition in file interval_filib.h.

11.130.2 Define Documentation

11.130.2.1 #define coconut_init_interval( ) filib::fp_traits<double>::setup()

Definition at line 45 of file interval_filib.h.

11.130.2.2 #define FILIB_EXTENDED 1

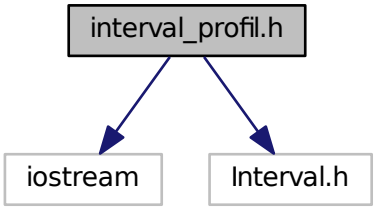
Definition at line 32 of file interval_filib.h.

11.130.2.3 #define FILIB_USE_MACROS 0

Definition at line 31 of file interval_filib.h.

11.131 interval_profil.h File Reference

`#include <iostream>` `#include <Interval.h>` Include dependency graph for interval_profil.h-
:



Classes

- struct coco::interval_st

  Constructor-free interval.
- class coco::interval

  Interval wrapper class.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define coconut_init_interval()

Functions

- double coco::safeguarded_mid (const interval &__i)
- interval coco::ipow (const interval &x, int n)
- interval coco::gauss (const interval &x)
- interval coco::atan2 (const interval &y, const interval &x)
- double coco::absmin (const interval &__i)
- double coco::gainfactor (const interval &_old, const interval &_new)
- bool coco::operator== (const interval &a, const interval &b)
- bool coco::operator!= (const interval &a, const interval &b)
- double coco::downward_plus (const double &a, const double &b)
- double coco::upward_plus (const double &a, const double &b)

- double coco::downward_minus (const double &a, const double &b)
- double coco::upward_minus (const double &a, const double &b)
- double coco::downward_multiplies (const double &a, const double &b)
- double coco::upward_multiplies (const double &a, const double &b)
- double coco::downward_divides (const double &a, const double &b)
- double coco::upward_divides (const double &a, const double &b)
- bool coco::operator< (const interval &a, const interval &b)
- interval coco::operator+ (const interval &a, const interval &b)
- interval coco::operator- (const interval &a, const interval &b)
- interval coco::cancel (const interval &a, const interval &b)
- interval coco::operator∗ (const interval &a, const interval &b)
- interval coco::operator/ (const interval &a, const interval &b)
- interval coco::division_part1 (const interval &x, const interval &y, bool &b)
- double coco::mid (const interval &)
- double coco::diam (const interval &)
- double coco::width (const interval &)
- double coco::relDiam (const interval &)
- double coco::rad (const interval &)
- double coco::mig (const interval &)
- double coco::mag (const interval &)
- double coco::dist (const interval &x, const interval &y)
- interval coco::acos (const interval &x)
- interval coco::abs (const interval &x)
- interval coco::acosh (const interval &x)
- interval coco::acot (const interval &x)
- interval coco::acoth (const interval &x)
- interval coco::asin (const interval &x)
- interval coco::asinh (const interval &x)
- interval coco::atan (const interval &x)
- interval coco::atanh (const interval &x)
- interval coco::cos (const interval &x)
- interval coco::cosh (const interval &x)
- interval coco::cot (const interval &x)
- interval coco::coth (const interval &x)
- interval coco::exp (const interval &x)
- interval coco::exp10 (const interval &x)
- interval coco::exp2 (const interval &x)
- interval coco::expm1 (const interval &x)
- interval coco::log (const interval &x)
- interval coco::log10 (const interval &x)
- interval coco::log1p (const interval &x)
- interval coco::log2 (const interval &x)
- interval coco::power (const interval &x, int n)
- interval coco::pow (const interval &x, const interval &y)
- interval coco::sin (const interval &x)
- interval coco::sinh (const interval &x)
- interval coco::sqr (const interval &x)
- interval coco::sqrt (const interval &x)
- interval coco::tan (const interval &x)
- interval coco::tanh (const interval &x)
- interval coco::imax (const interval &x, const interval &y)
- interval coco::imin (const interval &x, const interval &y)

- template<class _TC >
  bool coco::operator== (const interval &__i, const _TC &__d)

- template<class _TC >
  bool coco::operator!= (const interval &__i, const _TC &__d)

- bool coco::operator< (const interval &a, double b)
- bool coco::operator< (double a, const interval &b)

- interval coco::operator+ (const interval &a, double b)
- interval coco::operator+ (double b, const interval &a)

- interval coco::operator- (const interval &a, double b)
- interval coco::operator- (double b, const interval &a)

- interval coco::operator* (const interval &a, double b)
- interval coco::operator* (double b, const interval &a)

- interval coco::operator/ (const interval &a, double b)
- interval coco::operator/ (double b, const interval &a)

- interval coco::division_part2 (const interval &x, const interval &y, bool b=true)

11.131.1 Detailed Description

Definition in file interval_profil.h.

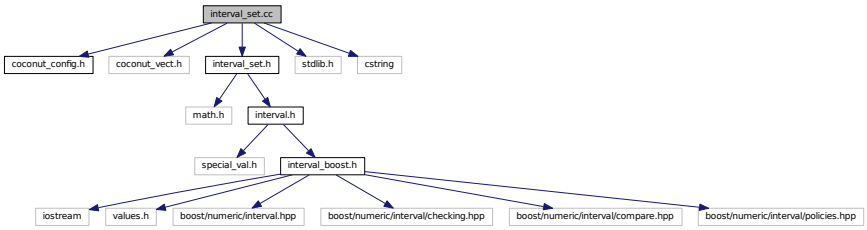
11.131.2 Define Documentation

11.131.2.1 #define coconut_init_interval( )

Definition at line 37 of file interval_profil.h.

11.132 interval_set.cc File Reference

`#include <coconut_config.h>` `#include <coconut_vect.h>` `#include <interval-_set.h>` `#include <stdlib.h>` `#include <cstring>` Include dependency graph for interval-_set.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API
- namespace std

  The standard namespace.

Defines

- #define INTERVAL_SET_DEBUG 0

Functions

- double coco::width (const interval_set &a)
- double coco::volume (const interval_set &a)
- interval_set coco::divide (const interval &a, const interval &b)
- bool coco::operator== (const interval_set &a, const interval_set &b)
- interval_set coco::imin (const interval_set &a, const interval_set &b)
- interval_set coco::imax (const interval_set &a, const interval_set &b)
- interval_set coco::power (const interval_set &a, int n)
- interval_set coco::sqr (const interval_set &a)
- interval_set coco::sqrt (const interval_set &a)
- interval_set coco::exp (const interval_set &a)
- interval_set coco::exp10 (const interval_set &a)
- interval_set coco::exp2 (const interval_set &a)
- interval_set coco::expm1 (const interval_set &a)
- interval_set coco::log (const interval_set &a)
- interval_set coco::log10 (const interval_set &a)
- interval_set coco::log1p (const interval_set &a)
- interval_set coco::log2 (const interval_set &a)
- interval_set coco::abs (const interval_set &a)

- interval_set coco::sin (const interval_set &a)
- interval_set coco::cos (const interval_set &a)
- interval_set coco::cot (const interval_set &a)
- interval_set coco::tan (const interval_set &a)
- interval_set coco::asin (const interval_set &a)
- interval_set coco::acos (const interval_set &a)
- interval_set coco::acot (const interval_set &a)
- interval_set coco::atan (const interval_set &a)
- interval_set coco::sinh (const interval_set &a)
- interval_set coco::cosh (const interval_set &a)
- interval_set coco::coth (const interval_set &a)
- interval_set coco::tanh (const interval_set &a)
- interval_set coco::asinh (const interval_set &a)
- interval_set coco::acosh (const interval_set &a)
- interval_set coco::acoth (const interval_set &a)
- interval_set coco::atanh (const interval_set &a)
- interval_set coco::pow (const interval_set &a, const interval_set &b)
- interval_set coco::atan2 (const interval_set &a, const interval_set &b)
- interval_set coco::round_to_integer (const interval_set &a)
- ostream & std::operator<< (ostream &s, const coco::interval_set &a)

11.132.1 Detailed Description

Definition in file interval_set.cc.

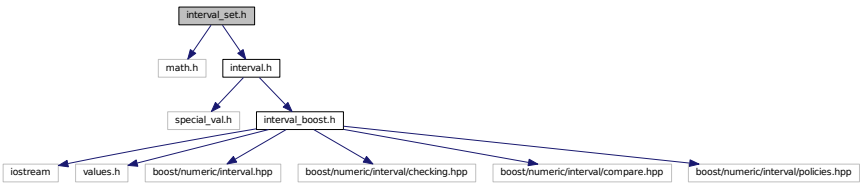
11.132.2 Define Documentation

11.132.2.1 #define INTERVAL_SET_DEBUG 0

Definition at line 39 of file interval_set.cc.

11.133 interval_set.h File Reference

`#include <math.h>` `#include "interval.h"` Include dependency graph for interval_set.h:



This graph shows which files directly or indirectly include this file:

Classes

- class coco::interval_set

Namespaces

- namespace coco

  the main namespace of the COCONUT API
- namespace std

  The standard namespace.

Defines

- #define SET_BASE_LENGTH 5
- #define SET_EXT 5
- #define SET_DEF_MAX_LENGTH 20
- #define SET_DEF_FILL 2

Functions

- double coco::width (const interval_set &a)
- double coco::volume (const interval_set &a)
- interval_set coco::operator+ (const interval_set &a, const interval_set &b)
- interval_set coco::operator+ (const interval_set &a, const interval &b)
- interval_set coco::operator+ (const interval &a, const interval_set &b)
- interval_set coco::operator+ (const interval_set &a, double b)
- interval_set coco::operator+ (double a, const interval_set &b)
- interval_set coco::operator- (const interval_set &a, const interval_set &b)
- interval_set coco::operator- (const interval_set &a, const interval &b)
- interval_set coco::operator- (const interval &a, const interval_set &b)
- interval_set coco::operator- (const interval_set &a, double b)
- interval_set coco::operator- (double a, const interval_set &b)
- interval_set coco::operator- (const interval_set &a)
- interval_set coco::operator∗ (const interval_set &a, const interval_set &b)
- interval_set coco::operator∗ (const interval_set &a, const interval &b)
- interval_set coco::operator∗ (const interval &a, const interval_set &b)
- interval_set coco::operator∗ (const interval_set &a, double b)
- interval_set coco::operator∗ (double a, const interval_set &b)
- interval_set coco::divide (const interval &a, const interval &b)
- interval_set coco::divide (const interval &a, double b)

- interval_set coco::divide (double a, const interval &b)
- interval_set coco::operator/ (const interval_set &a, const interval_set &b)
- interval_set coco::operator/ (const interval_set &a, const interval &b)
- interval_set coco::operator/ (const interval &a, const interval_set &b)
- interval_set coco::operator/ (const interval_set &a, double b)
- interval_set coco::operator/ (double a, const interval_set &b)
- bool coco::operator== (const interval_set &a, const interval_set &b)
- bool coco::operator!= (const interval_set &a, const interval_set &b)
- interval_set coco::imin (const interval_set &a, const interval_set &b)
- interval_set coco::imax (const interval_set &a, const interval_set &b)
- interval_set coco::power (const interval_set &a, int n)
- interval_set coco::sqr (const interval_set &a)
- interval_set coco::sqrt (const interval_set &a)
- interval_set coco::exp (const interval_set &a)
- interval_set coco::exp10 (const interval_set &a)
- interval_set coco::exp2 (const interval_set &a)
- interval_set coco::expm1 (const interval_set &a)
- interval_set coco::log (const interval_set &a)
- interval_set coco::log10 (const interval_set &a)
- interval_set coco::log1p (const interval_set &a)
- interval_set coco::log2 (const interval_set &a)
- interval_set coco::abs (const interval_set &a)
- interval_set coco::sin (const interval_set &a)
- interval_set coco::cos (const interval_set &a)
- interval_set coco::cot (const interval_set &a)
- interval_set coco::tan (const interval_set &a)
- interval_set coco::asin (const interval_set &a)
- interval_set coco::acos (const interval_set &a)
- interval_set coco::acot (const interval_set &a)
- interval_set coco::atan (const interval_set &a)
- interval_set coco::sinh (const interval_set &a)
- interval_set coco::cosh (const interval_set &a)
- interval_set coco::coth (const interval_set &a)
- interval_set coco::tanh (const interval_set &a)
- interval_set coco::asinh (const interval_set &a)
- interval_set coco::acosh (const interval_set &a)
- interval_set coco::acoth (const interval_set &a)
- interval_set coco::atanh (const interval_set &a)
- interval_set coco::pow (const interval_set &a, const interval_set &b)
- interval_set coco::atan2 (const interval_set &a, const interval_set &b)
- interval_set coco::round_to_integer (const interval_set &a)
- ostream & std::operator<< (ostream &s, const coco::interval_set &a)

11.133.1 Detailed Description

Definition in file interval_set.h.

11.133.2 Define Documentation

11.133.2.1 #define SET_BASE_LENGTH 5

Definition at line 37 of file interval_set.h.

11.133.2.2 #define SET_DEF_FILL 2

Definition at line 40 of file interval_set.h.

11.133.2.3 #define SET_DEF_MAX_LENGTH 20

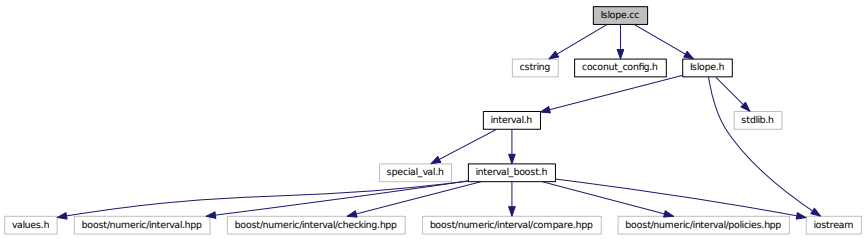
Definition at line 39 of file interval_set.h.

11.133.2.4 #define SET_EXT 5

Definition at line 38 of file interval_set.h.

11.134 Islope.cc File Reference

`#include <cstring>` `#include <coconut_config.h>` `#include <Islope.h>` Include dependency graph for Islope.cc:



Typedefs

- typedef interval I

Functions

- Islope exp (const Islope &a)
- Islope sin (const Islope &a)
- Islope cos (const Islope &a)
- Islope sqr (const Islope &a)
- Islope sqrt (const Islope &a)
- Islope pow (const Islope &a, const Islope &b)
- Islope power (const Islope &a, int n)
- Islope log (const Islope &a)

- Islope atan (const Islope &a)
- Islope atan2 (const Islope &a, const Islope &b)

11.134.1 Typedef Documentation

11.134.1.1 typedef interval I

Definition at line 7 of file Islope.cc.

11.134.2 Function Documentation

11.134.2.1 Islope atan (const Islope & *a*)

Used to compute the slope for the function atan(x) for a given x.

Definition at line 933 of file Islope.cc.

11.134.2.2 Islope atan2 (const Islope & *a*, const Islope & *b*)

Used to compute the slope for the function atan2(x) for a given x.

Definition at line 1022 of file Islope.cc.

11.134.2.3 Islope cos (const Islope & *a*)

Used to compute the slope for the function cos(x) for a given x.

Definition at line 308 of file Islope.cc.

11.134.2.4 Islope exp (const Islope & *a*)

Used to compute the slope for the function exp(x) for a given x.

Definition at line 14 of file Islope.cc.

11.134.2.5 Islope log (const Islope & *a*) [inline]

Used to compute the slope for the function log(x) for a given x.

Definition at line 848 of file Islope.cc.

11.134.2.6 Islope pow (const Islope & *a*, const Islope & *b*)

Used to compute the slope for the function pow(x,y) for given x, y.

Definition at line 593 of file Islope.cc.

11.134.2.7 Islope power (const Islope & *a*, int *n*)

Used to compute the slope for the function power(x,n) for a given x.

Definition at line 607 of file Islope.cc.

11.134.2.8 Islope sin (const Islope & *a*)

Used to compute the slope for the function sin(x) for a given x.

Definition at line 95 of file Islope.cc.

11.134.2.9 Islope sqr (const Islope & *a*)

Used to compute the slope for the function sqr(x) for a given x.

Definition at line 521 of file Islope.cc.

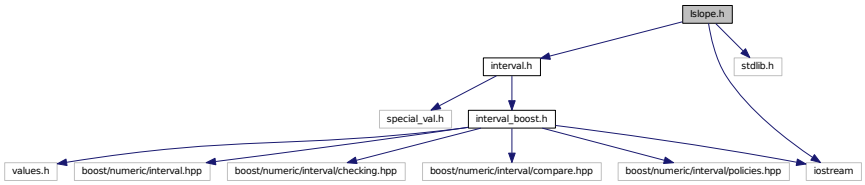
11.134.2.10 Islope sqrt (const Islope & *a*)

Used to compute the slope for the function sqrt(x) for a given x.

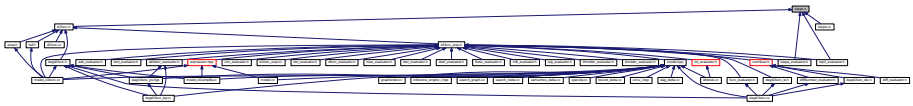
Definition at line 551 of file Islope.cc.

11.135 Islope.h File Reference

`#include "interval.h"` `#include <stdlib.h>` `#include <iostream>` Include dependency graph for Islope.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Islope

  Class for computing intervalslopes up to order 2.

Typedefs

- typedef interval I

Functions

- Islope operator+ (const Islope &a, const Islope &b)
- Islope operator+ (const double &a, const Islope &b)
- Islope operator+ (const Islope &a, const double &b)
- Islope operator- (const Islope &a, const Islope &b)
- Islope operator- (const double &a, const Islope &b)
- Islope operator- (const Islope &a, const double &b)
- Islope operator- (const Islope &a)
- Islope operator∗ (const Islope &a, const Islope &b)
- Islope operator∗ (const double &a, const Islope &b)
- Islope operator∗ (const Islope &a, const double &b)
- Islope operator/ (const Islope &a, const Islope &b)
- Islope operator/ (const double &a, const Islope &b)
- Islope operator/ (const Islope &a, const double &b)
- std::ostream & operator<< (std::ostream &s, const Islope &a)
- Islope exp (const Islope &a)
- Islope sin (const Islope &a)
- Islope cos (const Islope &a)
- Islope sqrt (const Islope &a)
- Islope sqr (const Islope &a)
- Islope power (const Islope &a, int n)
- Islope pow (const Islope &a, const Islope &b)
- Islope log (const Islope &a)
- Islope atan (const Islope &a)
- Islope atan2 (const Islope &a, const Islope &b)

11.135.1 Typedef Documentation

11.135.1.1 typedef interval I

Definition at line 10 of file Islope.h.

11.135.2 Function Documentation

11.135.2.1 Islope atan (const Islope & *a*)

Used to compute the slope for the function atan(x) for a given x.

Definition at line 933 of file Islope.cc.

11.135.2.2 Islope atan2 (const Islope & *a*, const Islope & *b*)

Used to compute the slope for the function atan2(x) for a given x.

Definition at line 1022 of file Islope.cc.

11.135.2.3 Islope cos (const Islope & *a*)

Used to compute the slope for the function cos(x) for a given x.

Definition at line 308 of file Islope.cc.

11.135.2.4 Islope exp (const Islope & *a*)

Used to compute the slope for the function exp(x) for a given x.

Definition at line 14 of file Islope.cc.

11.135.2.5 Islope log (const Islope & *a*) [inline]

Used to compute the slope for the function log(x) for a given x.

Definition at line 848 of file Islope.cc.

11.135.2.6 Islope operator∗ (const Islope & *a*, const Islope & *b*) [inline]

Operator ∗. Multiplicates slope a with slope b.

Definition at line 653 of file Islope.h.

11.135.2.7 Islope operator∗ (const double & *a*, const Islope & *b*) [inline]

Operator ∗. Multiplicates double a with slope b.

Definition at line 689 of file Islope.h.

11.135.2.8 Islope operator∗ (const Islope & *a*, const double & *b*) [inline]

Operator ∗. Multiplicates slope a with double b.

Definition at line 716 of file Islope.h.

11.135.2.9 Islope operator+ (const Islope & *a*, const Islope & *b*) [inline]

Operator +. Adds slope a and b.

Definition at line 446 of file Islope.h.

11.135.2.10 Islope operator+ (const double & *a*, const Islope & *b*) [inline]

Operator +. Adds slope b and double a.

Definition at line 482 of file Islope.h.

11.135.2.11 Islope operator+ (const Islope & *a*, const double & *b*) [inline]

Operator +. Adds slope a and double b.

Definition at line 509 of file Islope.h.

11.135.2.12 Islope operator- (const Islope & *a*, const Islope & *b*) [inline]

Operator -. Subtracts slope b from slope a.

Definition at line 536 of file Islope.h.

11.135.2.13 Islope operator- (const double & *a*, const Islope & *b*) [inline]

Operator -. Subtracts slope b from daouble a.

Definition at line 572 of file Islope.h.

11.135.2.14 Islope operator- (const Islope & *a*, const double & *b*) [inline]

Operator -. Subtracts double b from slope a.

Definition at line 599 of file Islope.h.

11.135.2.15 Islope operator- (const Islope & *a*) [inline]

Operator -. Changes sign of slope a.

Definition at line 626 of file Islope.h.

11.135.2.16 Islope operator/ (const Islope & *a*, const Islope & *b*) [inline]

Operator /. Divides the slope a by slope b.

Definition at line 758 of file Islope.h.

11.135.2.17 Islope operator/ (const double & *a*, const Islope & *b*) [inline]

Operator /. Divides the double a by slope b.

Definition at line 743 of file Islope.h.

11.135.2.18 Islope operator/ (const Islope & *a*, const double & *b*) [inline]

Operator /. Divides the slope a by double b.

Definition at line 790 of file Islope.h.

11.135.2.19 std::ostream& operator<< (std::ostream & *s*, const Islope & *a*)

The operator for ouput. Shows the range of the function, the functionvalue of the middle point and the slopeinterval. If the slope has order 2 the functionvalue of the first derivative of the function at the middle-point and the slopeinterval of order 2 are also shown.

11.135.2.20 Islope pow (const Islope & *a*, const Islope & *b*)

Used to compute the slope for the function pow(x,y) for given x, y.

Definition at line 593 of file Islope.cc.

11.135.2.21 Islope power (const Islope & *a*, int *n*)

Used to compute the slope for the function power(x,n) for a given x.

Definition at line 607 of file Islope.cc.

11.135.2.22 Islope sin (const Islope & *a*)

Used to compute the slope for the function sin(x) for a given x.

Definition at line 95 of file Islope.cc.

11.135.2.23 Islope sqr (const Islope & *a*)

Used to compute the slope for the function sqr(x) for a given x.

Definition at line 521 of file Islope.cc.

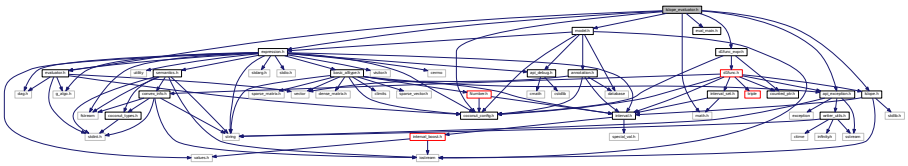
11.135.2.24 Islope sqrt (const Islope & *a*)

Used to compute the slope for the function sqrt(x) for a given x.

Definition at line 551 of file Islope.cc.

11.136 Islope_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <model.h>` `#include <eval_main.h>` `#include <math.h>` `#include <api_exception.h>` `#include <d1func_expr.h>` `#include <Islope.h>` Include dependency graph for Islope_evaluator.h:



Classes

- struct coco::Islope_eval_type
  Visitor data for Islope_eval.
- class coco::Islope_eval
  Forward function range evaluation.

Namespaces

- namespace coco
  the main namespace of the COCONUT API

Typedefs

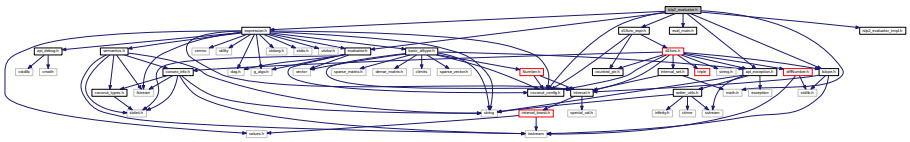
- typedef Islope(∗ coco::Islope_evaluator)(const std::vector< Islope > ∗__x, const variable_indicator &__v)

11.136.1 Detailed Description

Definition in file Islope_evaluator.h.

11.137 islp2_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <eval_main.h>` `#include <math.h>` `#include <api_exception.h>` × `#include <d1func_expr.h>` `#include <Islope.h>` `#include <islp2_evaluator-_tmpl.h>` Include dependency graph for islp2_evaluator.h:



Classes

- struct coco::prep_islp2_eval_type
- class coco::prep_islp2_eval
- struct coco::func_islp2_eval_ret_type
- struct coco::func_islp2_eval_type
- class coco::func_islp2_eval
- struct coco::islp2_eval_type_INTERNAL
- class coco::islp2_eval_INTERNAL
- struct coco::islp2_eval_type_INTERNAL
- class coco::islp2_eval_INTERNAL

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DEBUG_ISLP2_EVALUATOR 0
- #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0
- #define islp2_eval_INTERNAL islp2_eval
- #define islp2_eval_type_INTERNAL islp2_eval_type
- #define ISLP2_WITH_INTERMEDIATE_NODES 0
- #define islp2_eval_INTERNAL islp2_eval_in
- #define islp2_eval_type_INTERNAL islp2_eval_type_in
- #define ISLP2_WITH_INTERMEDIATE_NODES 1

Functions

- std::ostream & coco::operator<< (std::ostream &o, const func_islp2_eval_ret_type &r)

11.137.1 Detailed Description

Definition in file islp2_evaluator.h.

11.137.2 Define Documentation

11.137.2.1 #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

Definition at line 31 of file islp2_evaluator.h.

11.137.2.2 #define DEBUG_ISLP2_EVALUATOR 0

Definition at line 30 of file islp2_evaluator.h.

11.137.2.3 #define islp2_eval_INTERNAL islp2_eval

Definition at line 1455 of file islp2_evaluator.h.

11.137.2.4 #define islp2_eval_INTERNAL islp2_eval_in

Definition at line 1455 of file islp2_evaluator.h.

11.137.2.5 #define islp2_eval_type_INTERNAL islp2_eval_type

Definition at line 1456 of file islp2_evaluator.h.

11.137.2.6 #define islp2_eval_type_INTERNAL islp2_eval_type_in

Definition at line 1456 of file islp2_evaluator.h.

11.137.2.7 #define ISLP2_WITH_INTERMEDIATE_NODES 0

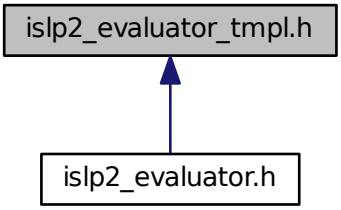
Definition at line 1457 of file islp2_evaluator.h.

11.137.2.8 #define ISLP2_WITH_INTERMEDIATE_NODES 1

Definition at line 1457 of file islp2_evaluator.h.

11.138 islp2_evaluator_tmpl.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

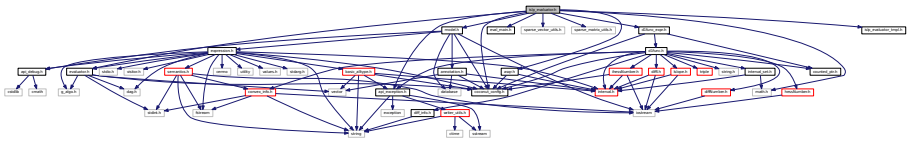
- struct islp2_eval_type_INTERNAL
- class islp2_eval_INTERNAL

11.138.1 Detailed Description

Definition in file islp2_evaluator_tmpl.h.

11.139 islp_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <model.h>` × `#include <eval_main.h>` `#include <sparse_vector_utils.h>` `#include <sparse-_matrix_utils.h>` `#include <math.h>` `#include <d1func_expr.h>` `#include <api-_exception.h>` `#include <asqr.h>` `#include <islp_evaluator_tmpl.h>` Include dependency graph for islp_evaluator.h:



Classes

- struct coco::func_islp_return_type

  The return type of the func_islp_eval evaluator.

- class coco::prep_islp_eval

  Preparation Evaluator for first order slopes.
- struct coco::func_islp_eval_type

  Visitor data for func_id_eval.
- class coco::func_islp_eval

  Forward function range evaluation with preparation of first order slope data.
- struct coco::islp_eval_type_INTERNAL

  Visitor data for islp_eval.
- class coco::islp_eval_INTERNAL

  Backward first order slope evaluation with prepared first order slope data.
- struct coco::islp_eval_type_INTERNAL

  Visitor data for islp_eval.
- class coco::islp_eval_INTERNAL

  Backward first order slope evaluation with prepared first order slope data.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DEBUG_ISLP_EVAL 0
- #define islp_eval_INTERNAL islp_eval
- #define islp_eval_type_INTERNAL islp_eval_type
- #define ISLP_WITH_INTERMEDIATE_NODES 0
- #define islp_eval_INTERNAL islp_eval_in
- #define islp_eval_type_INTERNAL islp_eval_type_in
- #define ISLP_WITH_INTERMEDIATE_NODES 1

Typedefs

- typedef bool(∗ coco::prep_islp_evaluator)()
- typedef func_islp_return_type(∗ coco::func_islp_evaluator)(const std::vector< interval > ∗__x, const variable_indicator &__v, std::vector< interval > &__islp_data)
- typedef std::vector< interval > &(∗ coco::islp_evaluator)(const std::vector< interval > &__d_dat, const variable_indicator &__v)

11.139.1 Detailed Description

Definition in file islp_evaluator.h.

11.139.2 Define Documentation

11.139.2.1 #define DEBUG_ISLP_EVAL 0

Definition at line 46 of file islp_evaluator.h.

11.139.2.2 #define islp_eval_INTERNAL islp_eval

Definition at line 1545 of file islp_evaluator.h.

11.139.2.3 #define islp_eval_INTERNAL islp_eval_in

Definition at line 1545 of file islp_evaluator.h.

11.139.2.4 #define islp_eval_type_INTERNAL islp_eval_type

Definition at line 1546 of file islp_evaluator.h.

11.139.2.5 #define islp_eval_type_INTERNAL islp_eval_type_in

Definition at line 1546 of file islp_evaluator.h.

11.139.2.6 #define ISLP_WITH_INTERMEDIATE_NODES 0

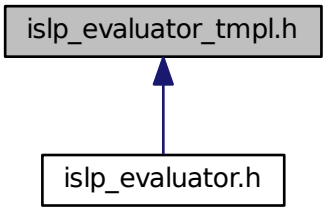
Definition at line 1547 of file islp_evaluator.h.

11.139.2.7 #define ISLP_WITH_INTERMEDIATE_NODES 1

Definition at line 1547 of file islp_evaluator.h.

11.140 islp_evaluator_tmpl.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct islp_eval_type_INTERNAL

  Visitor data for islp_eval.
- class islp_eval_INTERNAL

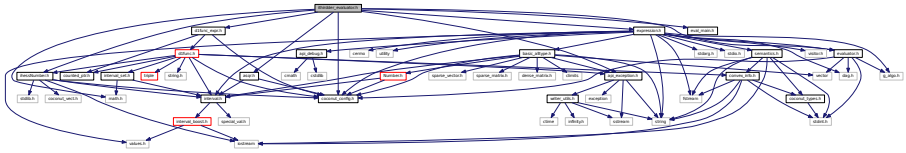
  Backward first order slope evaluation with prepared first order slope data.

11.140.1 Detailed Description

Definition in file islp_evaluator_tmpl.h.

11.141 ithirdder_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <eval_main.h>` `#include <math.h>` `#include <api_exception.h>` × `#include <d1func_expr.h>` `#include <ihessNumber.h>` `#include <asqr.h>` Include dependency graph for ithirdder_evaluator.h:



Classes

- struct coco::ithirdderPreparationEvaluatorType
- class coco::ithirdderPreparationEvaluator
- struct coco::ithirdderForwardEvaluatorReturnValue
- struct coco::ithirdderForwardEvaluatorType
- class coco::ithirdderForwardEvaluator
- struct coco::ithirdderBackwardEvaluatorType
- class coco::ithirdderBackwardEvaluator

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DEBUG_ITHIRDDER_EVALUATOR 0
- #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

11.141.1 Detailed Description

Definition in file ithirdder_evaluator.h.

11.141.2 Define Documentation

11.141.2.1 #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

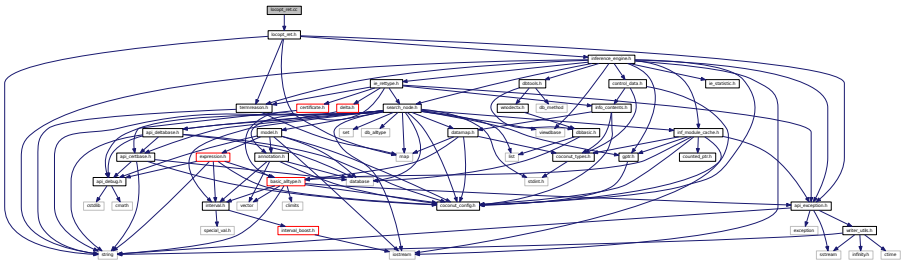
Definition at line 31 of file ithirdder_evaluator.h.

11.141.2.2 #define DEBUG_ITHIRDDER_EVALUATOR 0

Definition at line 30 of file ithirdder_evaluator.h.

11.142 locopt_ret.cc File Reference

`#include <locopt_ret.h>` Include dependency graph for locopt_ret.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

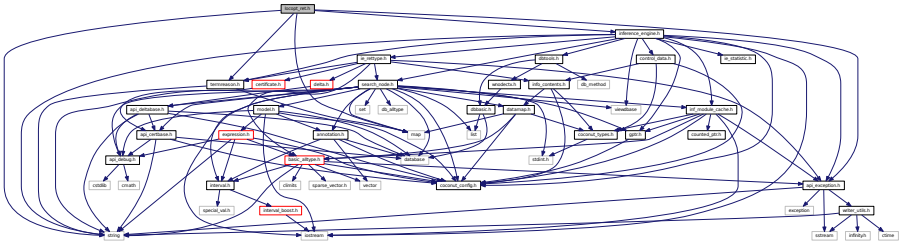
- double coco::get_locopt_weight_from_rval (const locopt_ret &ret_info, const std::string &iename, const int srval)
- termination_reason coco::get_locopt_termr_from_rval (const locopt_ret &ret_info, const std::string &iename, const int srval)

11.142.1 Detailed Description

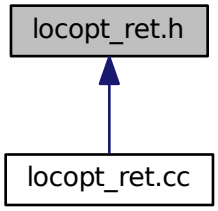
Definition in file locopt_ret.cc.

11.143 locopt_ret.h File Reference

`#include <string>` `#include <map>` `#include <termreason.h>` `#include <inference_engine.h>` `#include <api_exception.h>` Include dependency graph for locopt_ret.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::locopt_ret_record

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define LOPT_SUCCESS 0
- #define LOPT_PARTSUCC 1
- #define LOPT_INFEAS 2
- #define LOPT_UNBND 3

- #define LOPT_LIMIT 4
- #define LOPT_ERROR 5
- #define LOPT_WGHT_SUCCESS 10.0
- #define LOPT_WGHT_PARTSUCC 5.0
- #define LOPT_WGHT_INFEAS 3.0
- #define LOPT_WGHT_UNBND 3.0
- #define LOPT_WGHT_LIMIT 1.0
- #define LOPT_WGHT_ERROR 0.0

Typedefs

- typedef std::map< int, locopt_ret_record > coco::locopt_ret

Functions

- double coco::get_locopt_weight_from_rval (const locopt_ret &ret_info, const std::string &iename, const int srval)
- termination_reason coco::get_locopt_termr_from_rval (const locopt_ret &ret_info, const std::string &iename, const int srval)

11.143.1 Detailed Description

Definition in file locopt_ret.h.

11.143.2 Define Documentation

11.143.2.1 #define LOPT_ERROR 5

Definition at line 53 of file locopt_ret.h.

11.143.2.2 #define LOPT_INFEAS 2

Definition at line 50 of file locopt_ret.h.

11.143.2.3 #define LOPT_LIMIT 4

Definition at line 52 of file locopt_ret.h.

11.143.2.4 #define LOPT_PARTSUCC 1

Definition at line 49 of file locopt_ret.h.

11.143.2.5 #define LOPT_SUCCESS 0

Definition at line 48 of file locopt_ret.h.

11.143.2.6 #define LOPT_UNBND 3

Definition at line 51 of file locopt_ret.h.

11.143.2.7 #define LOPT_WGHT_ERROR 0.0

Definition at line 61 of file locopt_ret.h.

11.143.2.8 #define LOPT_WGHT_INFEAS 3.0

Definition at line 58 of file locopt_ret.h.

11.143.2.9 #define LOPT_WGHT_LIMIT 1.0

Definition at line 60 of file locopt_ret.h.

11.143.2.10 #define LOPT_WGHT_PARTSUCC 5.0

Definition at line 57 of file locopt_ret.h.

11.143.2.11 #define LOPT_WGHT_SUCCESS 10.0

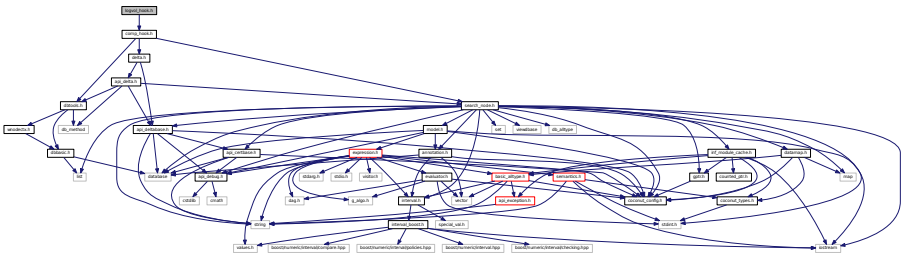
Definition at line 56 of file locopt_ret.h.

11.143.2.12 #define LOPT_WGHT_UNBND 3.0

Definition at line 59 of file locopt_ret.h.

11.144 logvol_hook.h File Reference

`#include <comp_hook.h>` Include dependency graph for logvol_hook.h:



Classes

- class coco::logvol_comp_hook

  The log-volume computation hook (work node computation hook)

Namespaces

- namespace coco

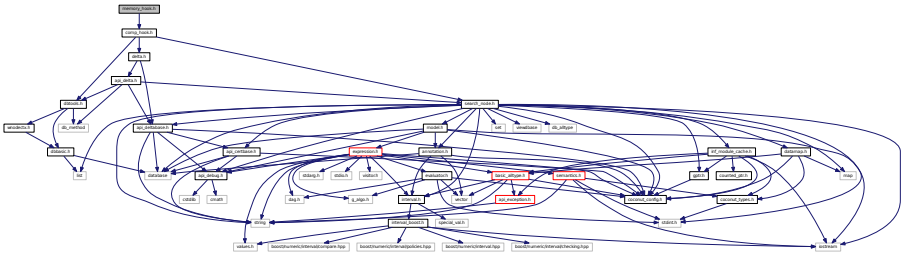
  the main namespace of the COCONUT API

11.144.1 Detailed Description

Definition in file logvol_hook.h.

11.145 memory_hook.h File Reference

`#include <comp_hook.h>` Include dependency graph for memory_hook.h:



Classes

- class coco::memory_comp_hook

  The memory computation hook (work node computation hook)

Namespaces

- namespace coco

  the main namespace of the COCONUT API

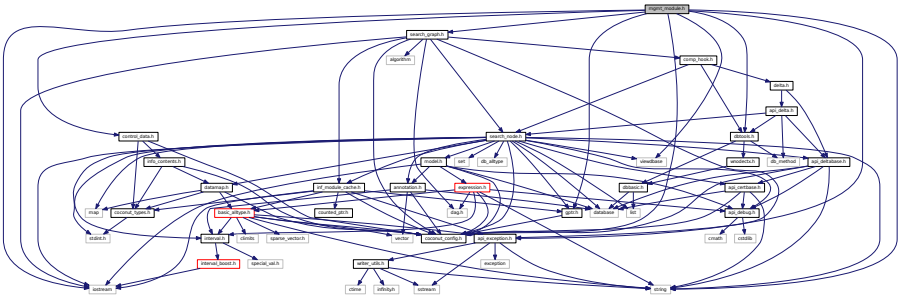
11.145.1 Detailed Description

Definition in file memory_hook.h.

11.146 mgmt_module.h File Reference

`#include <iostream>` `#include <coconut_config.h>` `#include <search_graph.-h>` `#include <gptr.h>` `#include <string>` `#include <dbtools.h>` `#include <viewdbase>` × `#include <control_data.h>` `#include <api_exception.h>` Include dependency graph

for mgmt_module.h:



Classes

- class coco::management_module_exception

  Management module exception class.
- class coco::management_module

  Management module base class.

Namespaces

- namespace coco

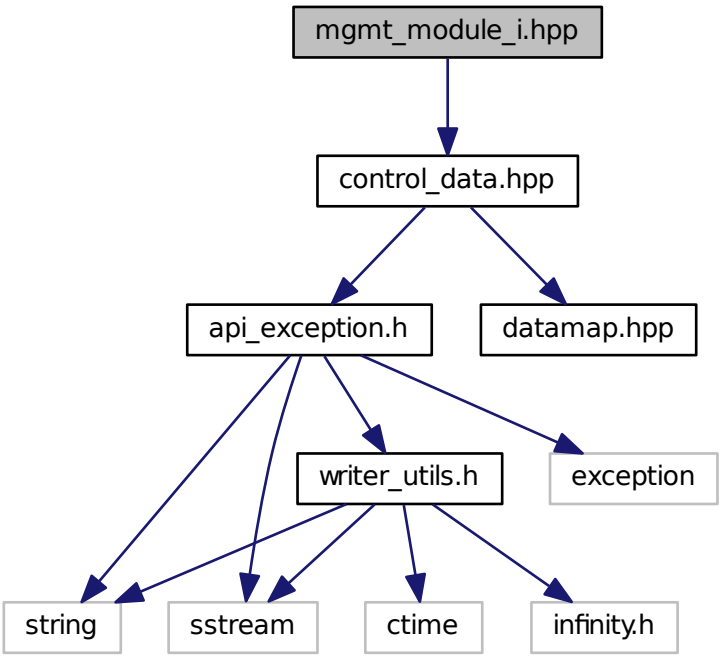
  the main namespace of the COCONUT API

11.146.1 Detailed Description

Definition in file mgmt_module.h.

11.147 mgmt_module_i.hpp File Reference

`#include <control_data.hpp>` Include dependency graph for mgmt_module_i.hpp:



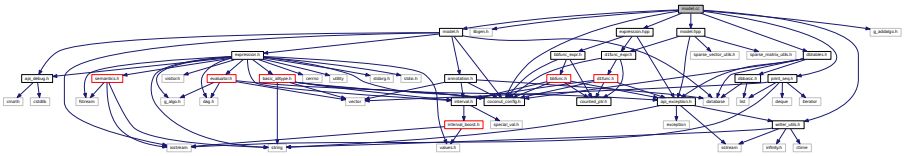
11.147.1 Detailed Description

Definition in file mgmt_module_i.hpp.

11.148 model.cc File Reference

`#include <coconut_config.h>` `#include <libgen.h>` `#include <model.h>` `#include <expression.hpp>` `#include <model.hpp>` `#include <dbtables.h>` `#include <print_seq.h>` `#include <g_addalgo.h>` `#include <writer_utils.h>` Include dependency

graph for model.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define throw_read(MSG)
- #define MT_DBL 1
- #define MT_INTV 2
- #define MT_INT 3
- #define MT_STORED 8
- #define MODEL_INITIAL_BUF_SIZE 1024

Variables

- const char * coco::correxpr_names [ ] = {"log10", "asin", "acos", "atan", "tan", "<=", "<", ">=", ">", "==", "!="}

11.148.1 Detailed Description

Definition in file model.cc.

11.148.2 Define Documentation

11.148.2.1 #define MODEL_INITIAL_BUF_SIZE 1024

Definition at line 660 of file model.cc.

11.148.2.2 #define MT_DBL 1

Definition at line 603 of file model.cc.

11.148.2.3 #define MT_INT 3

Definition at line 605 of file model.cc.

11.148.2.4 #define MT_INTV 2

Definition at line 604 of file model.cc.

11.148.2.5 #define MT_STORED 8

Definition at line 606 of file model.cc.

11.148.2.6 #define throw_read(*MSG*)

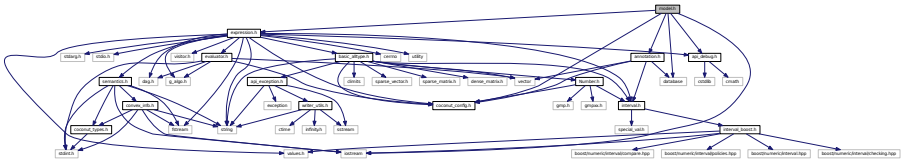
Value:

```
do { throw api_exception(apiee_io, \
                                  std::string("Malformed Input: Line ")+\
                                  convert_to_str(ln)+": "\
                                  +MSG); } while(0)
```

Definition at line 599 of file model.cc.

11.149 model.h File Reference

`#include <iostream>` `#include <coconut_config.h>` `#include <expression.-h>` `#include <database>` `#include <annotation.h>` `#include <api_debug.h>` ×
Include dependency graph for model.h:



This graph shows which files directly or indirectly include this file:

Classes

- class coco::model

  The model class (an attributed DAG of expression nodes, lowest class in the model hierarchy)
- class coco::model_iddata

  The model id-data class (the topmost in the model class hierarchy)
- class coco::model_gid

  Model Group Data Class (middle class in the model hierarchy)

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef model::walker coco::expression_walker

  Walker to the expression DAG.
- typedef model::const_walker coco::expression_const_walker

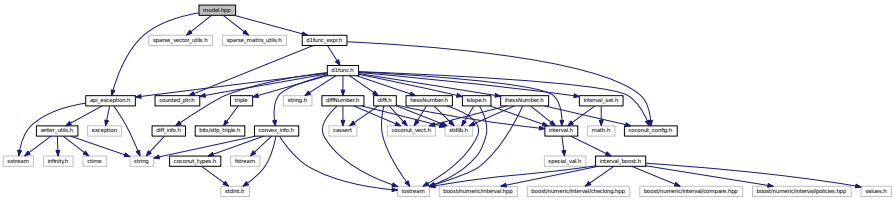
  Const walker to the expression DAG.

11.149.1 Detailed Description

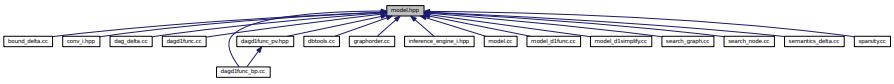
Definition in file model.h.

11.150 model.hpp File Reference

`#include <api_exception.h>` `#include <sparse_vector_utils.h>` `#include <sparse-_matrix_utils.h>` `#include <d1func_expr.h>` Include dependency graph for model.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::model::sort_constraints
- class coco::model::simplify_visitor_0
- class coco::model::simplify_visitor_m
- struct coco::model::__docompare_nodes
- struct coco::model::__docompare_variables

- struct coco::model::detect_0chain_visitor_st
- class coco::model::detect_0chain_visitor
- struct coco::model::lincoeff_visitor_ret
- struct coco::model::lincoeff_visitor_st
- class coco::model::lincoeff_visitor

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define delnodes_insert(N)
- #define delghosts_insert(N)
- #define SIMPLIFY_0_IS_CONST 1
- #define SIMPLIFY_0_CONST_IS_INTEGER (1<<24)
- #define SIMPLIFY_0_CONST_IS_EQUATION (1<<25)
- #define SIMPLIFY_0_IS_VAR (1<<1)
- #define SIMPLIFY_0_IS_SUM (1<<2)
- #define SIMPLIFY_0_SUM_IS_SIMPLE (1<<24)
- #define SIMPLIFY_0_IS_PROD (1<<3)
- #define SIMPLIFY_0_PROD_IS_SIMPLE (1<<24)
- #define SIMPLIFY_0_IS_MULTIPLICATIVE (1<<4)
- #define SIMPLIFY_0_IS_CORRECTEDMULT (1<<5)
- #define SIMPLIFY_0_IS_GHOST (1<<31)
- #define SIMPLIFY_0_SKIP_THIS_NODE (1<<30)

11.150.1 Detailed Description

Definition in file model.hpp.

11.150.2 Define Documentation

11.150.2.1 #define delghosts_insert(*N*)

Value:

```
do {\
  std::vector<unsigned int>::iterator __x = \
    lower_bound((*delghosts).begin(), (*delghosts).end(), (unsigned int)(N)); \
    if(__x == (*delghosts).end() || *__x != (unsigned int)(N)) \
      (*delghosts).insert(__x, (unsigned int)(N)); \
  } while(0)
```

Definition at line 45 of file model.hpp.

11.150.2.2 #define delnodes_insert(*N*)

Value:

```
do {\
  std::vector<unsigned int>::iterator __x = \
    lower_bound((*delnodes).begin(), (*delnodes).end(), (unsigned int)(N)); \
    if(__x == (*delnodes).end() || *__x != (unsigned int)(N)) \
      (*delnodes).insert(__x, (unsigned int)(N)); \
  } while(0)
```

Definition at line 38 of file model.hpp.

11.150.2.3 #define SIMPLIFY_0_CONST_IS_EQUATION (1<<25)

Definition at line 911 of file model.hpp.

11.150.2.4 #define SIMPLIFY_0_CONST_IS_INTEGER (1<<24)

Definition at line 910 of file model.hpp.

11.150.2.5 #define SIMPLIFY_0_IS_CONST 1

Definition at line 909 of file model.hpp.

11.150.2.6 #define SIMPLIFY_0_IS_CORRECTEDMULT (1<<5)

Definition at line 922 of file model.hpp.

11.150.2.7 #define SIMPLIFY_0_IS_GHOST (1<<31)

Definition at line 924 of file model.hpp.

11.150.2.8 #define SIMPLIFY_0_IS_MULTIPLICATIVE (1<<4)

Definition at line 921 of file model.hpp.

11.150.2.9 #define SIMPLIFY_0_IS_PROD (1<<3)

Definition at line 918 of file model.hpp.

11.150.2.10 #define SIMPLIFY_0_IS_SUM (1<<2)

Definition at line 915 of file model.hpp.

11.150.2.11 #define SIMPLIFY_0_IS_VAR (1<<1)

Definition at line 913 of file model.hpp.

11.150.2.12 #define SIMPLIFY_0_PROD_IS_SIMPLE (1<<24)

Definition at line 919 of file model.hpp.

11.150.2.13 #define SIMPLIFY_0_SKIP_THIS_NODE (1<<30)

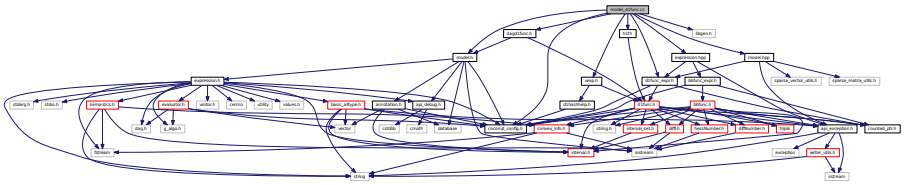
Definition at line 926 of file model.hpp.

11.150.2.14 #define SIMPLIFY_0_SUM_IS_SIMPLE (1<<24)

Definition at line 916 of file model.hpp.

11.151 model_d1func.cc File Reference

`#include <coconut_config.h>` `#include <libgen.h>` `#include <model.h>` `#include <expression.hpp>` `#include <model.hpp>` `#include <d1func_expr.h>` `#include <xexp.h>` `#include <hsf.h>` `#include <dagd1func.h>` Include dependency graph for model_d1func.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define D1FUNC_READ_DEBUG 0
- #define MODEL_SKIP_INITIAL_BUF_SIZE 1024

11.151.1 Define Documentation

11.151.1.1 #define D1FUNC_READ_DEBUG 0

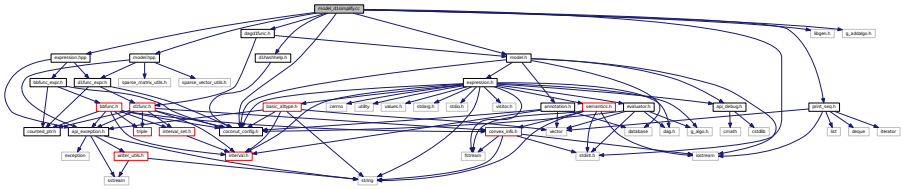
Definition at line 44 of file model_d1func.cc.

11.151.1.2 #define MODEL_SKIP_INITIAL_BUF_SIZE 1024

Definition at line 45 of file model_d1func.cc.

11.152 model_d1simplify.cc File Reference

`#include <coconut_config.h>` `#include <libgen.h>` `#include <model.h>` `#include <expression.hpp>` `#include <model.hpp>` `#include <print_seq.h>` `#include <g-_addalgo.h>` `#include <stdint.h>` `#include <d1hashhelp.h>` `#include <dagd1func.-h>` Include dependency graph for model_d1simplify.cc:



Classes

- class coco::d1func_visitor_0
- class coco::d1func_visitor_1

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define D1SIMPLIFY_DEBUG 0
- #define STORE_FUNCS

11.152.1 Define Documentation

11.152.1.1 #define D1SIMPLIFY_DEBUG 0

Definition at line 43 of file model_d1simplify.cc.

11.152.1.2 #define STORE_FUNCS

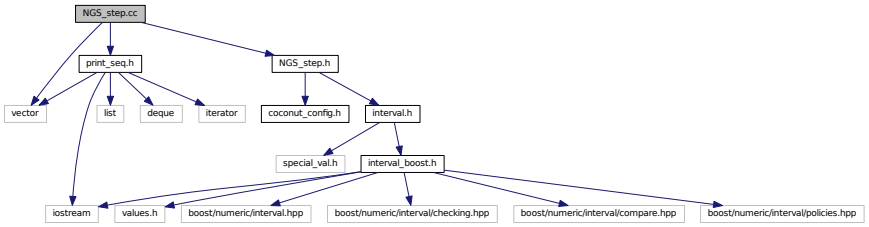
Value:

```
do                                                                      \
      if(_nm)                                                           \
      {                                                                 \
        d1func = new dag_d1func(_nm, init_range, preporder);            \
        d1fe = new d1func_expr(d1func);                                 \
        list_of_models.insert(std::make_pair(d1fe, todo));              \
        todo.clear();                                                   \
      }                                                                 \
    while(0)
```

Definition at line 525 of file model_d1simplify.cc.

11.153 NGS_step.cc File Reference

`#include <vector>` `#include <print_seq.h>` `#include <NGS_step.h>` Include dependency graph for NGS_step.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define NGS_STEP_DEBUG 0
- #define NGS_STEP_1D_DEBUG 0
- #define TINY 1e-50

Functions

- bool coco::mat_inv (const std::vector< std::vector< double > > &A0, std::vector< std::vector< double > > &R)
- bool coco::Newton_Step (const std::vector< std::vector< interval > > &J, const int N, std::vector< interval > x, const std::vector< interval > &c, const std::vector< interval > &fc, std::vector< interval > &xout, bool precond)
- bool coco::Newton_Step (const std::vector< std::vector< interval > > &J, const int N, std::vector< interval > x, const std::vector< interval > &c, const std::vector< interval > &fc, std::vector< std::vector< interval > > &splits, bool precond)
- bool coco::Newton_Step_1D (const interval &J, interval x, const interval &c, const interval &fc, interval &xout, bool precond)
- bool coco::Newton_Step_1D (const interval &J, interval x, const interval &c, const interval &fc, std::vector< interval > &splits, bool precond)

11.153.1 Detailed Description

Definition in file NGS_step.cc.

11.153.2 Define Documentation

11.153.2.1 #define NGS_STEP_1D_DEBUG 0

Definition at line 32 of file NGS_step.cc.

11.153.2.2 #define NGS_STEP_DEBUG 0

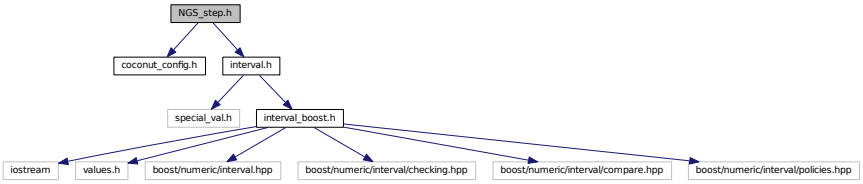
Definition at line 31 of file NGS_step.cc.

11.153.2.3 #define TINY 1e-50

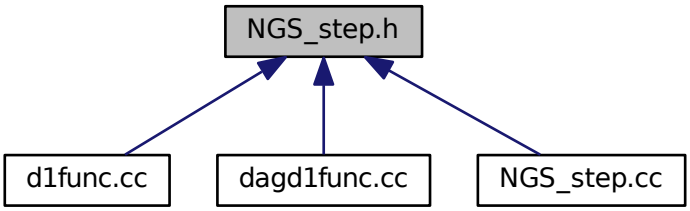
Definition at line 37 of file NGS_step.cc.

11.154 NGS_step.h File Reference

`#include <coconut_config.h>` `#include <interval.h>` Include dependency graph for NGS_step.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

the main namespace of the COCONUT API

Functions

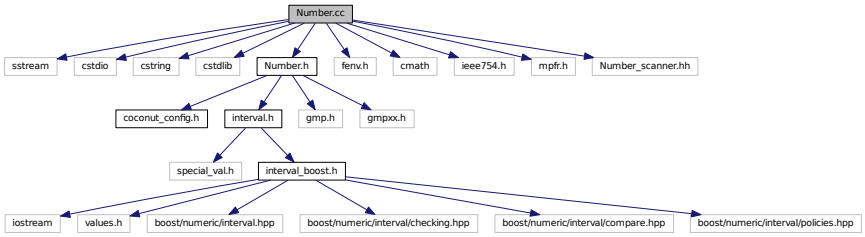
- bool coco::mat_inv (const std::vector< std::vector< double > > &A0, std::vector< std::vector< double > > &R)
- bool coco::Newton_Step (const std::vector< std::vector< interval > > &J, const int N, std::vector< interval > x, const std::vector< interval > &c, const std::vector< interval > &fc, std::vector< interval > &xout, bool precond)
- bool coco::Newton_Step (const std::vector< std::vector< interval > > &J, const int N, std::vector< interval > x, const std::vector< interval > &c, const std::vector< interval > &fc, std::vector< std::vector< interval > > &splits, bool precond)
- bool coco::Newton_Step_1D (const interval &J, interval x, const interval &c, const interval &fc, interval &xout, bool precond)
- bool coco::Newton_Step_1D (const interval &J, interval x, const interval &c, const interval &fc, std::vector< interval > &splits, bool precond)

11.154.1 Detailed Description

Definition in file NGS_step.h.

11.155 Number.cc File Reference

`#include <sstream>` `#include <cstdio>` `#include <cstring>` `#include <cstdlib>` `#include "Number.h"` `#include <fenv.h>` `#include <cmath>` `#include <ieee754.h>` `#include <mpfr.h>` `#include "Number_scanner.hh"` Include dependency graph for Number.cc:



Namespaces

- namespace num

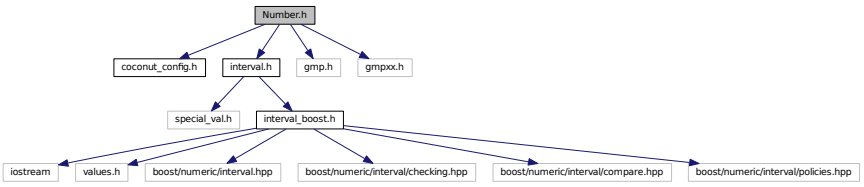
Functions

- std::ostream & num::operator<< (std::ostream &os, const Number &n)
- std::istream & num::operator>> (std::istream &is, Number &n)

- Number num::acos (const Number &x)
- Number num::abs (const Number &x)
- Number num::acosh (const Number &x)
- Number num::acot (const Number &x)
- Number num::acoth (const Number &x)
- Number num::asin (const Number &x)
- Number num::asinh (const Number &x)
- Number num::atan (const Number &x)
- Number num::atanh (const Number &x)
- Number num::cos (const Number &x)
- Number num::cosh (const Number &x)
- Number num::cot (const Number &x)
- Number num::coth (const Number &x)
- Number num::exp (const Number &x)
- Number num::exp10 (const Number &x)
- Number num::exp2 (const Number &x)
- Number num::expm1 (const Number &x)
- Number num::log (const Number &x)
- Number num::log10 (const Number &x)
- Number num::log1p (const Number &x)
- Number num::log2 (const Number &x)
- Number num::power (const Number &x, int n)
- Number num::pow (const Number &x, const Number &y)
- Number num::sin (const Number &x)
- Number num::sinh (const Number &x)
- Number num::sqr (const Number &x)
- Number num::sqrt (const Number &x)
- Number num::tan (const Number &x)
- Number num::tanh (const Number &x)

11.156 Number.h File Reference

`#include <coconut_config.h>` `#include "interval.h"` `#include <gmp.h>` `#include <gmpxx.h>` Include dependency graph for Number.h:



This graph shows which files directly or indirectly include this file:

Classes

- class num::Number
- class num::number_exception

Namespaces

- namespace num

Typedefs

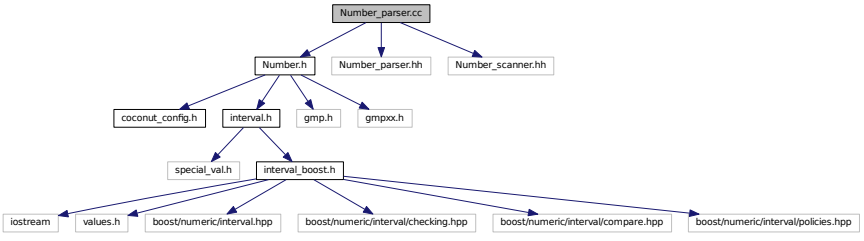
- typedef mpq_t num::rat
- typedef MP_RAT * num::prat
- typedef const MP_RAT * num::cprat
- typedef coco::interval num::intv
- typedef std::string num::str

Enumerations

- enum num::numtypes { num::NT_RAT, num::NT_STR, num::NT_INTV, num::NT_DBL }
- enum num::number_exception_type { num::NE_IRRATIONAL, num::NE_NOSTRING, num::NE_INVALIDINPUT, num::NE_SIMPLIFY, num::NE_INTERNAL_ERROR }

11.157 Number_parser.cc File Reference

`#include "Number.h"` `#include "Number_parser.hh"` `#include "Number_scanner.hh"` Include dependency graph for Number_parser.cc:



Namespaces

- namespace coco_api_internal

Defines

- #define yylex coco_api_internallex
- #define yylex scanner.yylex
- #define YY_(msgid) msgid
- #define YYUSE(e) ((void) (e))
- #define YYCDEBUG if (false) std::cerr
- #define YY_SYMBOL_PRINT(Title, Type, Value, Location)
- #define YY_REDUCE_PRINT(Rule)
- #define YY_STACK_PRINT()
- #define yyerrok (yyerrstatus_ = 0)
- #define yyclearin (yychar = yyempty_)
- #define YYACCEPT goto yyacceptlab
- #define YYABORT goto yyabortlab
- #define YYERROR goto yyerrorlab
- #define YYRECOVERING() (!!yyerrstatus_)

11.157.1 Define Documentation

11.157.1.1 #define YY_(*msgid*) msgid

Definition at line 85 of file Number_parser.cc.

11.157.1.2 #define YY_REDUCE_PRINT(*Rule*)

Definition at line 124 of file Number_parser.cc.

11.157.1.3 #define YY_STACK_PRINT( )

Definition at line 125 of file Number_parser.cc.

11.157.1.4 #define YY_SYMBOL_PRINT(*Title, Type, Value, Location*)

Definition at line 123 of file Number_parser.cc.

11.157.1.5 #define YYABORT goto yyabortlab

Definition at line 133 of file Number_parser.cc.

11.157.1.6 #define YYACCEPT goto yyacceptlab

Definition at line 132 of file Number_parser.cc.

11.157.1.7 #define YYCDEBUG if (false) std::cerr

Definition at line 122 of file Number_parser.cc.

11.157.1.8 #define yyclearin (yychar = yyempty_)

Definition at line 130 of file Number_parser.cc.

11.157.1.9 #define yyerrok (yyerrstatus_ = 0)

Definition at line 129 of file Number_parser.cc.

11.157.1.10 #define YYERROR goto yyerrorlab

Definition at line 134 of file Number_parser.cc.

11.157.1.11 #define yylex coco_api_internallex

Definition at line 65 of file Number_parser.cc.

11.157.1.12 #define yylex scanner.yylex

Definition at line 65 of file Number_parser.cc.

11.157.1.13 #define YYRECOVERING( ) (!!yyerrstatus_)

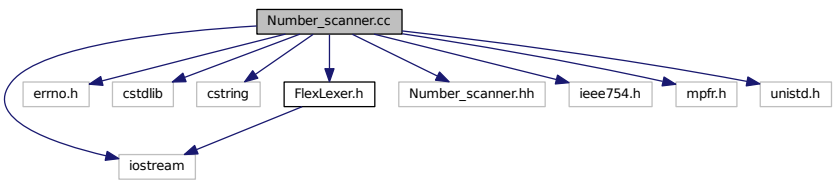
Definition at line 135 of file Number_parser.cc.

11.157.1.14 #define YYUSE(*e*) ((void) (e))

Definition at line 90 of file Number_parser.cc.

11.158 Number_scanner.cc File Reference

`#include <iostream>` `#include <errno.h>` `#include <cstdlib>` `#include <cstring>` × `#include <FlexLexer.h>` `#include "Number_scanner.hh"` `#include <ieee754.-h>` `#include <mpfr.h>` `#include <unistd.h>` Include dependency graph for Number_scanner.cc-:



Classes

- struct yy_buffer_state
- struct yy_trans_info

Defines

- #define YY_INT_ALIGNED short int
- #define FLEX_SCANNER
- #define YY_FLEX_MAJOR_VERSION 2
- #define YY_FLEX_MINOR_VERSION 5
- #define YY_FLEX_SUBMINOR_VERSION 33
- #define FLEX_BETA
- #define yyFlexLexer CoconutNumberFlexLexer
- #define FLEXINT_H
- #define INT8_MIN (-128)
- #define INT16_MIN (-32767-1)
- #define INT32_MIN (-2147483647-1)
- #define INT8_MAX (127)
- #define INT16_MAX (32767)
- #define INT32_MAX (2147483647)
- #define UINT8_MAX (255U)
- #define UINT16_MAX (65535U)
- #define UINT32_MAX (4294967295U)
- #define yyconst
- #define YY_NULL 0
- #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)
- #define BEGIN (yy_start) = 1 + 2 *
- #define YY_START (((yy_start) - 1) / 2)
- #define YYSTATE YY_START
- #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
- #define YY_NEW_FILE yyrestart(yyin)
- #define YY_END_OF_BUFFER_CHAR 0
- #define YY_BUF_SIZE 16384
- #define YY_STATE_BUF_SIZE ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))
- #define YY_TYPEDEF_YY_BUFFER_STATE
- #define EOB_ACT_CONTINUE_SCAN 0
- #define EOB_ACT_END_OF_FILE 1
- #define EOB_ACT_LAST_MATCH 2
- #define YY_LESS_LINENO(n)
- #define yyless(n)
- #define unput(c) yyunput(c, (yytext_ptr))
- #define YY_TYPEDEF_YY_SIZE_T
- #define YY_STRUCT_YY_BUFFER_STATE
- #define YY_BUFFER_NEW 0
- #define YY_BUFFER_NORMAL 1
- #define YY_BUFFER_EOF_PENDING 2
- #define YY_CURRENT_BUFFER
- #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]
- #define yy_new_buffer yy_create_buffer
- #define yy_set_interactive(is_interactive)
- #define yy_set_bol(at_bol)
- #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
- #define yywrap(n) 1
- #define YY_SKIP_YYWRAP

- #define yytext_ptr yytext
- #define YY_DECL int Number_scanner::yylex()
- #define YY_DO_BEFORE_ACTION
- #define YY_NUM_RULES 63
- #define YY_END_OF_BUFFER 64
- #define REJECT reject_used_but_not_detected
- #define yymore() yymore_used_but_not_detected
- #define YY_MORE_ADJ 0
- #define YY_RESTORE_YY_MORE_OFFSET
- #define INITIAL 0
- #define PowOrDivLast 1
- #define YY_EXTRA_TYPE void *
- #define YY_READ_BUF_SIZE 8192
- #define ECHO LexerOutput(yytext, yyleng)
- #define YY_INPUT(buf, result, max_size)
- #define yyterminate() return YY_NULL
- #define YY_START_STACK_INCR 25
- #define YY_FATAL_ERROR(msg) LexerError(msg)
- #define YY_USER_ACTION
- #define YY_BREAK break;
- #define YY_RULE_SETUP YY_USER_ACTION
- #define YY_EXIT_FAILURE 2
- #define yyless(n)
- #define YYTABLES_NAME "yytables"

Typedefs

- typedef signed char flex_int8_t
- typedef short int flex_int16_t
- typedef int flex_int32_t
- typedef unsigned char flex_uint8_t
- typedef unsigned short int flex_uint16_t
- typedef unsigned int flex_uint32_t
- typedef struct yy_buffer_state * YY_BUFFER_STATE
- typedef unsigned int yy_size_t
- typedef unsigned char YY_CHAR
- typedef coco_api_internal::Number_parser::token token

Functions

- void * CoconutNumberalloc (yy_size_t)
- void * CoconutNumberrealloc (void *, yy_size_t)
- void CoconutNumberfree (void *)
- if (!(yy_init))
- while (1)

Variables

- int yyleng
- YY_DECL register yy_state_type yy_current_state
- register char * yy_cp
- register char * yy_bp
- register int yy_act

11.158.1 Define Documentation

11.158.1.1 #define BEGIN (yy_start) = 1 + 2 *

Definition at line 135 of file Number_scanner.cc.

11.158.1.2 #define ECHO LexerOutput(yytext, yyleng)

Definition at line 672 of file Number_scanner.cc.

11.158.1.3 #define EOB_ACT_CONTINUE_SCAN 0

Definition at line 168 of file Number_scanner.cc.

11.158.1.4 #define EOB_ACT_END_OF_FILE 1

Definition at line 169 of file Number_scanner.cc.

11.158.1.5 #define EOB_ACT_LAST_MATCH 2

Definition at line 170 of file Number_scanner.cc.

11.158.1.6 #define FLEX_BETA

Definition at line 14 of file Number_scanner.cc.

11.158.1.7 #define FLEX_SCANNER

Definition at line 9 of file Number_scanner.cc.

11.158.1.8 #define FLEXINT_H

Definition at line 34 of file Number_scanner.cc.

11.158.1.9 #define INITIAL 0

Definition at line 638 of file Number_scanner.cc.

11.158.1.10 #define INT16_MAX (32767)

Definition at line 77 of file Number_scanner.cc.

11.158.1.11 #define INT16_MIN (-32767-1)

Definition at line 68 of file Number_scanner.cc.

11.158.1.12 #define INT32_MAX (2147483647)

Definition at line 80 of file Number_scanner.cc.

11.158.1.13 #define INT32_MIN (-2147483647-1)

Definition at line 71 of file Number_scanner.cc.

11.158.1.14 #define INT8_MAX (127)

Definition at line 74 of file Number_scanner.cc.

11.158.1.15 #define INT8_MIN (-128)

Definition at line 65 of file Number_scanner.cc.

11.158.1.16 #define PowOrDivLast 1

Definition at line 639 of file Number_scanner.cc.

11.158.1.17 #define REJECT reject_used_but_not_detected

Definition at line 577 of file Number_scanner.cc.

11.158.1.18 #define UINT16_MAX (65535U)

Definition at line 86 of file Number_scanner.cc.

11.158.1.19 #define UINT32_MAX (4294967295U)

Definition at line 89 of file Number_scanner.cc.

11.158.1.20 #define UINT8_MAX (255U)

Definition at line 83 of file Number_scanner.cc.

11.158.1.21 #define unput(*c*) yyunput(c, (yytext_ptr))

Definition at line 188 of file Number_scanner.cc.

11.158.1.22 #define YY_AT_BOL( ) (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)

Definition at line 307 of file Number_scanner.cc.

11.158.1.23 #define YY_BREAK break;

Definition at line 723 of file Number_scanner.cc.

11.158.1.24 #define YY_BUF_SIZE 16384

Definition at line 154 of file Number_scanner.cc.

11.158.1.25 #define YY_BUFFER_EOF_PENDING 2

Definition at line 261 of file Number_scanner.cc.

11.158.1.26 #define YY_BUFFER_NEW 0

Definition at line 249 of file Number_scanner.cc.

11.158.1.27 #define YY_BUFFER_NORMAL 1

Definition at line 250 of file Number_scanner.cc.

11.158.1.28 #define YY_CURRENT_BUFFER

Value:

```
( (yy_buffer_stack) \
                          ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
                          : NULL)
```

Definition at line 272 of file Number_scanner.cc.

11.158.1.29 #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]

Definition at line 279 of file Number_scanner.cc.

11.158.1.30 #define YY_DECL int Number_scanner::yylex()

Definition at line 323 of file Number_scanner.cc.

11.158.1.31 #define YY_DO_BEFORE_ACTION

Value:

```
(yytext_ptr) = yy_bp; \
        yyleng = (size_t) (yy_cp - yy_bp); \
        (yy_hold_char) = *yy_cp; \
        *yy_cp = '\0'; \
        (yy_c_buf_p) = yy_cp;
```

Definition at line 328 of file Number_scanner.cc.

11.158.1.32 #define YY_END_OF_BUFFER 64

Definition at line 336 of file Number_scanner.cc.

11.158.1.33 #define YY_END_OF_BUFFER_CHAR 0

Definition at line 150 of file Number_scanner.cc.

11.158.1.34 #define YY_EXIT_FAILURE 2

Definition at line 2046 of file Number_scanner.cc.

11.158.1.35 #define YY_EXTRA_TYPE void *

Definition at line 650 of file Number_scanner.cc.

11.158.1.36 #define YY_FATAL_ERROR(*msg*) LexerError(msg)

Definition at line 701 of file Number_scanner.cc.

11.158.1.37 #define YY_FLEX_MAJOR_VERSION 2

Definition at line 10 of file Number_scanner.cc.

11.158.1.38 #define YY_FLEX_MINOR_VERSION 5

Definition at line 11 of file Number_scanner.cc.

11.158.1.39 #define YY_FLEX_SUBMINOR_VERSION 33

Definition at line 12 of file Number_scanner.cc.

11.158.1.40 #define YY_INPUT(*buf, result, max_size*)

Value:

```
\
        if ( (result = LexerInput( (char *) buf, max_size )) < 0 ) \
                YY_FATAL_ERROR( "input in flex scanner failed" );
```

Definition at line 679 of file Number_scanner.cc.

11.158.1.41 #define YY_INT_ALIGNED short int

Definition at line 5 of file Number_scanner.cc.

11.158.1.42 #define YY_LESS_LINENO(*n*)

Definition at line 172 of file Number_scanner.cc.

11.158.1.43 #define YY_MORE_ADJ 0

Definition at line 579 of file Number_scanner.cc.

11.158.1.44 #define yy_new_buffer yy_create_buffer

Definition at line 285 of file Number_scanner.cc.

11.158.1.45 #define YY_NEW_FILE yyrestart(yyin)

Definition at line 148 of file Number_scanner.cc.

11.158.1.46 #define YY_NULL 0

Definition at line 122 of file Number_scanner.cc.

11.158.1.47 #define YY_NUM_RULES 63

Definition at line 335 of file Number_scanner.cc.

11.158.1.48 #define YY_READ_BUF_SIZE 8192

Definition at line 667 of file Number_scanner.cc.

11.158.1.49 #define YY_RESTORE_YY_MORE_OFFSET

Definition at line 580 of file Number_scanner.cc.

11.158.1.50 #define YY_RULE_SETUP YY_USER_ACTION

Definition at line 726 of file Number_scanner.cc.

11.158.1.51 #define YY_SC_TO_UI(*c*) ((unsigned int) (unsigned char) c)

Definition at line 129 of file Number_scanner.cc.

11.158.1.52 #define yy_set_bol(*at_bol*)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){\
        yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
            yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
        }
```

Definition at line 297 of file Number_scanner.cc.

11.158.1.53 #define yy_set_interactive(*is_interactive*)

Value:

```
{ \
        if ( ! YY_CURRENT_BUFFER ){ \
        yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
            yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
        }
```

Definition at line 287 of file Number_scanner.cc.

11.158.1.54 #define YY_SKIP_YYWRAP

Definition at line 310 of file Number_scanner.cc.

11.158.1.55 #define YY_START (((yy_start) - 1) / 2)

Definition at line 141 of file Number_scanner.cc.

11.158.1.56 #define YY_START_STACK_INCR 25

Definition at line 696 of file Number_scanner.cc.

11.158.1.57 #define YY_STATE_BUF_SIZE ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))

Definition at line 159 of file Number_scanner.cc.

11.158.1.58 #define YY_STATE_EOF(*state*) (YY_END_OF_BUFFER + state + 1)

Definition at line 145 of file Number_scanner.cc.

11.158.1.59 #define YY_STRUCT_YY_BUFFER_STATE

Definition at line 201 of file Number_scanner.cc.

11.158.1.60 #define YY_TYPEDEF_YY_BUFFER_STATE

Definition at line 162 of file Number_scanner.cc.

11.158.1.61 #define YY_TYPEDEF_YY_SIZE_T

Definition at line 196 of file Number_scanner.cc.

11.158.1.62 #define YY_USER_ACTION

Definition at line 718 of file Number_scanner.cc.

11.158.1.63 #define yyconst

Definition at line 118 of file Number_scanner.cc.

11.158.1.64 #define yyFlexLexer CoconutNumberFlexLexer

Definition at line 23 of file Number_scanner.cc.

11.158.1.65 #define yyless(*n*)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                *yy_cp = (yy_hold_char); \
                YY_RESTORE_YY_MORE_OFFSET \
                (yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ;
      \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
```

Definition at line 2058 of file Number_scanner.cc.

11.158.1.66 #define yyless(n)

Value:

```
do \
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                yytext[yyleng] = (yy_hold_char); \
                (yy_c_buf_p) = yytext + yyless_macro_arg; \
                (yy_hold_char) = *(yy_c_buf_p); \
                *(yy_c_buf_p) = '\0'; \
                yyleng = yyless_macro_arg; \
                } \
        while ( 0 )
```

Definition at line 2058 of file Number_scanner.cc.

11.158.1.67 #define yymore( ) yymore_used_but_not_detected

Definition at line 578 of file Number_scanner.cc.

11.158.1.68 #define YYSTATE YY_START

Definition at line 142 of file Number_scanner.cc.

11.158.1.69 #define YYTABLES_NAME "yytables"

Definition at line 2120 of file Number_scanner.cc.

11.158.1.70 #define yyterminate( ) return YY_NULL

Definition at line 691 of file Number_scanner.cc.

11.158.1.71 #define yytext_ptr yytext

Definition at line 314 of file Number_scanner.cc.

11.158.1.72 #define yywrap(n) 1

Definition at line 309 of file Number_scanner.cc.

11.158.2 Typedef Documentation

11.158.2.1 typedef short int flex_int16_t

Definition at line 56 of file Number_scanner.cc.

11.158.2.2 typedef int flex_int32_t

Definition at line 57 of file Number_scanner.cc.

11.158.2.3 typedef signed char flex_int8_t

Definition at line 55 of file Number_scanner.cc.

11.158.2.4 typedef unsigned short int flex_uint16_t

Definition at line 59 of file Number_scanner.cc.

11.158.2.5 typedef unsigned int flex_uint32_t

Definition at line 60 of file Number_scanner.cc.

11.158.2.6 typedef unsigned char flex_uint8_t

Definition at line 58 of file Number_scanner.cc.

11.158.2.7 typedef coco_api_internal::Number_parser::token token

Definition at line 611 of file Number_scanner.cc.

11.158.2.8 typedef struct yy_buffer_state* YY_BUFFER_STATE

Definition at line 163 of file Number_scanner.cc.

11.158.2.9 typedef unsigned char YY_CHAR

Definition at line 312 of file Number_scanner.cc.

11.158.2.10 typedef unsigned int yy_size_t

Definition at line 197 of file Number_scanner.cc.

11.158.3 Function Documentation

11.158.3.1 void * CoconutNumberalloc (yy_size_t *size*)

Definition at line 2098 of file Number_scanner.cc.

11.158.3.2 void CoconutNumberfree (void * *ptr*)

Definition at line 2115 of file Number_scanner.cc.

11.158.3.3 void * CoconutNumberrealloc (void * *ptr,* yy_size_t *size*)

Definition at line 2103 of file Number_scanner.cc.

11.158.3.4 if (! *yy_init*)

Definition at line 741 of file Number_scanner.cc.

11.158.3.5 while (1)

Definition at line 767 of file Number_scanner.cc.

11.158.4 Variable Documentation

11.158.4.1 register int yy_act

Definition at line 735 of file Number_scanner.cc.

11.158.4.2 register char * yy_bp

Definition at line 734 of file Number_scanner.cc.

11.158.4.3 register char* yy_cp

Definition at line 734 of file Number_scanner.cc.

11.158.4.4 YY_DECL register yy_state_type yy_current_state

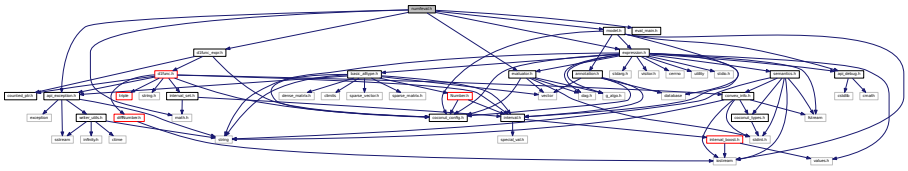
The main scanner function which does all the work.

Definition at line 733 of file Number_scanner.cc.

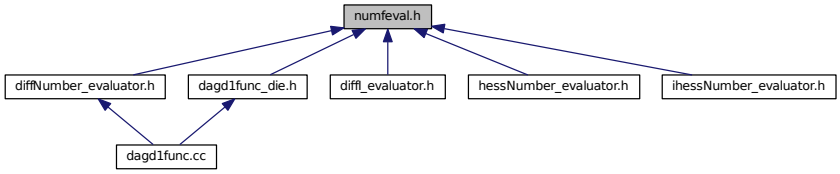
11.158.4.5 int yyleng

11.159 numfeval.h File Reference

`#include <evaluator.h>` `#include <expression.h>` `#include <model.h>` `#include <eval_main.h>` `#include <math.h>` `#include <api_exception.h>` `#include <d1func_expr.h>` Include dependency graph for numfeval.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct coco::xxxNumber_eval_type
  Visitor data for xxxNumber_eval.
- class coco::xxxNumber_eval
  Forward function range evaluation.

Namespaces

- namespace coco
  the main namespace of the COCONUT API

Defines

- #define _xxxNumber_eval_type_(N) N ## _eval_type
- #define xxxNumber_eval_type_(N) _xxxNumber_eval_type_(N)
- #define xxxNumber_eval_type xxxNumber_eval_type_(xxxNumber_name)
- #define _xxxNumber_eval_(N) N ## _eval
- #define xxxNumber_eval_(N) _xxxNumber_eval_(N)
- #define xxxNumber_eval xxxNumber_eval_(xxxNumber_name)
- #define _xxxNumber_evaluator_(N) N ## _evaluator
- #define xxxNumber_evaluator_(N) _xxxNumber_evaluator_(N)
- #define xxxNumber_evaluator xxxNumber_evaluator_(xxxNumber_name)
- #define _xxxNumber_str_(N) #N
- #define xxxNumber_str_(N) _xxxNumber_str_(N)
- #define xxxNumber_eval_str xxxNumber_str_(xxxNumber_eval)

Typedefs

- typedef xxxNumber_t(∗ coco::xxxNumber_evaluator)(const std::vector< xxxNumber_t > ∗__x, const variable_indicator &__v)

11.159.1 Detailed Description

Definition in file numfeval.h.

11.159.2 Define Documentation

11.159.2.1 #define _xxxNumber_eval_(N) N ## _eval

Definition at line 49 of file numfeval.h.

11.159.2.2 #define _xxxNumber_eval_type_(N) N ## _eval_type

Definition at line 45 of file numfeval.h.

11.159.2.3 #define _xxxNumber_evaluator_(N) N ## _evaluator

Definition at line 53 of file numfeval.h.

11.159.2.4 #define _xxxNumber_str_(N) #N

Definition at line 57 of file numfeval.h.

11.159.2.5 #define xxxNumber_eval xxxNumber_eval_(xxxNumber_name)

Definition at line 51 of file numfeval.h.

11.159.2.6 #define xxxNumber_eval_(N) _xxxNumber_eval_(N)

Definition at line 50 of file numfeval.h.

11.159.2.7 #define xxxNumber_eval_str xxxNumber_str_(xxxNumber_eval)

Definition at line 59 of file numfeval.h.

11.159.2.8 #define xxxNumber_eval_type xxxNumber_eval_type_(xxxNumber_name)

Definition at line 47 of file numfeval.h.

11.159.2.9 #define xxxNumber_eval_type_(N) _xxxNumber_eval_type_(N)

Definition at line 46 of file numfeval.h.

11.159.2.10 #define xxxNumber_evaluator xxxNumber_evaluator_(xxxNumber_name)

Definition at line 55 of file numfeval.h.

11.159.2.11 #define xxxNumber_evaluator_(N) _xxxNumber_evaluator_(N)

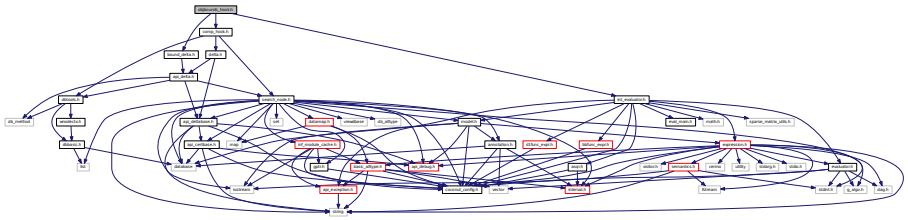
Definition at line 54 of file numfeval.h.

11.159.2.12 #define xxxNumber_str_(N) _xxxNumber_str_(N)

Definition at line 58 of file numfeval.h.

11.160 objbounds_hook.h File Reference

`#include <comp_hook.h>` `#include <int_evaluator.h>` `#include <bound_delta.-h>` Include dependency graph for objbounds_hook.h:



Classes

- class coco::objbounds_comp_hook

  The objective-bounds computation hook (work node computation hook)

Namespaces

- namespace coco

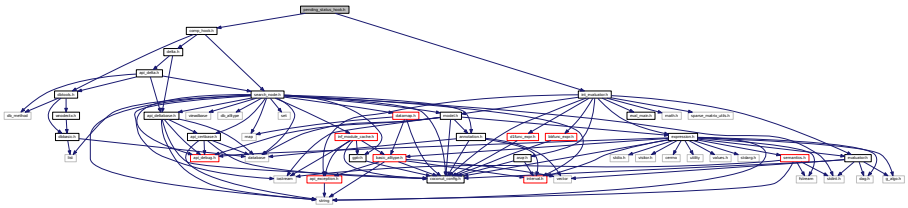
  the main namespace of the COCONUT API

11.160.1 Detailed Description

Definition in file objbounds_hook.h.

11.161 pending_status_hook.h File Reference

`#include <comp_hook.h>` `#include <int_evaluator.h>` Include dependency graph for pending_status_hook.h:



Classes

- class coco::pending_status_comp_hook

  The pending status computation hook (work node computation hook)

Namespaces

- namespace coco

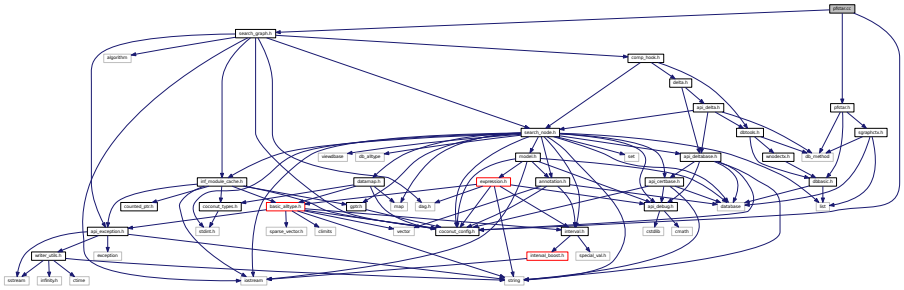
  the main namespace of the COCONUT API

11.161.1 Detailed Description

Definition in file pending_status_hook.h.

11.162 pfstar.cc File Reference

`#include <coconut_config.h>` `#include <search_graph.h>` `#include <pfstar.-h>` Include dependency graph for pfstar.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define PFSTAR_DEBUG 0

11.162.1 Detailed Description

Definition in file pfstar.cc.

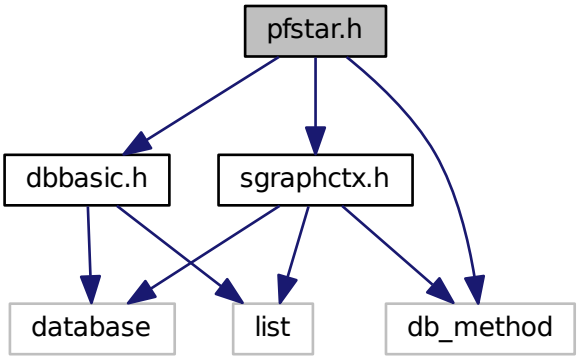
11.162.2 Define Documentation

11.162.2.1 #define PFSTAR_DEBUG 0

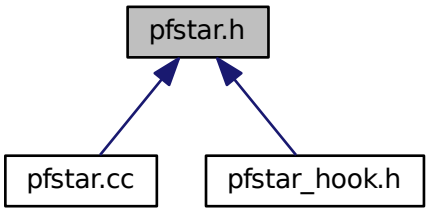
Definition at line 32 of file pfstar.cc.

11.163 pfstar.h File Reference

`#include <dbbasic.h>` `#include <db_method>` `#include <sgraphctx.h>` Include dependency graph for pfstar.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::calc_pf_star

  Stored procedure calculating the pf∗ value of a box.

Namespaces

- namespace coco

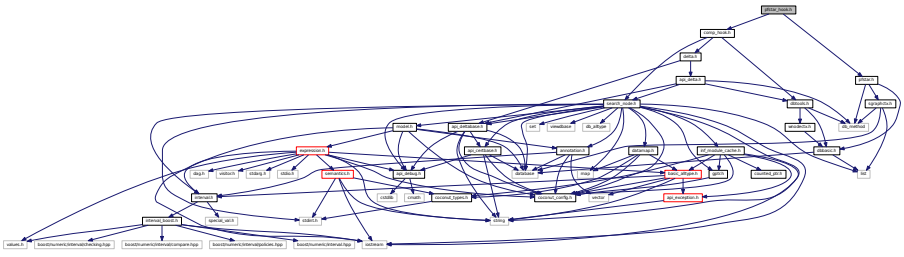
the main namespace of the COCONUT API

11.163.1 Detailed Description

Definition in file pfstar.h.

11.164 pfstar_hook.h File Reference

`#include <comp_hook.h>` `#include <pfstar.h>` Include dependency graph for pfstar_hook.h-
:



Classes

- class coco::pfstar_hook

  The pfstar computation hook (work node computation hook)

Namespaces

- namespace coco

  the main namespace of the COCONUT API

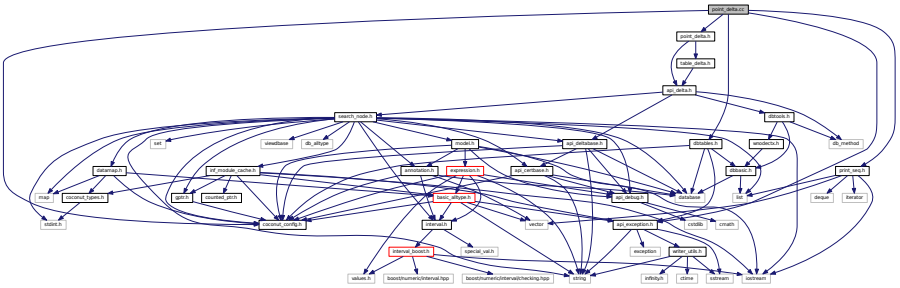
11.164.1 Detailed Description

Definition in file pfstar_hook.h.

11.165 point_delta.cc File Reference

`#include <coconut_config.h>` `#include <point_delta.h>` `#include <print_-
seq.h>` `#include <api_exception.h>` `#include <dbtables.h>` Include dependency graph

for point_delta.cc:



Namespaces

- namespace coco

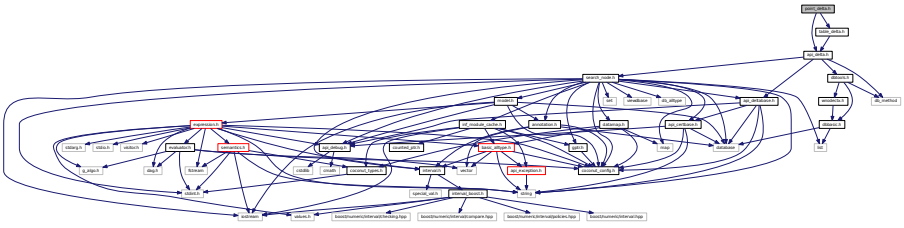
  the main namespace of the COCONUT API

11.165.1 Detailed Description

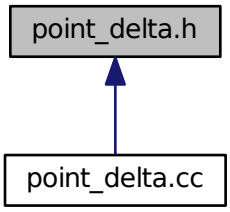
Definition in file point_delta.cc.

11.166 point_delta.h File Reference

`#include <api_delta.h>` `#include <table_delta.h>` Include dependency graph for point-_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::point_delta

  A delta class which adds new points to the search database.

Namespaces

- namespace coco

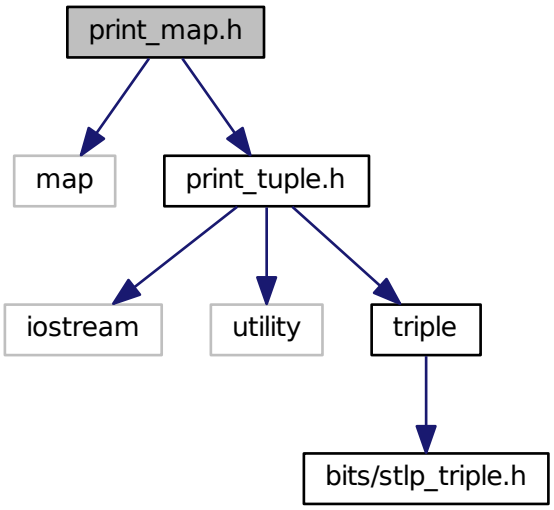
  the main namespace of the COCONUT API

11.166.1 Detailed Description

Definition in file point_delta.h.

11.167 print_map.h File Reference

`#include <map>` `#include <print_tuple.h>` Include dependency graph for print_map.h:



Namespaces

- namespace std

  The standard namespace.

Functions

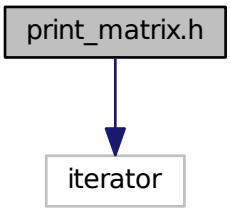
- template<typename _TK , typename _Ve , typename _TC , typename _TA >
  ostream & std::operator<< (ostream &o, const map< _TK, _Ve, _TC, _TA > &a)
- template<typename _TK , typename _Ve , typename _TC , typename _TA >
  ostream & std::operator<< (ostream &o, const multimap< _TK, _Ve, _TC, _TA > &a)

11.167.1 Detailed Description

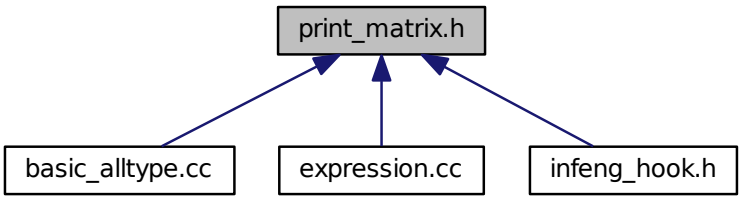
Definition in file print_map.h.

11.168 print_matrix.h File Reference

`#include <iterator>` Include dependency graph for print_matrix.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace std

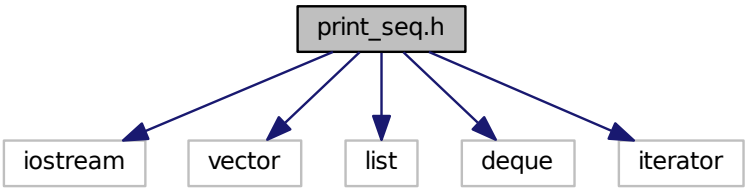
  The standard namespace.

11.168.1 Detailed Description

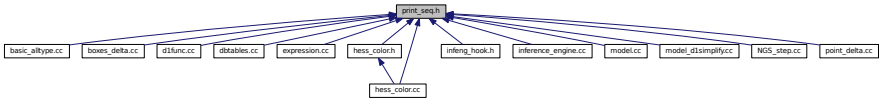
Definition in file print_matrix.h.

11.169 print_seq.h File Reference

`#include <iostream>` `#include <vector>` `#include <list>` `#include <deque>` × `#include <iterator>` Include dependency graph for print_seq.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace std

  The standard namespace.

Functions

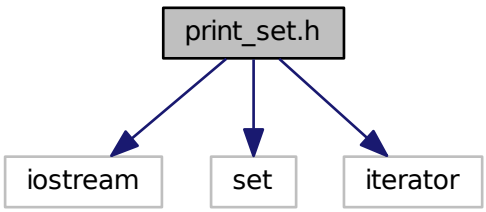
- template<typename _Ve , typename _TA >
  ostream & std::operator<< (ostream &o, const vector< _Ve, _TA > &a)
- template<typename _Ve , typename _TA >
  ostream & std::operator<< (ostream &o, const list< _Ve, _TA > &a)
- template<typename _Ve , typename _TA >
  ostream & std::operator<< (ostream &o, const deque< _Ve, _TA > &a)

11.169.1 Detailed Description

Definition in file print_seq.h.

11.170 print_set.h File Reference

`#include <iostream>` `#include <set>` `#include <iterator>` Include dependency graph for print_set.h:



Namespaces

- namespace std

  The standard namespace.

Functions

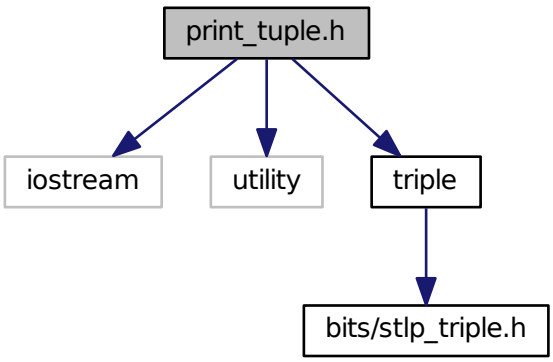
- template<typename _Ve , typename _TC , typename _TA >
  ostream & std::operator<< (ostream &o, const set< _Ve, _TC, _TA > &a)
- template<typename _Ve , typename _TC , typename _TA >
  ostream & std::operator<< (ostream &o, const multiset< _Ve, _TC, _TA > &a)

11.170.1 Detailed Description

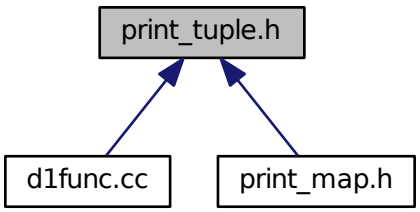
Definition in file print_set.h.

11.171 print_tuple.h File Reference

`#include <iostream>` `#include <utility>` `#include <triple>` Include dependency graph for print_tuple.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace std

  The standard namespace.

Functions

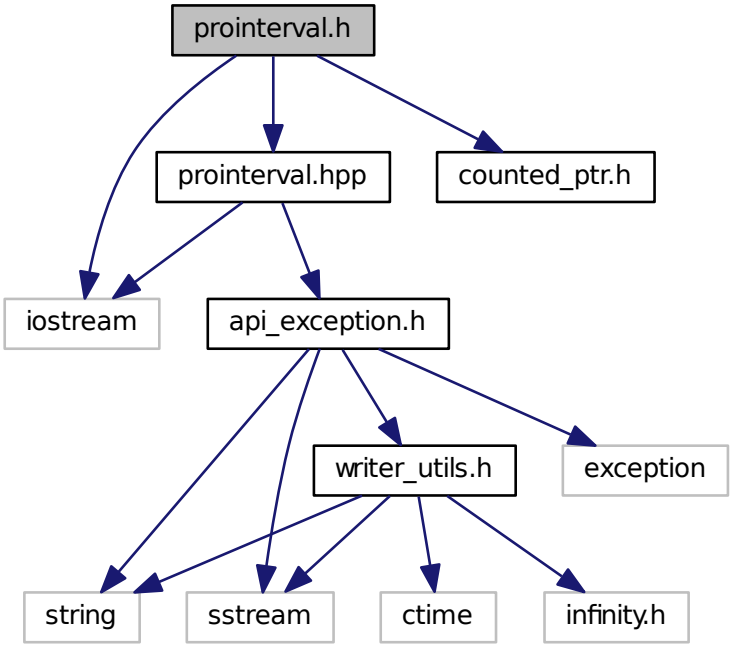
- template<typename _TA , typename _TB >
  ostream & std::operator<< (ostream &o, const pair< _TA, _TB > &a)
- template<typename _TA , typename _TB , typename _TC >
  ostream & std::operator<< (ostream &o, const triple< _TA, _TB, _TC > &a)

11.171.1 Detailed Description

Definition in file print_tuple.h.

11.172 prointerval.h File Reference

`#include <iostream>` `#include <counted_ptr.h>` `#include <prointerval.hpp>` ×
Include dependency graph for prointerval.h:



Classes

- class coco::proj_rational
- class coco::projective_interval

Namespaces

- namespace coco

  the main namespace of the COCONUT API
- namespace std

  The standard namespace.

Functions

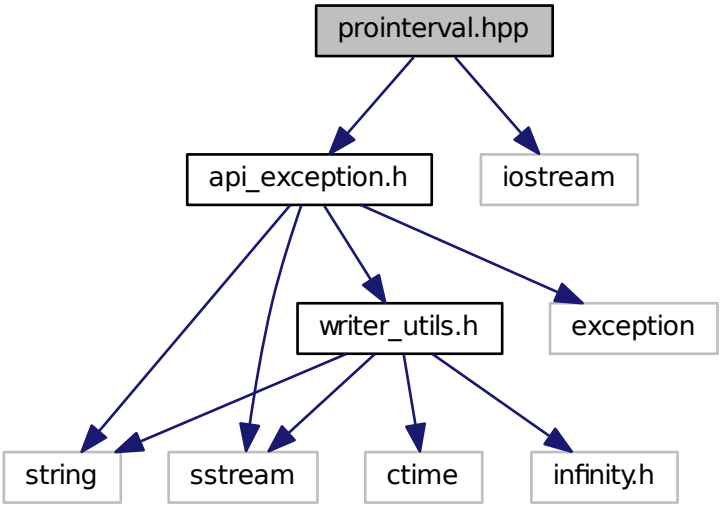
- bool coco::operator< (int n, const proj_rational &R)
- bool coco::operator<= (int n, const proj_rational &R)
- bool coco::operator> (int n, const proj_rational &R)
- bool coco::operator>= (int n, const proj_rational &R)
- bool coco::operator== (int n, const proj_rational &R)
- bool coco::operator!= (int n, const proj_rational &R)
- proj_rational coco::abs (const proj_rational &r)
- template<class I >
  I coco::pow (const I &i, const proj_rational &r)
- ostream & std::operator<< (ostream &o, const coco::proj_rational &x)
- template<class I >
  ostream & std::operator<< (ostream &o, const coco::projective_interval< I > &a)

11.172.1 Detailed Description

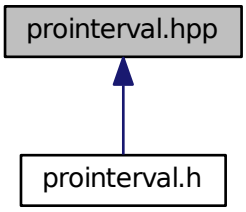
Definition in file prointerval.h.

11.173 prointerval.hpp File Reference

`#include <api_exception.h>` `#include <iostream>` Include dependency graph for prointerval.hpp-
:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco
  the main namespace of the COCONUT API

Functions

- template<class I >
  projective_interval< I > coco::operator+ (const projective_interval< I > &a, const projective_interval< I > &b)
- template<class I >
  projective_interval< I > coco::operator+ (const projective_interval< I > &a, double b)
- template<class I >
  projective_interval< I > coco::operator+ (double b, const projective_interval< I > &a)
- template<class I >
  projective_interval< I > coco::operator- (const projective_interval< I > &a, const projective_interval< I > &b)
- template<class I >
  projective_interval< I > coco::operator- (const projective_interval< I > &a, double b)
- template<class I >
  projective_interval< I > coco::operator- (double b, const projective_interval< I > &a)
- template<class I >
  projective_interval< I > coco::operator∗ (const projective_interval< I > &a, const projective_interval< I > &b)
- template<class I >
  projective_interval< I > coco::operator∗ (const projective_interval< I > &a, double b)
- template<class I >
  projective_interval< I > coco::operator∗ (double b, const projective_interval< I > &a)
- template<class I >
  projective_interval< I > coco::operator/ (const projective_interval< I > &a, const projective_interval< I > &b)
- template<class I >
  projective_interval< I > coco::operator/ (const projective_interval< I > &a, double b)
- template<class I >
  projective_interval< I > coco::operator/ (double b, const projective_interval< I > &a)
- template<class I >
  projective_interval< I > coco::acos (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::abs (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::acosh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::acot (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::acoth (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::asin (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::asinh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::atan (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::atanh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::cos (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::cosh (const projective_interval< I > &x)

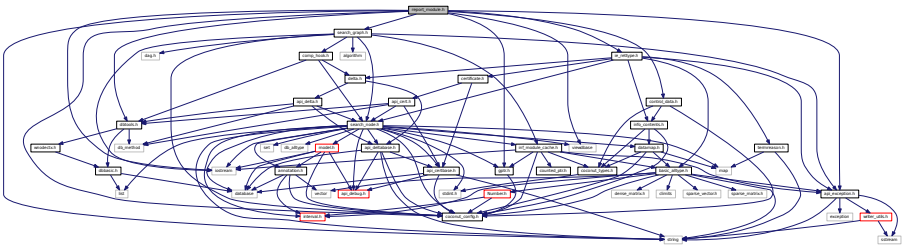
- template<class I >
  projective_interval< I > coco::cot (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::coth (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::exp (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::exp10 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::exp2 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::expm1 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::log (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::log10 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::log1p (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::log2 (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::power (const projective_interval< I > &x, int n)
- template<class I >
  projective_interval< I > coco::pow (const projective_interval< I > &x, const projective_interval< I > &y)
- template<class I >
  projective_interval< I > coco::sin (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::sinh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::sqr (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::sqrt (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::tan (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::tanh (const projective_interval< I > &x)
- template<class I >
  projective_interval< I > coco::imax (const projective_interval< I > &x, const projective_interval< I > &y)
- template<class I >
  projective_interval< I > coco::imin (const projective_interval< I > &x, const projective_interval< I > &y)
- template<class I >
  projective_interval< I > coco::atan2 (const projective_interval< I > &x, const projective_interval< I > &y)
- template<class I >
  projective_interval< I > coco::division_part1 (const projective_interval< I > &x, const projective_interval< I > &y, bool &b)
- template<class I >
  projective_interval< I > coco::division_part2 (const projective_interval< I > &x, const projective_interval< I > &y, bool b)

11.173.1 Detailed Description

Definition in file prointerval.hpp.

11.174 report_module.h File Reference

`#include <iostream> #include <coconut_config.h> #include <search_graph.-h> #include <gptr.h> #include <string> #include <dbtools.h> #include <viewdbase> × #include <control_data.h> #include <ie_rettype.h> #include <api_exception.-h>` Include dependency graph for report_module.h:



Classes

- class coco::report_module_exception

  Report module exception class.
- class coco::report_module

  Report module base class.

Namespaces

- namespace coco

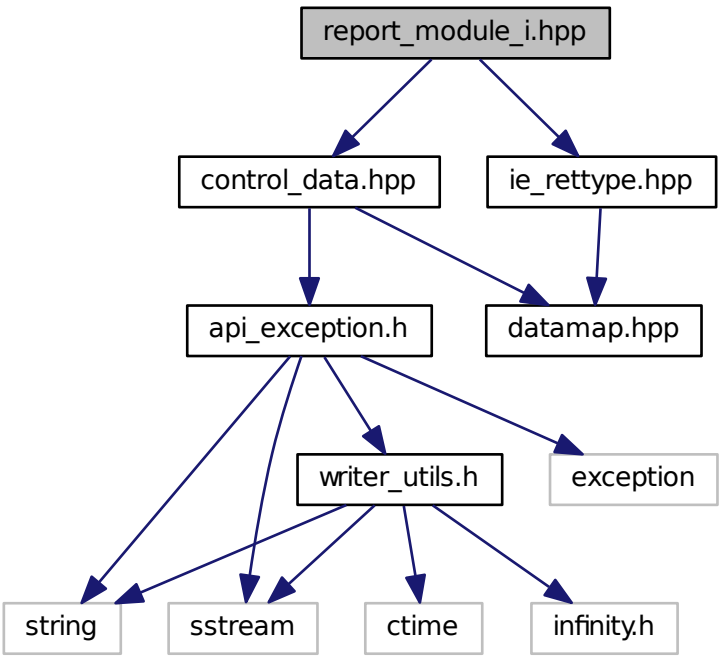
  the main namespace of the COCONUT API

11.174.1 Detailed Description

Definition in file report_module.h.

11.175 report_module_i.hpp File Reference

`#include <control_data.hpp>` `#include <ie_rettype.hpp>` Include dependency graph for report_module_i.hpp:



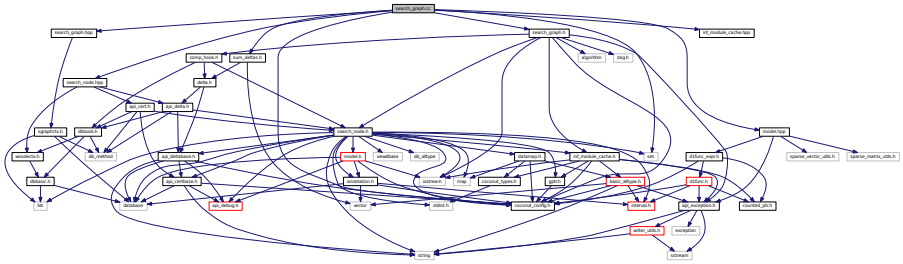
11.175.1 Detailed Description

Definition in file report_module_i.hpp.

11.176 search_graph.cc File Reference

`#include <coconut_config.h>` `#include <search_graph.h>` `#include <model.-hpp>` `#include <search_graph.hpp>` `#include <search_node.hpp>` `#include <inf-_module_cache.hpp>` `#include <set>` `#include "sum_deltas.h"` Include dependency

graph for search_graph.cc:



Classes

- class coco::coco::certificate

  The certificate class (certifies deltas for rigorous mode operation)
- class coco::coco::certificate_base

  Base class for the certificates.
- class coco::coco::delta

  The delta class (updates to work nodes)
- class coco::coco::delta_base

  Base class for the deltas.
- class coco::coco::undelta

  The undelta class (undo of updates to work nodes)
- class coco::coco::undelta_base

  Base class for the undeltas.
- class coco::coco::certificate

  The certificate class (certifies deltas for rigorous mode operation)
- class coco::coco::certificate_base

  Base class for the certificates.
- class coco::coco::delta

  The delta class (updates to work nodes)
- class coco::coco::delta_base

  Base class for the deltas.
- class coco::coco::undelta

  The undelta class (undo of updates to work nodes)
- class coco::coco::undelta_base

  Base class for the undeltas.
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval

- class coco::coco::convex_e

  Convexity information.
- class coco::coco::semantics

  Expression Semantics.
- class coco::coco::variable_indicator

  Bitmap class used to indicate variable occurrence.
- class coco::coco::_evaluator_base

  Base class of all evaluators.
- class coco::coco::evaluator_base

  Base class of all (non-caching) evaluators.
- class coco::coco::cached_evaluator_base

  Base class of all caching evaluators.
- class coco::coco::forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class coco::coco::backward_evaluator_base

  Base class of all (non-caching) backward evaluators.
- class coco::coco::cached_forward_evaluator_base

  Base class of all (non-caching) forward evaluators.
- class coco::coco::cached_backward_evaluator_base

  Base class of all caching backward evaluators.
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- class coco::num::Number
- class coco::num::number_exception
- class coco::coco::api_exception

  API exception class.
- class coco::coco::nyi_exception

  Not Yet Implemented exception class.
- class coco::coco::basic_alltype

  The basic alltype which can hold any of a number of basic types.
- struct coco::coco::coco::checking_my< T >
- struct coco::coco::coco::my_rounded_math< T >
- struct coco::coco::coco::interval_st
- class coco::coco::coco::interval
- struct coco::coco::coco::checking_my< T >
- struct coco::coco::coco::my_rounded_math< T >
- struct coco::coco::coco::interval_st
- class coco::coco::coco::interval
- class coco::coco::num::Number
- class coco::coco::num::number_exception
- class coco::coco::coco::api_exception

  API exception class.

- class coco::coco::coco::nyi_exception

  Not Yet Implemented exception class.
- class coco::coco::coco::basic_alltype

  The basic alltype which can hold any of a number of basic types.
- class coco::coco::expression_node

  The base class for a node in the expression DAGs.
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- class coco::coco::annotation

  Annotations for Models.
- class coco::coco::model

  The model class (an attributed DAG of expression nodes, lowest class in the model hierarchy)
- class coco::coco::model_iddata

  The model id-data class (the topmost in the model class hierarchy)
- class coco::coco::model_gid

  Model Group Data Class (middle class in the model hierarchy)
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- class coco::coco::annotation

  Annotations for Models.
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- class coco::num::Number
- class coco::num::number_exception
- class coco::coco::api_exception

  API exception class.
- class coco::coco::nyi_exception

  Not Yet Implemented exception class.
- class coco::coco::basic_alltype

  The basic alltype which can hold any of a number of basic types.
- class coco::coco::datamap

  The base class for communicating with COCONUT modules.
- class coco::coco::api_exception

  API exception class.
- class coco::coco::nyi_exception

  Not Yet Implemented exception class.
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >

- struct coco::coco::interval_st
- class coco::coco::interval
- struct coco::coco::checking_my< T >
- struct coco::coco::my_rounded_math< T >
- struct coco::coco::interval_st
- class coco::coco::interval
- class coco::num::Number
- class coco::num::number_exception
- class coco::coco::api_exception

  API exception class.
- class coco::coco::nyi_exception

  Not Yet Implemented exception class.
- class coco::coco::basic_alltype

  The basic alltype which can hold any of a number of basic types.
- class coco::coco::inference_module_cache_autogen

  Inference module cache auto-generate method base class.
- class coco::coco::inference_module_cache

  Inference module cache class.
- class coco::coco::certificate

  The certificate class (certifies deltas for rigorous mode operation)
- class coco::coco::certificate_base

  Base class for the certificates.
- class coco::coco::delta

  The delta class (updates to work nodes)
- class coco::coco::delta_base

  Base class for the deltas.
- class coco::coco::undelta

  The undelta class (undo of updates to work nodes)
- class coco::coco::undelta_base

  Base class for the undeltas.
- class coco::coco::certificate

  The certificate class (certifies deltas for rigorous mode operation)
- class coco::coco::certificate_base

  Base class for the certificates.
- class coco::coco::search_node

  Base type of the nodes in the search graph.
- class coco::coco::delta_node

  Class holding the delta nodes in the search graph.
- class coco::coco::full_node

  Class holding the full nodes in the search graph.
- class coco::coco::work_node

  Work node, which is passed to the inference engines.
- class coco::coco::work_node::constraint_iterator_base

  The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.
- class coco::coco::dbt_row

  This type is used to hold one row in some table of the search database.
- class coco::coco::dbt_row

This type is used to hold one row in some table of the search database.

- class coco::coco::work_node_context

  The evaluation context when retrieving from the search database.
- class coco::coco::point_check_feasibility

  Stored procedure checking the feasibility of a point.
- class coco::coco::box_check_intersection

  Stored procedure checking whether a box intersects the work_node's box.
- class coco::coco::delta_get_action

  Stored procedure class for computing the delta action specifier.
- class coco::sum_deltas

  Pre-post visitor for summing up all the deltas during work node extraction.
- class coco::__sg_anc_visitor

Namespaces

- namespace coco

  the main namespace of the COCONUT API
- namespace coco::coco
- namespace coco::num
- namespace coco::coco::coco
- namespace coco::coco::num

Defines

- #define DEBUG_SGRAPH_EXTRACT 0

Typedefs

- typedef vdbl::rowid coco::coco::delta_id

  The class for delta ids.
- typedef enum coco::coco::tristate_e coco::coco::tristate
- typedef uint32_t coco::coco::search_node_id

  Type of the unique search node identifier.
- typedef interval coco::coco::rhs_t
- typedef std::vector< void * > coco::coco::evaluator_v
- typedef boost::numeric::interval_lib::policies < my_rounded_math< double > , checking_my< double >> coco::coco::coco::my_policies
- typedef mpq_t coco::coco::num::rat
- typedef MP_RAT * coco::coco::num::prat
- typedef const MP_RAT * coco::coco::num::cprat
- typedef coco::interval coco::coco::num::intv
- typedef std::string coco::coco::num::str
- typedef model::walker coco::coco::expression_walker

  Walker to the expression DAG.
- typedef model::const_walker coco::coco::expression_const_walker

  Const walker to the expression DAG.

Enumerations

- enum coco::coco::tristate_e { coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0 }
- enum coco::coco::tristate_e { coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0 }
- enum coco::coco::convex_info { coco::coco::c_convex = 1, coco::coco::c_linear = 0, coco::coco::c_concave = -1, coco::coco::c_maybe = 2, coco::coco::c_not = -2, coco::coco::c_constant = 3 }

  Convexity information enum.
- enum coco::coco::type_annotation { coco::coco::v_exists = 0, coco::coco::v_forall = 1, coco::coco::v_free = 2, coco::coco::v_stochastic = 3 }

  Node type information enum.
- enum coco::coco::activity_descr { coco::coco::a_redundant = 1, coco::coco::a_active_lo = 2, coco::coco::a_active_lo_red = a_active_lo|a_redundant, coco::coco::a_active_hi = 4, coco::coco::a_active_hi_red = a_active_hi|a_redundant, coco::coco::a_active = a_active_lo|a_active_hi, coco::coco::a_active_red = a_active|a_redundant }

  Constraint activity information enum.
- enum coco::num::numtypes { coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL }
- enum coco::num::number_exception_type { coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR }
- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer =

10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

Enum for classifying api_exceptions.

- enum coco::coco::e_expression_type { coco::coco::ex_bound = 1, coco::coco::ex_linear = 1<<1, coco::coco::ex_quadratic = 1<<2, coco::coco::ex_polynomial = 1<<3, coco::coco::ex_other = 1<<4, coco::coco::ex_kj = 1<<7, coco::coco::ex_org = 1<<8, coco::coco::ex_redundant = 1<<9, coco::coco::ex_notredundant = 1<<10, coco::coco::ex_active_lo = 1<<11, coco::coco::ex_inactive_lo = 1<<12, coco::coco::ex_active_hi = 1<<13, coco::coco::ex_inactive_hi = 1<<14, coco::coco::ex_active = ex_active_lo|ex_active_hi, coco::coco::ex_inactive = ex_inactive_lo|ex_inactive_hi, coco::coco::ex_integer = 1<<15, coco::coco::ex_exists = 1<<16, coco::coco::ex_forall = 1<<17, coco::coco::ex_free = 1<<18, coco::coco::ex_stochastic = 1<<19, coco::coco::ex_convex = 1<<20, coco::coco::ex_concave = 1<<21, coco::coco::ex_inequality = 1<<28, coco::coco::ex_equality = 1<<29, coco::coco::ex_leftbound = 1<<30, coco::coco::ex_rightbound = 1<<31, coco::coco::ex_atmlin = ex_bound|ex_linear, coco::coco::ex_atmquad = ex_atmlin|ex_quadratic, coco::coco::ex_atmpoly = ex_atmquad|ex_polynomial, coco::coco::ex_nonlin = ex_quadratic|ex_polynomial|ex_other, coco::coco::ex_nonbnd = ex_linear|ex_nonlin, coco::coco::ex_any = ex_atmlin|ex_nonlin, coco::coco::ex_bothbound = ex_leftbound|ex_rightbound }
- enum coco::coco::num::numtypes { coco::coco::num::NT_RAT, coco::coco::num::NT_STR, coco::coco::num::NT_INTV, coco::coco::num::NT_DBL }
- enum coco::coco::num::number_exception_type { coco::coco::num::NE_IRRATIONAL, coco::coco::num::NE_NOSTRING, coco::coco::num::NE_INVALIDINPUT, coco::coco::num::NE_SIMPLIFY, coco::coco::num::NE_INTERNAL_ERROR }
- enum coco::coco::coco::api_exception_type { coco::coco::coco::apiee_internal = 1, coco::coco::coco::apiee_evaluator = 2, coco::coco::coco::apiee_io = 3, coco::coco::coco::apiee_delta = 4, coco::coco::coco::apiee_search_graph = 5, coco::coco::coco::apiee_communication_data = 6, coco::coco::coco::apiee_inference_engine = 7, coco::coco::coco::apiee_graph_analyzer = 8, coco::coco::coco::apiee_management_module = 9, coco::coco::coco::apiee_initializer = 10, coco::coco::coco::apiee_report_module = 11, coco::coco::coco::apiee_oom = 12, coco::coco::coco::apiee_nyi = 13, coco::coco::coco::apiee_other = 14 }

  Enum for classifying api_exceptions.
- enum coco::coco::tristate_e { coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0 }
- enum coco::num::numtypes { coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL }
- enum coco::num::number_exception_type { coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR }

- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

  Enum for classifying api_exceptions.
- enum coco::coco::tristate_e { coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0, coco::coco::t_true = 1, coco::coco::t_false = -1, coco::coco::t_maybe = 0 }
- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco-

::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

Enum for classifying api_exceptions.

- enum coco::num::numtypes { coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL, coco::num::NT_RAT, coco::num::NT_STR, coco::num::NT_INTV, coco::num::NT_DBL }
- enum coco::num::number_exception_type { coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR, coco::num::NE_IRRATIONAL, coco::num::NE_NOSTRING, coco::num::NE_INVALIDINPUT, coco::num::NE_SIMPLIFY, coco::num::NE_INTERNAL_ERROR }
- enum coco::coco::api_exception_type { coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14, coco::coco::apiee_internal = 1, coco::coco::apiee_evaluator = 2, coco::coco::apiee_io = 3, coco::coco::apiee_delta = 4, coco::coco::apiee_search_graph = 5, coco::coco::apiee_communication_data = 6, coco::coco::apiee_inference_engine = 7, coco::coco::apiee_graph_analyzer = 8, coco::coco::apiee_management_module = 9, coco::coco::apiee_initializer = 10, coco::coco::apiee_report_module = 11, coco::coco::apiee_oom = 12, coco::coco::apiee_nyi = 13, coco::coco::apiee_other = 14 }

  Enum for classifying api_exceptions.
- enum coco::coco::search_node_relation { coco::coco::snr_root, coco::coco::snr_reduction, coco::coco::snr_relaxation, coco::coco::snr_split, coco::coco::snr_glue, coco::coco::snr_worknode, coco::coco::snr_virtual }

  Enum specifying node relations in the search graph.

Functions

- std::ostream & coco::coco::operator<< (std::ostream &o, const certificate &t)

  Output Operator for certificates.
- std::ostream & coco::coco::operator<< (std::ostream &o, const delta &t)

  Output Operator for deltas.
- double coco::coco::safeguarded_mid (const interval &__i)
- interval coco::coco::ipow (const interval &x, int n)
- interval coco::coco::gauss (const interval &x)
- interval coco::coco::atan2 (const interval &y, const interval &x)
- double coco::coco::absmin (const interval &__i)
- double coco::coco::gainfactor (const interval &_old, const interval &_new)
- template<class _TC >
  bool coco::coco::operator== (const interval &__i, const _TC &__d)
- template<class _TC >
  bool coco::coco::operator!= (const interval &__i, const _TC &__d)
- bool coco::coco::operator== (const interval &a, double b)
- bool coco::coco::operator!= (const interval &a, double b)
- bool coco::coco::operator< (const interval &a, double b)
- bool coco::coco::operator< (double a, const interval &b)
- interval coco::coco::operator+ (const interval &a, double b)
- interval coco::coco::operator+ (double b, const interval &a)
- interval coco::coco::operator- (const interval &a, double b)
- interval coco::coco::operator- (double b, const interval &a)
- interval coco::coco::cancel (const interval &a, const interval &b)
- interval coco::coco::operator∗ (const interval &a, double b)
- interval coco::coco::operator∗ (double b, const interval &a)
- interval coco::coco::operator/ (const interval &a, double b)
- interval coco::coco::operator/ (double b, const interval &a)
- std::ostream & coco::coco::operator<< (std::ostream &s, const interval &a)
- double coco::coco::mid (const interval &)
- double coco::coco::diam (const interval &)
- double coco::coco::width (const interval &)
- double coco::coco::relDiam (const interval &)
- double coco::coco::rad (const interval &)
- double coco::coco::mig (const interval &)
- double coco::coco::mag (const interval &)
- double coco::coco::dist (const interval &x, const interval &y)
- interval coco::coco::round_to_integer (const interval &x)
- interval coco::coco::acos (const interval &x)
- interval coco::coco::abs (const interval &x)
- interval coco::coco::acosh (const interval &x)
- interval coco::coco::acot (const interval &x)
- interval coco::coco::acoth (const interval &x)
- interval coco::coco::asin (const interval &x)
- interval coco::coco::asinh (const interval &x)
- interval coco::coco::atan (const interval &x)
- interval coco::coco::atanh (const interval &x)
- interval coco::coco::cos (const interval &x)
- interval coco::coco::cosh (const interval &x)

- interval coco::coco::cot (const interval &x)
- interval coco::coco::coth (const interval &x)
- interval coco::coco::exp (const interval &x)
- interval coco::coco::exp10 (const interval &x)
- interval coco::coco::exp2 (const interval &x)
- interval coco::coco::expm1 (const interval &x)
- interval coco::coco::log (const interval &x)
- interval coco::coco::log10 (const interval &x)
- interval coco::coco::log1p (const interval &x)
- interval coco::coco::log2 (const interval &x)
- interval coco::coco::power (const interval &x, int n)
- interval coco::coco::pow (const interval &x, const interval &y)
- interval coco::coco::sin (const interval &x)
- interval coco::coco::sinh (const interval &x)
- interval coco::coco::sqr (const interval &x)
- interval coco::coco::sqrt (const interval &x)
- interval coco::coco::tan (const interval &x)
- interval coco::coco::tanh (const interval &x)
- interval coco::coco::imax (const interval &x, const interval &y)
- interval coco::coco::imin (const interval &x, const interval &y)
- interval coco::coco::division_part2 (const interval &x, const interval &y, bool b=true)
- bool coco::coco::operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.
- bool coco::coco::operator== (const convex_e &__c, const convex_info &__d)
- bool coco::coco::operator== (const convex_info &__c, const convex_e &__d)
- bool coco::coco::operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.
- bool coco::coco::operator!= (const convex_e &__c, const convex_info &__d)
- bool coco::coco::operator!= (const convex_info &__c, const convex_e &__d)
- std::ostream & coco::coco::operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.
- std::ostream & coco::coco::operator<< (std::ostream &o, const semantics &__s)

  C++ stream output operator for semantics.
- template<class _Walker , class _Visitor >
  _Visitor::return_value coco::recursive_short_cut_walk (_Walker __w, _Visitor __f)

  Perform a recursive graph walk with possible caching and short-cuts.
- template<class _Walker , class _Visitor >
  _Visitor::return_value coco::_recursive_short_cut_walk (_Walker __w, _Visitor __f)

  Perform a recursive graph walk with possible caching and short-cuts (internal)
- template<class _Visitor , class _Walker >
  _Visitor::return_value coco::evaluate (_Visitor __v, _Walker __start)

  Evaluate an evaluator on a DAG.
- template<class _TH >
  std::string coco::coco::convert_to_str (const _TH &_h)

  Convert an object of any printable class to a string.
- std::string coco::coco::i2a (int x)

  converter int -> string
- std::string coco::coco::localdatetime ()

  converter date & time into std::string

- std::string coco::coco::datetime (const struct tm *timeptr)

  local time as C++ string
- std::string coco::coco::e2D (double d)

  Fortran real as C++ string.
- const basic_alltype & coco::coco::basic_alltype_empty ()
- std::ostream & coco::coco::operator<< (std::ostream &os, const basic_alltype &b)
- bool coco::coco::less_than (const basic_alltype &a, const basic_alltype &b)
- bool coco::coco::less_equal (const basic_alltype &a, const basic_alltype &b)
- double coco::coco::coco::safeguarded_mid (const interval &__i)
- interval coco::coco::coco::ipow (const interval &x, int n)
- interval coco::coco::coco::gauss (const interval &x)
- interval coco::coco::coco::atan2 (const interval &y, const interval &x)
- double coco::coco::coco::absmin (const interval &__i)
- double coco::coco::coco::gainfactor (const interval &_old, const interval &_new)
- template<class _TC >
  bool coco::coco::coco::operator== (const interval &__i, const _TC &__d)
- template<class _TC >
  bool coco::coco::coco::operator!= (const interval &__i, const _TC &__d)
- bool coco::coco::coco::operator== (const interval &a, const interval &b)
- bool coco::coco::coco::operator== (const interval &a, double b)
- bool coco::coco::coco::operator!= (const interval &a, const interval &b)
- bool coco::coco::coco::operator!= (const interval &a, double b)
- bool coco::coco::coco::operator< (const interval &a, const interval &b)
- bool coco::coco::coco::operator< (const interval &a, double b)
- bool coco::coco::coco::operator< (double a, const interval &b)
- interval coco::coco::coco::operator+ (const interval &a, const interval &b)
- interval coco::coco::coco::operator+ (const interval &a, double b)
- interval coco::coco::coco::operator+ (double b, const interval &a)
- interval coco::coco::coco::operator- (const interval &a, const interval &b)
- interval coco::coco::coco::operator- (const interval &a, double b)
- interval coco::coco::coco::operator- (double b, const interval &a)
- interval coco::coco::coco::cancel (const interval &a, const interval &b)
- interval coco::coco::coco::operator* (const interval &a, const interval &b)
- interval coco::coco::coco::operator* (const interval &a, double b)
- interval coco::coco::coco::operator* (double b, const interval &a)
- interval coco::coco::coco::operator/ (const interval &a, const interval &b)
- interval coco::coco::coco::operator/ (const interval &a, double b)
- interval coco::coco::coco::operator/ (double b, const interval &a)
- std::ostream & coco::coco::coco::operator<< (std::ostream &s, const interval &a)
- double coco::coco::coco::mid (const interval &)
- double coco::coco::coco::diam (const interval &)
- double coco::coco::coco::width (const interval &)
- double coco::coco::coco::relDiam (const interval &)
- double coco::coco::coco::rad (const interval &)
- double coco::coco::coco::mig (const interval &)
- double coco::coco::coco::mag (const interval &)
- double coco::coco::coco::dist (const interval &x, const interval &y)
- interval coco::coco::coco::round_to_integer (const interval &x)
- interval coco::coco::coco::acos (const interval &x)
- interval coco::coco::coco::abs (const interval &x)

- interval coco::coco::coco::acosh (const interval &x)
- interval coco::coco::coco::acot (const interval &x)
- interval coco::coco::coco::acoth (const interval &x)
- interval coco::coco::coco::asin (const interval &x)
- interval coco::coco::coco::asinh (const interval &x)
- interval coco::coco::coco::atan (const interval &x)
- interval coco::coco::coco::atanh (const interval &x)
- interval coco::coco::coco::cos (const interval &x)
- interval coco::coco::coco::cosh (const interval &x)
- interval coco::coco::coco::cot (const interval &x)
- interval coco::coco::coco::coth (const interval &x)
- interval coco::coco::coco::exp (const interval &x)
- interval coco::coco::coco::exp10 (const interval &x)
- interval coco::coco::coco::exp2 (const interval &x)
- interval coco::coco::coco::expm1 (const interval &x)
- interval coco::coco::coco::log (const interval &x)
- interval coco::coco::coco::log10 (const interval &x)
- interval coco::coco::coco::log1p (const interval &x)
- interval coco::coco::coco::log2 (const interval &x)
- interval coco::coco::coco::power (const interval &x, int n)
- interval coco::coco::coco::pow (const interval &x, const interval &y)
- interval coco::coco::coco::sin (const interval &x)
- interval coco::coco::coco::sinh (const interval &x)
- interval coco::coco::coco::sqr (const interval &x)
- interval coco::coco::coco::sqrt (const interval &x)
- interval coco::coco::coco::tan (const interval &x)
- interval coco::coco::coco::tanh (const interval &x)
- interval coco::coco::coco::imax (const interval &x, const interval &y)
- interval coco::coco::coco::imin (const interval &x, const interval &y)
- interval coco::coco::coco::division_part1 (const interval &x, const interval &y, bool &b)
- interval coco::coco::coco::division_part2 (const interval &x, const interval &y, bool b=true)
- std::string coco::coco::coco::i2a (int x)

  converter int -> string
- std::string coco::coco::coco::localdatetime ()

  converter date & time into std::string
- std::string coco::coco::coco::datetime (const struct tm *timeptr)

  local time as C++ string
- std::string coco::coco::coco::e2D (double d)

  Fortran real as C++ string.
- const basic_alltype & coco::coco::coco::basic_alltype_empty ()
- std::ostream & coco::coco::coco::operator<< (std::ostream &os, const basic_alltype &b)
- bool coco::coco::coco::less_than (const basic_alltype &a, const basic_alltype &b)
- bool coco::coco::coco::less_equal (const basic_alltype &a, const basic_alltype &b)
- work_node coco::full_node_to_work_node (full_node &n_full, gptr< search_node > &ground)

  This function converts a full_node to a work_node.

- bool coco::coco::operator== (const interval &a, const interval &b)

- bool coco::coco::operator!= (const interval &a, const interval &b)
- bool coco::coco::operator< (const interval &a, const interval &b)
- interval coco::coco::operator+ (const interval &a, const interval &b)
- interval coco::coco::operator- (const interval &a, const interval &b)
- interval coco::coco::operator∗ (const interval &a, const interval &b)
- interval coco::coco::operator/ (const interval &a, const interval &b)
- interval coco::coco::division_part1 (const interval &x, const interval &y, bool &b)

11.176.1 Detailed Description

Definition in file search_graph.cc.

11.176.2 Define Documentation

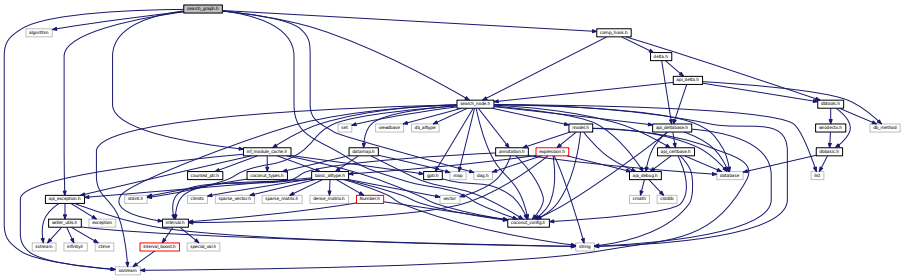
11.176.2.1 #define DEBUG_SGRAPH_EXTRACT 0

Definition at line 36 of file search_graph.cc.

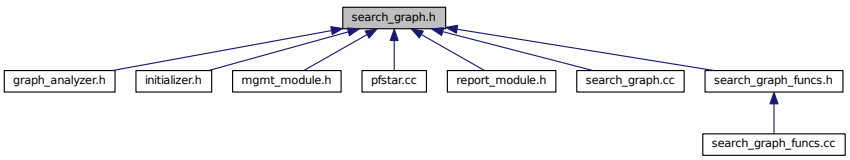
11.177 search_graph.h File Reference

```
#include <iostream> #include <algorithm> #include <dag.h> #include <coconut-
_config.h> #include <search_node.h> #include <comp_hook.h> #include <api-
```

_exception.h> #include <inf_module_cache.h> Include dependency graph for search_-graph.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::search_graph

  The search graph.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef search_graph::const_walker coco::search_inspector

  The search inspector for graph analysis.
- typedef search_graph::walker coco::search_focus

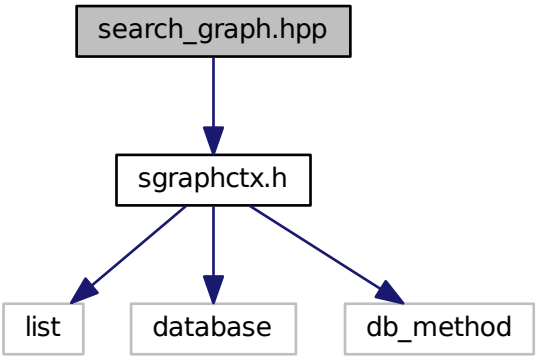
  The search focus for work node selection.

11.177.1 Detailed Description

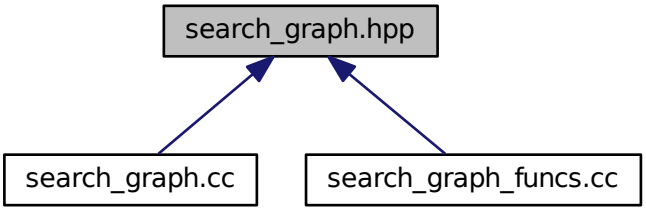
Definition in file search_graph.h.

11.178 search_graph.hpp File Reference

`#include <sgraphctx.h>` Include dependency graph for search_graph.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

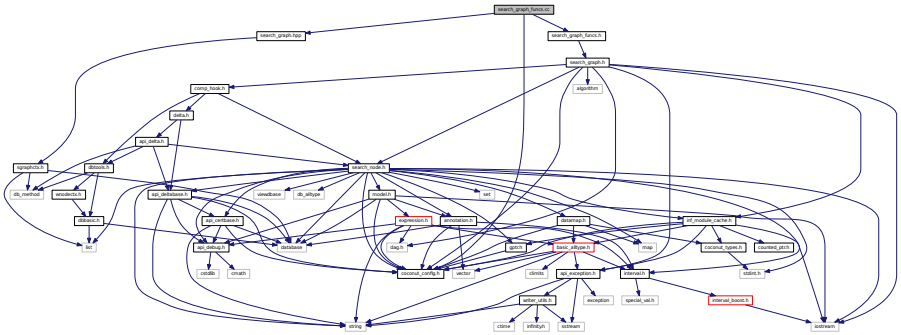
  the main namespace of the COCONUT API

11.178.1 Detailed Description

Definition in file search_graph.hpp.

11.179 search_graph_funcs.cc File Reference

`#include <coconut_config.h>` `#include <search_graph_funcs.h>` `#include <search_graph.hpp>` Include dependency graph for search_graph_funcs.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- void coco::get_ids (const std::vector< si_ri_pair > &riv, std::vector< search_node_id > &v)
- void coco::get_leaves_vector (const search_graph &__sgraph, std::vector< si_ri_pair > &v, search_inspector *sfoc=NULL)

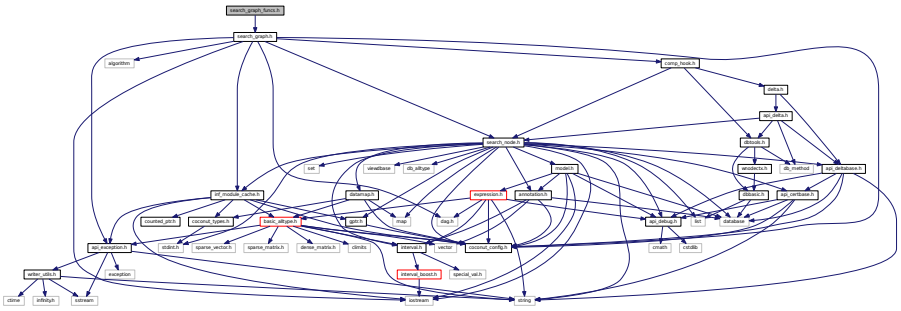
  Sparsity calculation (Hessian and Jacobian)
- bool coco::get_children_vector (const search_graph &__sgraph, const search_node_id &pid, std::vector< si_ri_pair > &v, search_inspector *sfoc)

11.179.1 Detailed Description

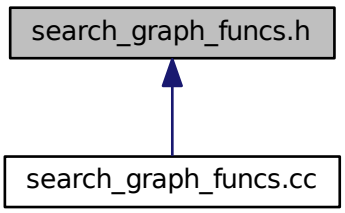
Definition in file search_graph_funcs.cc.

11.180 search_graph_funcs.h File Reference

`#include <search_graph.h>` Include dependency graph for search_graph_funcs.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef std::pair< vdbl::rowid, search_node_id > coco::si_ri_pair

Functions

- void coco::get_ids (const std::vector< si_ri_pair > &riv, std::vector< search_node_id > &v)
- void coco::get_leaves_vector (const search_graph &__sgraph, std::vector< si_ri_pair > &v, search_inspector *sfoc=NULL)

Sparsity calculation (Hessian and Jacobian)

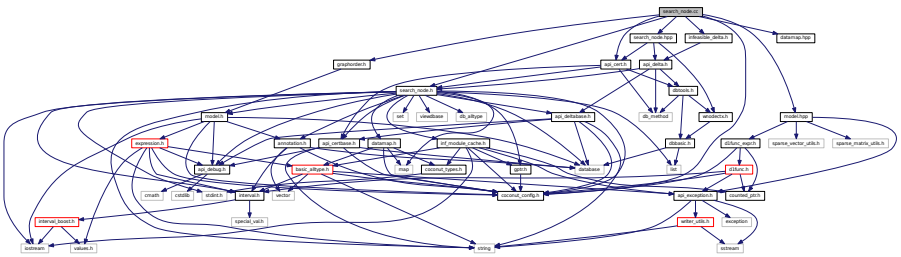
- bool coco::get_children_vector (const search_graph &__sgraph, const search_node_id &pid, std::vector< si_ri_pair > &v, search_inspector *sfoc)

11.180.1 Detailed Description

Definition in file search_graph_funcs.h.

11.181 search_node.cc File Reference

`#include <coconut_config.h>` `#include <search_node.h>` `#include <model.hpp>` `#include <search_node.hpp>` `#include <infeasible_delta.h>` `#include <graphorder.h>` `#include <api_cert.h>` `#include <datamap.hpp>` Include dependency graph for search_node.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define MAX_VOL_COMP 1.e12
- #define MIN_VOL_WIDTH DBL_MIN

Functions

- work_node coco::operator- (const work_node &_w, const delta_id &_d)

  Remove one delta from a work node.
- work_node & coco::operator-= (work_node &_w, const delta_id &_d)

  Remove one delta from a work node.

11.181.1 Detailed Description

Definition in file search_node.cc.

11.181.2 Define Documentation

11.181.2.1 #define MAX_VOL_COMP 1.e12

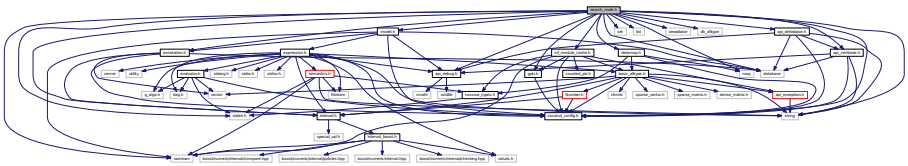
Definition at line 282 of file search_node.cc.

11.181.2.2 #define MIN_VOL_WIDTH DBL_MIN

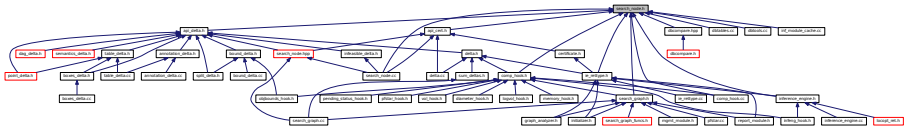
Definition at line 283 of file search_node.cc.

11.182 search_node.h File Reference

`#include <iostream>` `#include <coconut_config.h>` `#include <interval.h>` × `#include <model.h>` `#include <annotation.h>` `#include <map>` `#include <set>` × `#include <list>` `#include <string>` `#include <gptr.h>` `#include <stdint.h>` `#include <database>` `#include <viewdbase>` `#include <db_alltype>` `#include <api_debug.h>` `#include <datamap.h>` `#include <inf_module_cache.h>` `#include <api_deltabase.h>` `#include <api_certbase.h>` Include dependency graph for search_node.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::search_node

  Base type of the nodes in the search graph.
- class coco::delta_node

  Class holding the delta nodes in the search graph.
- class coco::full_node

  Class holding the full nodes in the search graph.
- class coco::work_node

  Work node, which is passed to the inference engines.
- class coco::work_node::constraint_iterator_base

  The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Enumerations

- enum coco::search_node_relation { coco::snr_root, coco::snr_reduction, coco::snr_relaxation, coco::snr_split, coco::snr_glue, coco::snr_worknode, coco::snr_virtual }

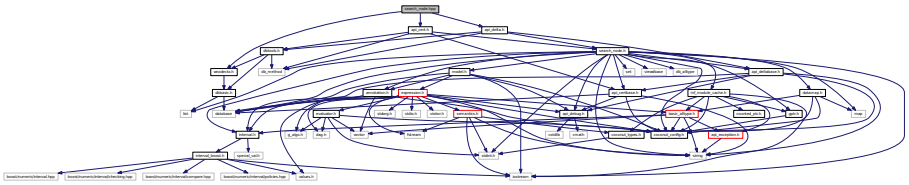
  Enum specifying node relations in the search graph.

11.182.1 Detailed Description

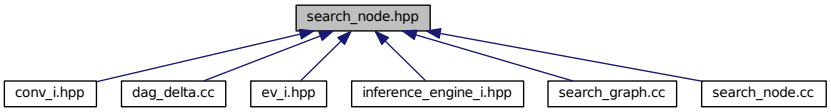
Definition in file search_node.h.

11.183 search_node.hpp File Reference

`#include <wnodectx.h>` `#include <api_delta.h>` `#include <api_cert.h>` Include dependency graph for search_node.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- work_node coco::operator+ (const work_node &_w, const delta_id &_i)

  Add one delta to a work node.
- work_node & coco::operator+= (work_node &_w, const delta_id &_i)

  Add one delta to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node coco::operator+ (const work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Add a number of deltas to a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node & coco::operator+= (work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Add a number of deltas to a work node.
- work_node coco::operator- (const work_node &_w, const delta_id &_d)

  Remove one delta from a work node.
- work_node & coco::operator-= (work_node &_w, const delta_id &_d)

  Remove one delta from a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node coco::operator- (const work_node &_w, const _Ctr< delta_id, _Al > &_d)

  Remove a number of deltas from a work node.
- template<template< class _Tp, class _TA > class _Ctr, class _Al >
  work_node & coco::operator-= (work_node &_w, const _Ctr< delta_id, _Al > &_d)

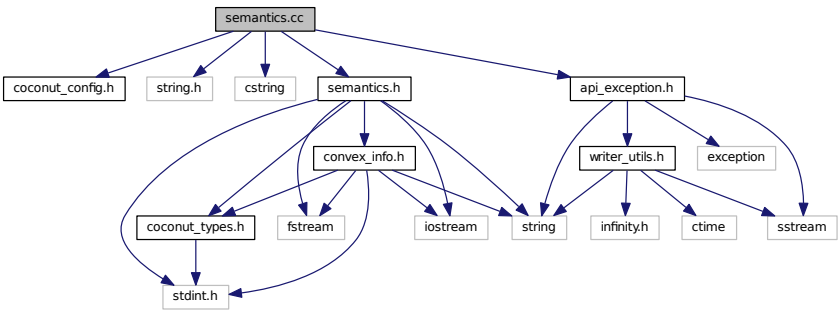
  Remove a number of deltas from a work node.

11.183.1 Detailed Description

Definition in file search_node.hpp.

11.184 semantics.cc File Reference

`#include <coconut_config.h>` `#include <string.h>` `#include <cstring>` `#include <semantics.h>` `#include <api_exception.h>` Include dependency graph for semantics.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- std::ostream & coco::operator<< (std::ostream &o, const semantics &__s)

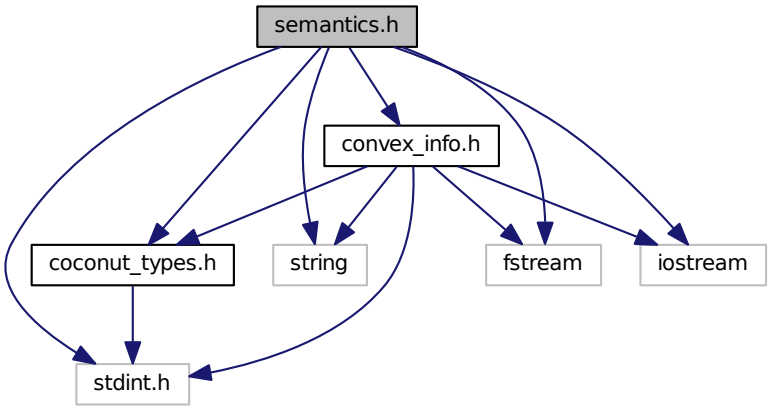
  C++ stream output operator for semantics.

11.184.1 Detailed Description

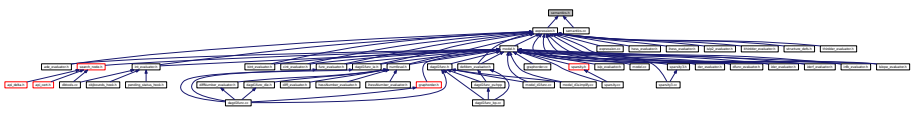
Definition in file semantics.cc.

11.185 semantics.h File Reference

`#include <stdint.h>` `#include <string>` `#include <fstream>` `#include <iostream>` × `#include <coconut_types.h>` `#include <convex_info.h>` Include dependency graph for semantics.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::semantics

  Expression Semantics.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Enumerations

- enum coco::type_annotation { coco::v_exists = 0, coco::v_forall = 1, coco::v_free = 2, coco::v_-stochastic = 3 }

  Node type information enum.
- enum coco::activity_descr { coco::a_redundant = 1, coco::a_active_lo = 2, coco::a_active_lo_red = a_active_lo|a_redundant, coco::a_active_hi = 4, coco::a_active_hi_red = a_active_hi|a_redundant, coco::a_active = a_active_lo|a_active_hi, coco::a_active_red = a_active|a_redundant }

  Constraint activity information enum.

Functions

- std::ostream & coco::coco::operator<< (std::ostream &o, const semantics &__s)

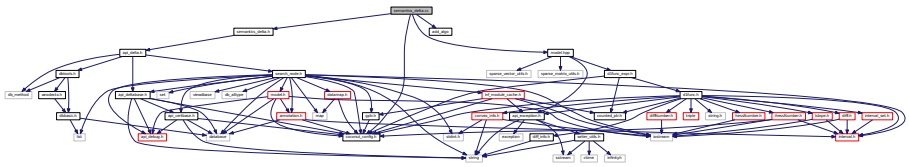
  C++ stream output operator for semantics.

11.185.1 Detailed Description

Definition in file semantics.h.

11.186 semantics_delta.cc File Reference

`#include <coconut_config.h>` `#include <semantics_delta.h>` `#include <add-_algo>` `#include <model.hpp>` Include dependency graph for semantics_delta.cc:



Namespaces

- namespace coco

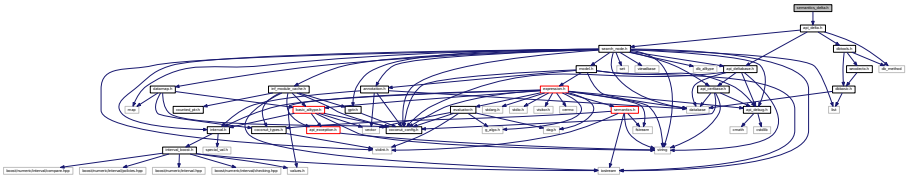
  the main namespace of the COCONUT API

11.186.1 Detailed Description

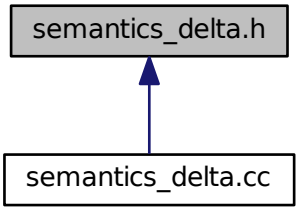
Definition in file semantics_delta.cc.

11.187 semantics_delta.h File Reference

`#include <api_delta.h>` Include dependency graph for semantics_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::semantics_undelta

  The semantics undelta class for undoing changes to the node semantics in a model.
- class coco::semantics_delta

  The semantics delta class for changing the node semantics within a model.

Namespaces

- namespace coco

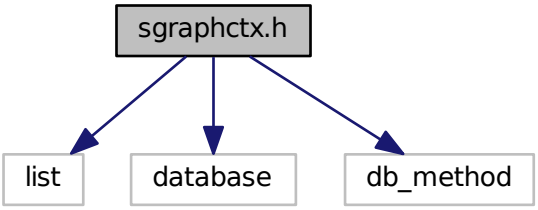
  the main namespace of the COCONUT API

11.187.1 Detailed Description

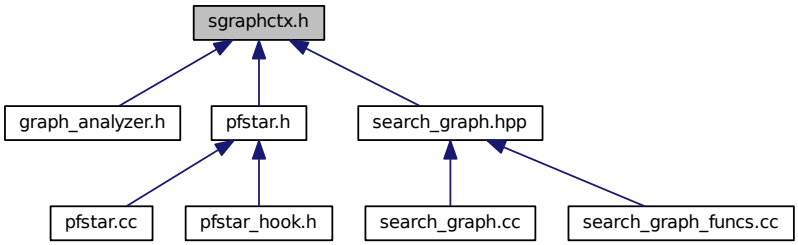
Definition in file semantics_delta.h.

11.188 sgraphctx.h File Reference

`#include <list>` `#include <database>` `#include <db_method>` Include dependency graph for sgraphctx.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::search_graph_context

  An evaluation context when retrieving from the search database.

Namespaces

- namespace coco

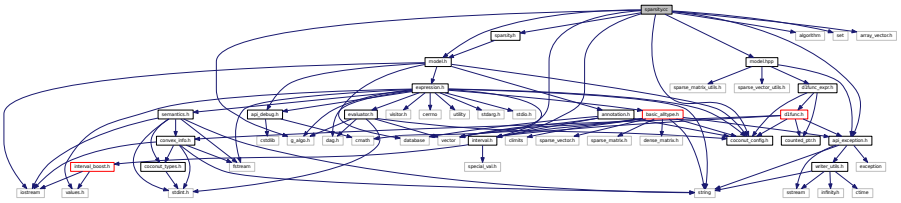
the main namespace of the COCONUT API

11.188.1 Detailed Description

Definition in file sgraphctx.h.

11.189 sparsity.cc File Reference

`#include <coconut_config.h>` `#include <model.h>` `#include <vector>` `#include <algorithm>` `#include <g_algo.h>` `#include <cmath>` `#include <set>` `#include <api_exception.h>` `#include <array_vector.h>` `#include <sparsity.h>` `#include <model.hpp>` Include dependency graph for sparsity.cc:



Classes

- class coco::sparsity_visitor

  Preorder visitor which calculates sparsity structures.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define SP_DBG 0

Typedefs

- typedef std::pair< unsigned int, unsigned int > coco::ipair

  Row-column index pair.
- typedef std::set< ipair > coco::splist

  Sparsity information.

Functions

- void coco::sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity (const model &DAG, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)
- void coco::sparsity (const model &DAG, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)
- void coco::sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity (const model &DAG, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

11.189.1 Detailed Description

Definition in file sparsity.cc.

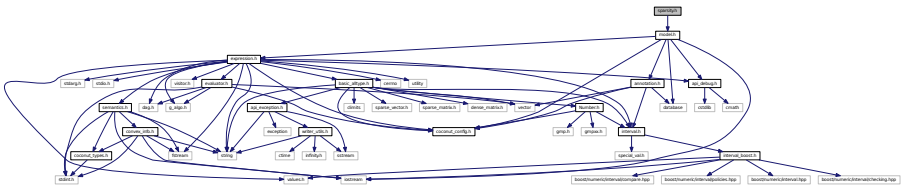
11.189.2 Define Documentation

11.189.2.1 #define SP_DBG 0

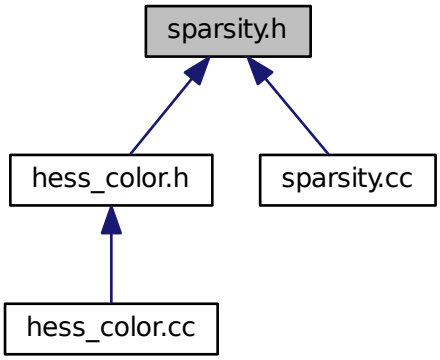
Definition at line 40 of file sparsity.cc.

11.190 sparsity.h File Reference

`#include <model.h>` Include dependency graph for sparsity.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Enumerations

- enum coco::sp_indexing { coco::full_indexing = 0, coco::row_wise_indexing = 1, coco::column_wise_indexing = 2, coco::full_indexing = 0, coco::row_wise_indexing = 1, coco::column_wise_indexing = 2 }

Functions

- void coco::sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity (const model &DAG, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity (const model &DAG, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)

- void coco::sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

- void coco::sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)

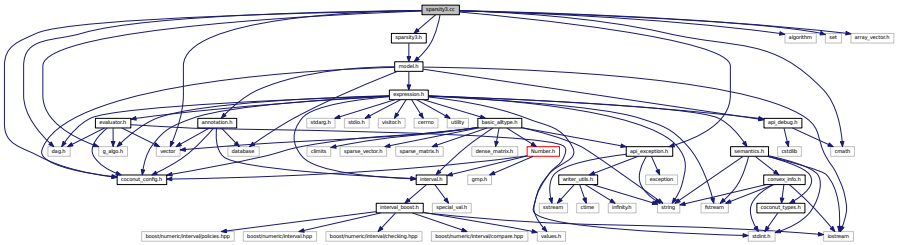
- void coco::sparsity (const model &DAG, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

11.190.1 Detailed Description

Definition in file sparsity.h.

11.191 sparsity3.cc File Reference

`#include <coconut_config.h>` `#include <dag.h>` `#include <model.h>` `#include <vector>` `#include <algorithm>` `#include <g_algo.h>` `#include <cmath>` `#include <set>` `#include <api_exception.h>` `#include <array_vector.h>` `#include <sparsity3.-h>` Include dependency graph for sparsity3.cc:



Classes

- class coco::sparsity3_visitor

  Preorder visitor which calculates sparsity structures.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define SP_DBG 0

Typedefs

- typedef std::triple< unsigned int, unsigned int, unsigned int > coco::itriple

  Row-column-depth index triple.
- typedef std::set< itriple > coco::stlist

  Sparsity information.

Functions

- void coco::sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity (const model &DAG, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)
- void coco::sparsity (const model &DAG, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)
- void coco::sparsity (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity (const model &DAG, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

11.191.1 Detailed Description

Definition in file sparsity3.cc.

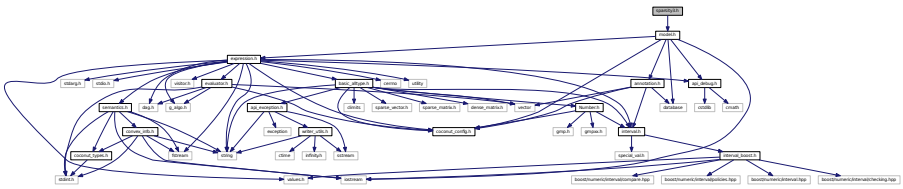
11.191.2 Define Documentation

11.191.2.1 #define SP_DBG 0

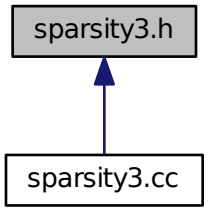
Definition at line 40 of file sparsity3.cc.

11.192 sparsity3.h File Reference

`#include <model.h>` Include dependency graph for sparsity3.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Enumerations

- enum coco::sp_indexing { coco::full_indexing = 0, coco::row_wise_indexing = 1, coco::column_wise_indexing = 2, coco::full_indexing = 0, coco::row_wise_indexing = 1, coco::column_wise_indexing = 2 }

Functions

- void coco::sparsity3 (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity3 (const model &DAG, std::vector< unsigned int > &H_ridx, std::vector< unsigned int > &H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity3 (const model &DAG, std::vector< unsigned int > &J_funidx, std::vector< unsigned int > &J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)

- void coco::sparsity3 (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)
- void coco::sparsity3 (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Hessian and Jacobian)

- void coco::sparsity3 (const model &DAG, unsigned int &nnz_H, unsigned int *&H_ridx, unsigned int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)
- void coco::sparsity3 (const model &DAG, int &nnz_H, int *&H_ridx, int *&H_cidx, bool org_only, bool H_upper_only, sp_indexing indexing, bool var_one_based)

  Sparsity calculation (Hessian)

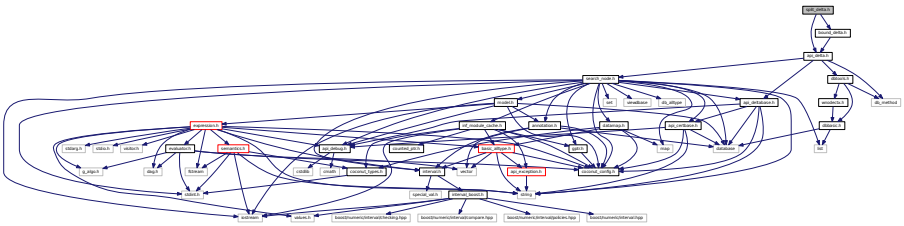
- void coco::sparsity3 (const model &DAG, unsigned int &nnz_J, unsigned int *&J_funidx, unsigned int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)
- void coco::sparsity3 (const model &DAG, int &nnz_J, int *&J_funidx, int *&J_varidx, bool J_with_obj, bool J_obj_is_first, bool org_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)

  Sparsity calculation (Jacobian)

11.193 split_delta.h File Reference

`#include <api_delta.h>` `#include <bound_delta.h>` Include dependency graph for split-_delta.h:



Classes

- class coco::split_undelta

  The split undelta class for removing proposed splits from a work_node.
- class coco::split_delta

  The split delta class for proposing useful splits.

Namespaces

- namespace coco

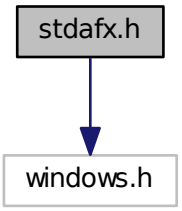
  the main namespace of the COCONUT API

11.193.1 Detailed Description

Definition in file split_delta.h.

11.194 stdafx.h File Reference

`#include <windows.h>` Include dependency graph for stdafx.h:



Defines

- #define WIN32_LEAN_AND_MEAN

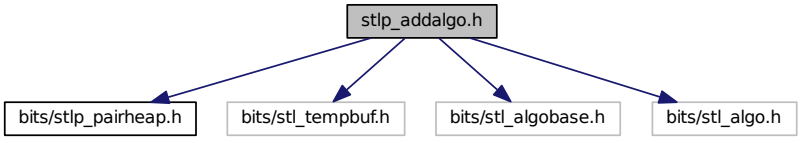
11.194.1 Define Documentation

11.194.1.1 #define WIN32_LEAN_AND_MEAN

Definition at line 4 of file stdafx.h.

11.195 stlp_addalgo.h File Reference

`#include <bits/stlp_pairheap.h>` `#include <bits/stl_tempbuf.h>` `#include <bits/stl-_algobase.h>` `#include <bits/stl_algo.h>` Include dependency graph for stlp_addalgo.h:



Namespaces

- namespace std

The standard namespace.

Defines

- #define __ISTLGLIBCXX_function_requires(...) __glibcpp_function_requires(__gnu_cxx::__VA-_ARGS__)
- #define __ISTLGLIBCXX_requires_valid_range(A, B)
- #define __GNU_CXX_NAMESPACE

Enumerations

- enum { std::_M_p_chunk_size = 7 }
- enum { std::_M_p_threshold = 16 }

Functions

- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp >
  pair< _RandomAccessIter, _RandomAccessIterp > std::__unguarded_p_partition (_RandomAccess-Iter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __-lastp, _Tp __pivot)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp , typename _Compare >
  pair< _RandomAccessIter, _RandomAccessIterp > std::__unguarded_p_partition (_RandomAccess-Iter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __-lastp, _Tp __pivot, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp , typename _Tpp >
  void std::__unguarded_linear_p_insert (_RandomAccessIter __last1, _Tp __val1, _RandomAccess-Iterp __last2, _Tpp __val2)
- template<typename _RandomAccessIter , typename _Tp , typename _RandomAccessIterp , typename _Tpp , typename _-Compare >
  void std::__unguarded_linear_p_insert (_RandomAccessIter __last1, _Tp __val1, _RandomAccess-Iterp __last2, _Tpp __val2, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void std::__insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccess-Iterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void std::__insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccess-Iterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void std::__unguarded_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _-RandomAccessIterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void std::__unguarded_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _-RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void std::__final_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _Random-AccessIterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void std::__final_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _Random-AccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size >
  void std::__intro_p_sort_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size , typename _Compare >
  void std::__intro_p_sort_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size >
  void std::__introsort_p_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size , typename _Compare >
  void std::__introsort_p_loop (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Size __depth_limit, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  bool std::pair_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

  Sort the elements of two sequences in common.
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  bool std::pair_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

  Sort the elements of a sequence using a predicate for comparison.
- template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance >
  void std::__merge_p_without_buffer (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2)
- template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Compare >
  void std::__merge_p_without_buffer (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2, _Compare __comp)
- template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Pointer , typename _Pointerp >
  void std::__p_merge_adaptive (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2, _Pointer __buffer, _Pointerp __bufferp, _Distance __buffer_size)
- template<typename _BidirectionalIter , typename _BidirectionalIterp , typename _Distance , typename _Pointer , typename _Pointerp , typename _Compare >
  void std::__p_merge_adaptive (_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _BidirectionalIterp __firstp, _BidirectionalIterp __middlep, _BidirectionalIterp __lastp, _Distance __len1, _Distance __len2, _Pointer __buffer, _Pointerp __bufferp, _Distance __buffer_size, _Compare __comp)
- template<typename _InputIter1 , typename _InputIter2 , typename _OutputIter , typename _InputIter1p , typename _InputIter2p , typename _OutputIterp >
  pair< _OutputIter, _OutputIterp > std::pair_merge (_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _InputIter1p __first1p, _InputIter1p __last1p, _InputIter2p __first2p, _InputIter2p __last2p, _OutputIterp __resultp)

  Merges two sorted ranges.
- template<typename _InputIter1 , typename _InputIter2 , typename _OutputIter , typename _InputIter1p , typename _InputIter2p , typename _OutputIterp , typename _Compare >
  pair< _OutputIter, _OutputIterp > std::pair_merge (_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _InputIter1p __first1p, _InputIter1p __last1p, _InputIter2p __first2p, _InputIter2p __last2p, _OutputIterp __resultp, _Compare __comp)

  Merges two sorted ranges.

- template<typename _BidirectionalIter1 , typename _BidirectionalIter2 , typename _BidirectionalIter3 , typename _BidirectionalIter1p , typename _BidirectionalIter2p , typename _BidirectionalIter3p >
  pair< _BidirectionalIter3, _BidirectionalIter3p > std::__p_merge_backward (_BidirectionalIter1 __first1, _BidirectionalIter1 __last1, _BidirectionalIter2 __first2, _BidirectionalIter2 __last2, _BidirectionalIter3 __result, _BidirectionalIter1p __first1p, _BidirectionalIter1p __last1p, _BidirectionalIter2p __first2p, _BidirectionalIter2p __last2p, _BidirectionalIter3p __resultp)
- template<typename _BidirectionalIter1 , typename _BidirectionalIter2 , typename _BidirectionalIter3 , typename _BidirectionalIter1p , typename _BidirectionalIter2p , typename _BidirectionalIter3p , typename _Compare >
  pair< _BidirectionalIter3, _BidirectionalIter3p > std::__p_merge_backward (_BidirectionalIter1 __first1, _BidirectionalIter1 __last1, _BidirectionalIter2 __first2, _BidirectionalIter2 __last2, _BidirectionalIter3 __result, _BidirectionalIter1p __first1p, _BidirectionalIter1p __last1p, _BidirectionalIter2p __first2p, _BidirectionalIter2p __last2p, _BidirectionalIter3p __resultp, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void std::__inplace_stable_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void std::__inplace_stable_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)
- template<typename _RandomAccessIter1 , typename _RandomAccessIter2 , typename _RandomAccessIter1p , typename _RandomAccessIter2p , typename _Distance >
  void std::__merge_p_sort_loop (_RandomAccessIter1 __first, _RandomAccessIter1 __last, _RandomAccessIter2 __result, _RandomAccessIter1p __firstp, _RandomAccessIter1p __lastp, _RandomAccessIter2p __resultp, _Distance __step_size)
- template<typename _RandomAccessIter1 , typename _RandomAccessIter2 , typename _RandomAccessIter1p , typename _RandomAccessIter2p , typename _Distance , typename _Compare >
  void std::__merge_p_sort_loop (_RandomAccessIter1 __first, _RandomAccessIter1 __last, _RandomAccessIter2 __result, _RandomAccessIter1p __firstp, _RandomAccessIter1p __lastp, _RandomAccessIter2p __resultp, _Distance __step_size, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Distance >
  void std::__chunk_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Distance __chunk_size)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Distance , typename _Compare >
  void std::__chunk_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Distance __chunk_size, _Compare __comp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Pointer , typename _Pointerp >
  void std::__merge_p_sort_with_buffer (_RandomAccessIter __first, _RandomAccessIter __last, _Pointer __buffer, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Pointerp __bufferp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Pointer , typename _Pointerp , typename _Compare >
  void std::__merge_p_sort_with_buffer (_RandomAccessIter __first, _RandomAccessIter __last, _Pointer __buffer, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Pointerp __bufferp, _Compare __comp)
- template<typename _RandomAccessIter , typename _Pointer , typename _RandomAccessIterp , typename _Pointerp , typename _Distance >
  void std::__stable_p_sort_adaptive (_RandomAccessIter __first, _RandomAccessIter __last, _Pointer __buffer, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Pointerp __bufferp, _Distance __buffer_size)
- template<typename _RandomAccessIter , typename _Pointer , typename _RandomAccessIterp , typename _Pointerp , typename _Distance , typename _Compare >
  void std::__stable_p_sort_adaptive (_RandomAccessIter __first, _RandomAccessIter __last, _Pointer __buffer, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Pointerp __bufferp, _Distance __buffer_size, _Compare __comp)

- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  bool std::stable_pair_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

  Sort the elements of a pair of sequences, preserving the relative order of equivalent elements.
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  bool std::stable_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

  Sort the elements of a pair of sequences using a predicate for comparison, preserving the relative order of equivalent elements.
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  bool std::partial_pair_sort (_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

  Sort the smallest elements of a pair of sequences.
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  bool std::partial_pair_sort (_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp, _Compare __comp)

  Sort the smallest elements of a pair of sequences.
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  bool std::partial_sort (_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last, _RandomAccessIter __firstp, _RandomAccessIter __lastp, _Compare __comp)

  Sort the smallest elements of a pair of sequences using a predicate for comparison.

11.195.1 Detailed Description

Definition in file stlp_addalgo.h.

11.195.2 Define Documentation

11.195.2.1 #define __GNU_CXX_NAMESPACE

Definition at line 55 of file stlp_addalgo.h.

11.195.2.2 #define __ISTLGLIBCXX_function_requires(...) __glibcpp_function_requires(__gnu_cxx::__VA_ARGS__)

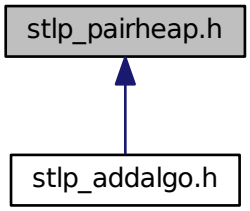
Definition at line 42 of file stlp_addalgo.h.

11.195.2.3 #define __ISTLGLIBCXX_requires_valid_range(*A, B*)

Definition at line 43 of file stlp_addalgo.h.

11.196 stlp_pairheap.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace std

  The standard namespace.

Functions

- template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp >
  void std::__push_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __topIndex, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __topIndexp, _Tpp __valuep)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp >
  void std::push_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)
- template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp , typename _Compare >
  void std::__push_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __topIndex, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __topIndexp, _Tpp __valuep, _Compare __comp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare >
  void std::push_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp)
- template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp >
  void std::__adjust_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __len, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __lenp, _Tpp __valuep)
- template<typename _RandomAccessIterator , typename _Tp , typename _RandomAccessIteratorp , typename _Tpp >
  void std::__pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIterator __result, _Tp __value, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _RandomAccessIteratorp __resultp, _Tpp __valuep)

- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp >
  void std::pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)
- template<typename _RandomAccessIterator , typename _Distance , typename _Tp , typename _RandomAccessIteratorp , typename _Distancep , typename _Tpp , typename _Compare >
  void std::__adjust_pair_heap (_RandomAccessIterator __first, _Distance __holeIndex, _Distance __len, _Tp __value, _RandomAccessIteratorp __firstp, _Distancep __holeIndexp, _Distancep __lenp, _Tpp __valuep, _Compare __comp)
- template<typename _RandomAccessIterator , typename _Tp , typename _RandomAccessIteratorp , typename _Tpp , typename _Compare >
  void std::__pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIterator __result, _Tp __value, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _RandomAccessIteratorp __resultp, _Tpp __valuep, _Compare __comp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare >
  void std::pop_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp >
  void std::make_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare >
  void std::make_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp >
  void std::sort_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp)
- template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare >
  void std::sort_pair_heap (_RandomAccessIterator __first, _RandomAccessIterator __last, _RandomAccessIteratorp __firstp, _RandomAccessIteratorp __lastp, _Compare __comp)

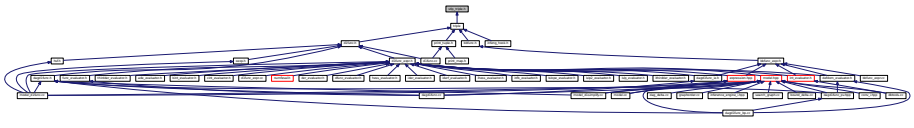
11.196.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

Definition in file stlp_pairheap.h.

11.197 stlp_triple.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct std::triple
  triple holds three objects of arbitrary type.

Namespaces

- namespace std

  The standard namespace.

Functions

- template<class _T1 , class _T2 , class _T3 >
  bool std::operator== (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Two triples of the same type are equal iff their members are equal.
- template<class _T1 , class _T2 , class _T3 >
  bool std::operator< (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  This is lexicographic ordering of triples.
- template<class _T1 , class _T2 , class _T3 >
  bool std::operator!= (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Uses `operator==` to find the result.
- template<class _T1 , class _T2 , class _T3 >
  bool std::operator> (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Uses `operator<` to find the result.
- template<class _T1 , class _T2 , class _T3 >
  bool std::operator<= (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

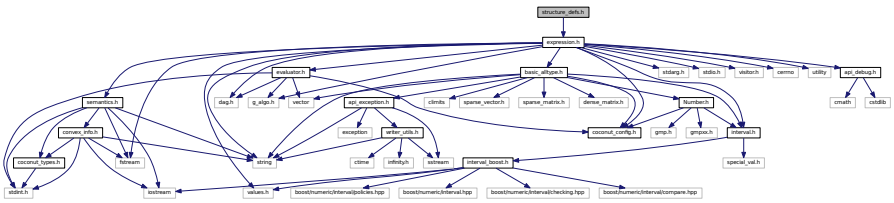
  Uses `operator<` to find the result.
- template<class _T1 , class _T2 , class _T3 >
  bool std::operator>= (const triple< _T1, _T2, _T3 > &__x, const triple< _T1, _T2, _T3 > &__y)

  Uses `operator<` to find the result.
- template<class _T1 , class _T2 , class _T3 >
  triple< _T1, _T2, _T3 > std::make_triple (const _T1 &__x, const _T2 &__y, const _T3 &__z)

  A convenience wrapper for creating a triple from three objects.

11.198 structure_defs.h File Reference

`#include <expression.h>` Include dependency graph for structure_defs.h:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

Counter Indices for DAG Structure Analysis

These are definitions of counter indices used by the various models dealing with DAG structure analysis.

- #define CTR_NODES 0
  number of nodes counter
- #define CTR_INTEGER 1
  number of integer nodes counter
- #define CTR_KJNODES 2
  number of KJ nodes counter
- #define CTR_FREE 3
  number of free nodes counter
- #define CTR_EXISTS 4
  number of exists nodes counter
- #define CTR_FORALL 5
  number of forall nodes counter
- #define CTR_STOCHASTIC 6
  number of stochastic nodes counter
- #define NUM_CTR 7
  number of basic counters
- #define CTR_VARS 0
  number of variables counter
- #define CTR_VBINARY 1
  number of integer variables counter
- #define CTR_VINTEGER 1
  number of non-binary integer variables counter
- #define CTR_VFREE 3
  number of free variables counter
- #define CTR_VEXISTS 4
  number of exists variables counter
- #define CTR_VFORALL 5
  number of forall variables counter
- #define CTR_VSTOCHASTIC 6
  number of stochastic variables counter
- #define CTR_VKJ 7
  number of KJ variables counter
- #define CTR_V2BOUNDED 8
  number of twosided bounded variables counter
- #define CTR_V1BOUNDED 9
  number of onesided bounded variables counter
- #define CTR_VUBOUNDED 10
  number of unbounded variables counter
- #define NUM_VCTR 11
  number of variables counters
- #define CTR_OBJ 0
  number of objectives counter
- #define CTR_OINTEGER 1
  number of integer objectives counter
- #define CTR_OFREE 2
  number of free objectives counter
- #define CTR_OEXISTS 3
  number of exists objectives counter

- #define CTR_OFORALL 4
  number of forall objectives counter
- #define CTR_OSTOCHASTIC 5
  number of stochastic objectives counter
- #define CTR_O2BOUNDED 6
  number of twosided bounded objectives counter
- #define CTR_O1BOUNDED 7
  number of onesided bounded objectives counter
- #define CTR_OUBOUNDED 8
  number of unbounded objectives counter
- #define CTR_ODIM1 9
  number of univariate objectives counter
- #define CTR_ODIM2 10
  number of twodimensional objectives counter
- #define CTR_OMTDIM2 11
  number of multidimensional objectives counter
- #define CTR_ODEG1 12
  number of linear objectives counter
- #define CTR_ODEG2 13
  number of quadratic objectives counter
- #define CTR_ODEG3 14
  number of cubic objectives counter
- #define CTR_OMTDEG3P 15
  number of polynomial noncubic objectives counter
- #define CTR_ONLN 16
  number of nonpolynomial objectives counter
- #define CTR_OCONVEX 17
  number of convex objectives counter
- #define CTR_OCONCAVE 18
  number of concave objectives counter
- #define CTR_MIN 19
  number of objectives to be minimized counter
- #define NUM_OCTR 20
  number of objectives counters
- #define CTR_CONSTR 0
  number of constraints counter
- #define CTR_CEQU 1
  number of equality constraints counter
- #define CTR_CINEQ 2
  number of inequality constraints counter
- #define CTR_EBOUND 3
  number of equality bound constraints counter
- #define CTR_EDEG1 4
  number of linear equality constraints counter
- #define CTR_EDEG2 5
  number of quadratic equality constraints counter
- #define CTR_EDEG3 6
  number of cubic equality constraints counter
- #define CTR_EMTDEG3P 7
  number of polynomial noncubic equality constraints counter
- #define CTR_ENLN 8
  number of nonpolynomial equality constraints counter
- #define CTR_EDIM1 9
  number of univariate equality constraints counter

- #define CTR_EDIM2 10
  number of twodimensional equality constraints counter
- #define CTR_EMTDIM2 11
  number of multidimensional equality constraints counter
- #define CTR_EINTEGER 12
  number of integer equality constraints counter
- #define CTR_IBOUND 13
  number of inequality bound constraints counter
- #define CTR_IDEG1 14
  number of linear equality constraints counter
- #define CTR_IDEG2 15
  number of quadratic equality constraints counter
- #define CTR_IDEG3 16
  number of cubic equality constraints counter
- #define CTR_IMTDEG3P 17
  number of polynomial noncubic inequality constraints counter
- #define CTR_INLN 18
  number of nonpolynomial inequality constraints counter
- #define CTR_IDIM1 19
  number of univariate inequality constraints counter
- #define CTR_IDIM2 20
  number of twodimensional inequality constraints counter
- #define CTR_IMTDIM2 21
  number of multidimensional inequality constraints counter
- #define CTR_IINTEGER 22
  number of integer inequality constraints counter
- #define CTR_ICONVEX 23
  number of integer inequality constraints counter
- #define CTR_ICONCAVE 24
  number of integer inequality constraints counter
- #define CTR_IACTIVE 25
  number of integer inequality constraints counter
- #define CTR_IINACTIVE 26
  number of active inequality constraints counter
- #define CTR_IACTIVE_LO 27
  number of inactive inequality constraints counter
- #define CTR_IINACTIVE_LO 28
  number of inequality constraints active at the lower bound counter
- #define CTR_IACTIVE_HI 29
  number of inequality constraints inactive at the lower bound counter
- #define CTR_IINACTIVE_HI 30
  number of inequality constraints active at the upper bound counter
- #define CTR_IREDUNDANT 31
  number of inequality constraints inactive at the upper bound counter
- #define CTR_IRESTRICT 32
  number of redundant inequality constraints counter
- #define NUM_CCTR 33
  number of constraints counters
- #define CTR_EDGES 0
  number of edges counter
- #define CTR_LEAVES 1
  number of leaves counter
- #define CTR_ROOTS 2
  number of roots counter

- #define NUM_ECTR 3
  number of graph structure counters
- #define CTR_SNNZJ 0
  number of nonzero entries of the Jacobian
- #define CTR_SNNZH 1
  number of nonzero entries of the Hessian of the Lagrangian
- #define CTR_SDIFF 2
  differentiability of the problem
- #define NUM_SCTR 3
  number of structure counters
- #define CTR_EXPRINFO_USERDEFINED (EXPRINFO_NUMOFPREDEF+1)
  number of nodes counter

type for structure info summaries

- enum coco::func_type { coco::func_const = 0, coco::func_lin = 1, coco::func_quad = 2, coco::func_poly = 3, coco::func_nonpoly = 4 }
- enum coco::fset_type { coco::fset_unbd = 0, coco::fset_box = 1, coco::fset_lin = 2, coco::fset_iquad = 3, coco::fset_equad = 4, coco::fset_ipoly = 5, coco::fset_epoly = 6, coco::fset_inonpoly = 7, coco::fset_enonpoly = 8 }
- typedef enum coco::func_type coco::func_type_t
- typedef enum coco::fset_type coco::fset_type_t

11.198.1 Detailed Description

Definition in file structure_defs.h.

11.198.2 Define Documentation

11.198.2.1 #define CTR_CEQU 1

Definition at line 135 of file structure_defs.h.

11.198.2.2 #define CTR_CINEQ 2

Definition at line 137 of file structure_defs.h.

11.198.2.3 #define CTR_CONSTR 0

Definition at line 133 of file structure_defs.h.

11.198.2.4 #define CTR_EBOUND 3

Definition at line 139 of file structure_defs.h.

11.198.2.5 #define CTR_EDEG1 4

Definition at line 141 of file structure_defs.h.

11.198.2.6 #define CTR_EDEG2 5

Definition at line 143 of file structure_defs.h.

11.198.2.7 #define CTR_EDEG3 6

Definition at line 145 of file structure_defs.h.

11.198.2.8 #define CTR_EDGES 0

Definition at line 202 of file structure_defs.h.

11.198.2.9 #define CTR_EDIM1 9

Definition at line 151 of file structure_defs.h.

11.198.2.10 #define CTR_EDIM2 10

Definition at line 153 of file structure_defs.h.

11.198.2.11 #define CTR_EINTEGER 12

Definition at line 157 of file structure_defs.h.

11.198.2.12 #define CTR_EMTDEG3P 7

Definition at line 147 of file structure_defs.h.

11.198.2.13 #define CTR_EMTDIM2 11

Definition at line 155 of file structure_defs.h.

11.198.2.14 #define CTR_ENLN 8

Definition at line 149 of file structure_defs.h.

11.198.2.15 #define CTR_EXISTS 4

Definition at line 56 of file structure_defs.h.

11.198.2.16 #define CTR_EXPRINFO_USERDEFINED (EXPRINFO_NUMOFPREDEF+1)

Definition at line 219 of file structure_defs.h.

11.198.2.17 #define CTR_FORALL 5

Definition at line 58 of file structure_defs.h.

11.198.2.18 #define CTR_FREE 3

Definition at line 54 of file structure_defs.h.

11.198.2.19 #define CTR_IACTIVE 25

Definition at line 183 of file structure_defs.h.

11.198.2.20 #define CTR_IACTIVE_HI 29

Definition at line 191 of file structure_defs.h.

11.198.2.21 #define CTR_IACTIVE_LO 27

Definition at line 187 of file structure_defs.h.

11.198.2.22 #define CTR_IBOUND 13

Definition at line 159 of file structure_defs.h.

11.198.2.23 #define CTR_ICONCAVE 24

Definition at line 181 of file structure_defs.h.

11.198.2.24 #define CTR_ICONVEX 23

Definition at line 179 of file structure_defs.h.

11.198.2.25 #define CTR_IDEG1 14

Definition at line 161 of file structure_defs.h.

11.198.2.26 #define CTR_IDEG2 15

Definition at line 163 of file structure_defs.h.

11.198.2.27 #define CTR_IDEG3 16

Definition at line 165 of file structure_defs.h.

11.198.2.28 #define CTR_IDIM1 19

Definition at line 171 of file structure_defs.h.

11.198.2.29 #define CTR_IDIM2 20

Definition at line 173 of file structure_defs.h.

11.198.2.30 #define CTR_IINACTIVE 26

Definition at line 185 of file structure_defs.h.

11.198.2.31 #define CTR_IINACTIVE_HI 30

Definition at line 193 of file structure_defs.h.

11.198.2.32 #define CTR_IINACTIVE_LO 28

Definition at line 189 of file structure_defs.h.

11.198.2.33 #define CTR_IINTEGER 22

Definition at line 177 of file structure_defs.h.

11.198.2.34 #define CTR_IMTDEG3P 17

Definition at line 167 of file structure_defs.h.

11.198.2.35 #define CTR_IMTDIM2 21

Definition at line 175 of file structure_defs.h.

11.198.2.36 #define CTR_INLN 18

Definition at line 169 of file structure_defs.h.

11.198.2.37 #define CTR_INTEGER 1

Definition at line 50 of file structure_defs.h.

11.198.2.38 #define CTR_IREDUNDANT 31

Definition at line 195 of file structure_defs.h.

11.198.2.39 #define CTR_IRESTRICT 32

Definition at line 197 of file structure_defs.h.

11.198.2.40 #define CTR_KJNODES 2

Definition at line 52 of file structure_defs.h.

11.198.2.41 #define CTR_LEAVES 1

Definition at line 204 of file structure_defs.h.

11.198.2.42 #define CTR_MIN 19

Definition at line 128 of file structure_defs.h.

11.198.2.43 #define CTR_NODES 0

Definition at line 48 of file structure_defs.h.

11.198.2.44 #define CTR_O1BOUNDED 7

Definition at line 104 of file structure_defs.h.

11.198.2.45 #define CTR_O2BOUNDED 6

Definition at line 102 of file structure_defs.h.

11.198.2.46 #define CTR_OBJ 0

Definition at line 90 of file structure_defs.h.

11.198.2.47 #define CTR_OCONCAVE 18

Definition at line 126 of file structure_defs.h.

11.198.2.48 #define CTR_OCONVEX 17

Definition at line 124 of file structure_defs.h.

11.198.2.49 #define CTR_ODEG1 12

Definition at line 114 of file structure_defs.h.

11.198.2.50 #define CTR_ODEG2 13

Definition at line 116 of file structure_defs.h.

11.198.2.51 #define CTR_ODEG3 14

Definition at line 118 of file structure_defs.h.

11.198.2.52 #define CTR_ODIM1 9

Definition at line 108 of file structure_defs.h.

11.198.2.53 #define CTR_ODIM2 10

Definition at line 110 of file structure_defs.h.

11.198.2.54 #define CTR_OEXISTS 3

Definition at line 96 of file structure_defs.h.

11.198.2.55 #define CTR_OFORALL 4

Definition at line 98 of file structure_defs.h.

11.198.2.56 #define CTR_OFREE 2

Definition at line 94 of file structure_defs.h.

11.198.2.57 #define CTR_OINTEGER 1

Definition at line 92 of file structure_defs.h.

11.198.2.58 #define CTR_OMTDEG3P 15

Definition at line 120 of file structure_defs.h.

11.198.2.59 #define CTR_OMTDIM2 11

Definition at line 112 of file structure_defs.h.

11.198.2.60 #define CTR_ONLN 16

Definition at line 122 of file structure_defs.h.

11.198.2.61 #define CTR_OSTOCHASTIC 5

Definition at line 100 of file structure_defs.h.

11.198.2.62 #define CTR_OUBOUNDED 8

Definition at line 106 of file structure_defs.h.

11.198.2.63 #define CTR_ROOTS 2

Definition at line 206 of file structure_defs.h.

11.198.2.64 #define CTR_SDIFF 2

Definition at line 215 of file structure_defs.h.

11.198.2.65 #define CTR_SNNZH 1

Definition at line 213 of file structure_defs.h.

11.198.2.66 #define CTR_SNNZJ 0

Definition at line 211 of file structure_defs.h.

11.198.2.67 #define CTR_STOCHASTIC 6

Definition at line 60 of file structure_defs.h.

11.198.2.68 #define CTR_V1BOUNDED 9

Definition at line 83 of file structure_defs.h.

11.198.2.69 #define CTR_V2BOUNDED 8

Definition at line 81 of file structure_defs.h.

11.198.2.70 #define CTR_VARS 0

Definition at line 65 of file structure_defs.h.

11.198.2.71 #define CTR_VBINARY 1

Definition at line 67 of file structure_defs.h.

11.198.2.72 #define CTR_VEXISTS 4

Definition at line 73 of file structure_defs.h.

11.198.2.73 #define CTR_VFORALL 5

Definition at line 75 of file structure_defs.h.

11.198.2.74 #define CTR_VFREE 3

Definition at line 71 of file structure_defs.h.

11.198.2.75 #define CTR_VINTEGER 1

Definition at line 69 of file structure_defs.h.

11.198.2.76 #define CTR_VKJ 7

Definition at line 79 of file structure_defs.h.

11.198.2.77 #define CTR_VSTOCHASTIC 6

Definition at line 77 of file structure_defs.h.

11.198.2.78 #define CTR_VUBOUNDED 10

Definition at line 85 of file structure_defs.h.

11.198.2.79 #define NUM_CCTR 33

Definition at line 199 of file structure_defs.h.

11.198.2.80 #define NUM_CTR 7

Definition at line 62 of file structure_defs.h.

11.198.2.81 #define NUM_ECTR 3

Definition at line 208 of file structure_defs.h.

11.198.2.82 #define NUM_OCTR 20

Definition at line 130 of file structure_defs.h.

11.198.2.83 #define NUM_SCTR 3

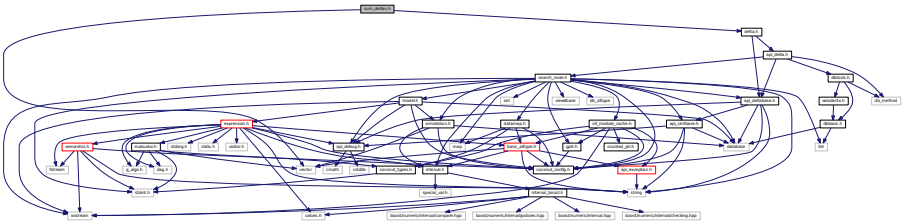
Definition at line 217 of file structure_defs.h.

11.198.2.84 #define NUM_VCTR 11

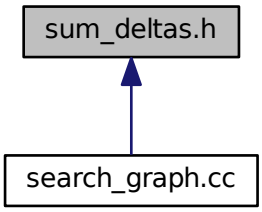
Definition at line 87 of file structure_defs.h.

11.199 sum_deltas.h File Reference

`#include <vector>` `#include "delta.h"` Include dependency graph for sum_deltas.h:



This graph shows which files directly or indirectly include this file:



Classes

- class sum_deltas

  Pre-post visitor for summing up all the deltas during work node extraction.

Functions

- work_node full_node_to_work_node (full_node &n_full, gptr< search_node > &ground)

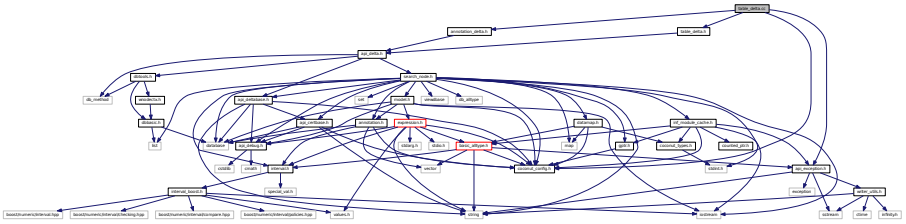
  This function converts a full_node to a work_node.

11.199.1 Detailed Description

Definition in file sum_deltas.h.

11.200 table_delta.cc File Reference

`#include <coconut_config.h>` `#include <table_delta.h>` `#include <annotation_delta.h>` `#include <api_exception.h>` Include dependency graph for table_delta.cc:



Namespaces

- namespace coco

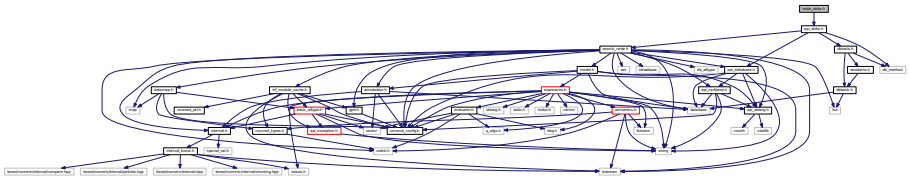
  the main namespace of the COCONUT API

11.200.1 Detailed Description

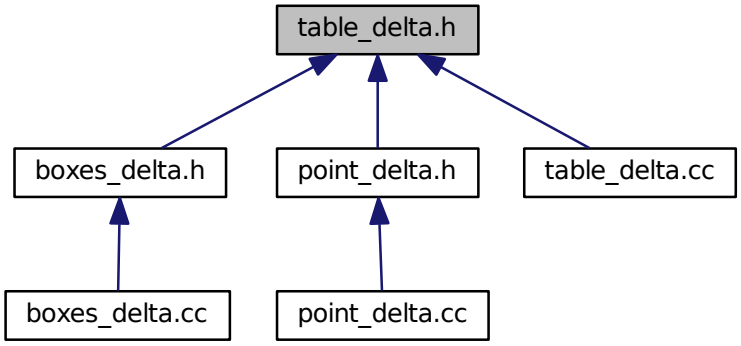
Definition in file table_delta.cc.

11.201 table_delta.h File Reference

`#include <api_delta.h>` Include dependency graph for table_delta.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::table_delta

  The base class for all deltas adding information to the search database.

Namespaces

- namespace coco

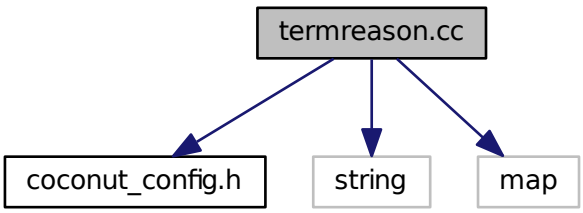
  the main namespace of the COCONUT API

11.201.1 Detailed Description

Definition in file table_delta.h.

11.202 termreason.cc File Reference

`#include <coconut_config.h>` `#include <string>` `#include <map>` Include dependency graph for termreason.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- void coco::set_translation_map (const std::map< std::string, std::string > &tr_msg_map)

  A datamap function for setting the message translation map.
- const std::string & coco::get_translated_message (const std::string &msg)

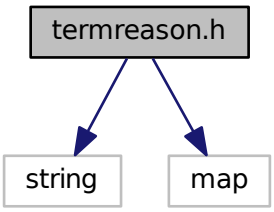
  A datamap function for message translation.

11.202.1 Detailed Description

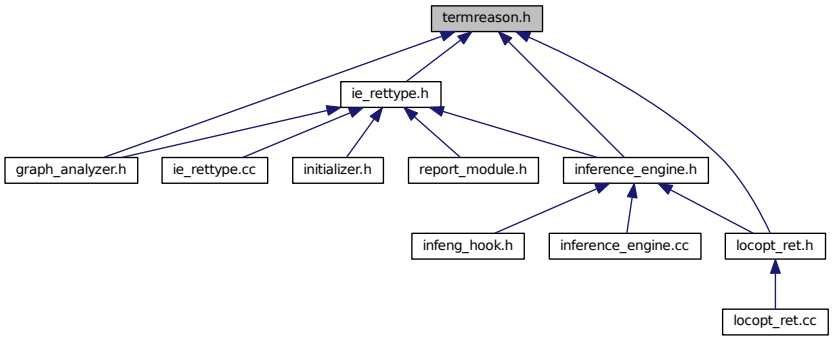
Definition in file termreason.cc.

11.203 termreason.h File Reference

`#include <string>` `#include <map>` Include dependency graph for termreason.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::termination_reason

  This class holds the reason of termination of inference and management modules.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- const std::string & coco::get_translated_message (const std::string &msg)

  A datamap function for message translation.
- void coco::set_translation_map (const std::map< std::string, std::string > &tr_msg_map)

  A datamap function for setting the message translation map.
- std::ostream & coco::operator<< (std::ostream &o, const termination_reason &__x)

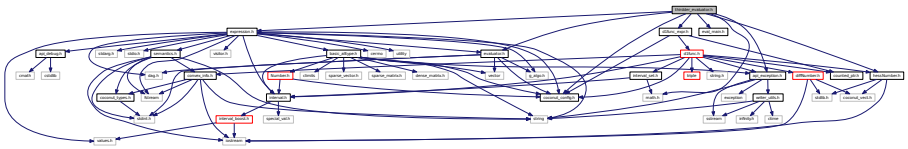
  C++ stream output operator for the termination_reason.

11.203.1 Detailed Description

Definition in file termreason.h.

11.204 thirdder_evaluator.h File Reference

`#include <coconut_config.h>` `#include <evaluator.h>` `#include <expression.-h>` `#include <eval_main.h>` `#include <math.h>` `#include <api_exception.h>` × `#include <d1func_expr.h>` `#include <hessNumber.h>` Include dependency graph for thirdder-_evaluator.h:



Classes

- struct coco::thirdderPreparationEvaluatorType
- class coco::thirdderPreparationEvaluator
- struct coco::thirdderForwardEvaluatorReturnValue
- struct coco::thirdderForwardEvaluatorType
- class coco::thirdderForwardEvaluator
- struct coco::thirdderBackwardEvaluatorType
- class coco::thirdderBackwardEvaluator

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define DEBUG_THIRDDER_EVALUATOR 0
- #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

11.204.1 Detailed Description

Definition in file thirdder_evaluator.h.

11.204.2 Define Documentation

11.204.2.1 #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

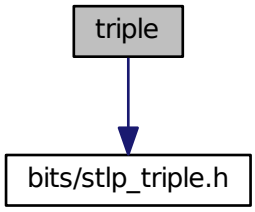
Definition at line 31 of file thirdder_evaluator.h.

11.204.2.2 #define DEBUG_THIRDDER_EVALUATOR 0

Definition at line 30 of file thirdder_evaluator.h.

11.205 triple File Reference

`#include <bits/stlp_triple.h>` Include dependency graph for triple:



This graph shows which files directly or indirectly include this file:

Defines

- #define __TRIPLE_H

11.205.1 Detailed Description

This is the external header file intended for direct use.

Definition in file triple.

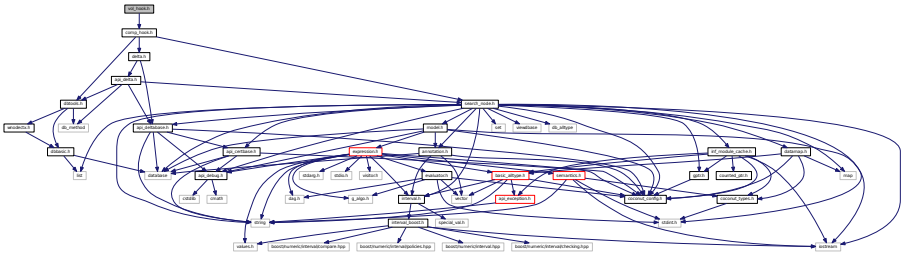
11.205.2 Define Documentation

11.205.2.1 #define __TRIPLE_H

Definition at line 30 of file triple.

11.206 vol_hook.h File Reference

`#include <comp_hook.h>` Include dependency graph for vol_hook.h:



Classes

- class coco::vol_comp_hook

  The volume computation hook (work node computation hook)

Namespaces

- namespace coco

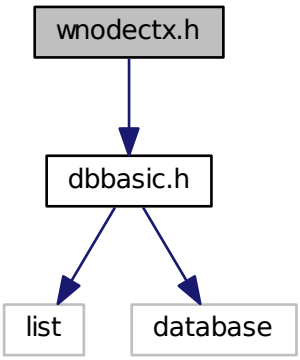
  the main namespace of the COCONUT API

11.206.1 Detailed Description

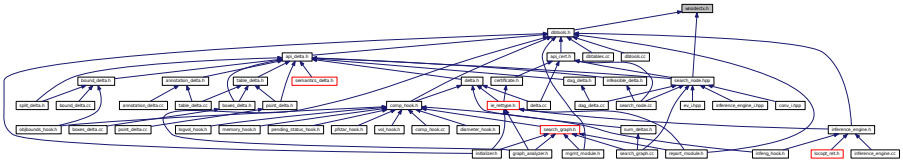
Definition in file vol_hook.h.

11.207 wnodectx.h File Reference

`#include <dbbasic.h>` Include dependency graph for wnodectx.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::work_node_context

  The evaluation context when retrieving from the search database.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

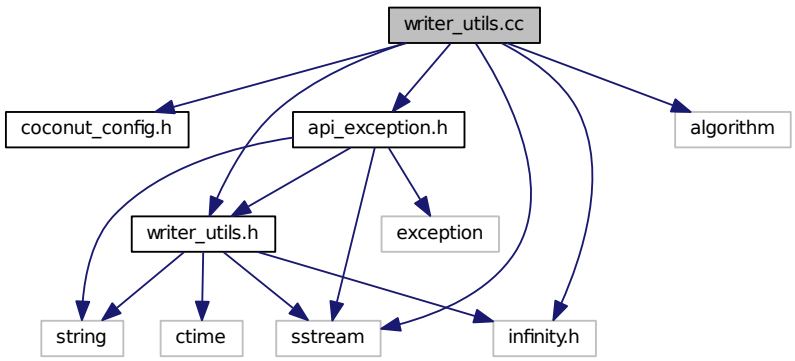
11.207.1 Detailed Description

Definition in file wnodectx.h.

11.208 writer_utils.cc File Reference

```
#include <coconut_config.h> #include <writer_utils.h> #include <sstream> ×
#include <algorithm> #include <api_exception.h> #include <infinity.h> ×
```

Include dependency graph for writer_utils.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Functions

- std::string coco::i2a (int x)

  converter int -> string
- std::string coco::datetime (const struct tm *timeptr)

  local time as C++ string
- std::string coco::localdatetime ()

  converter date & time into std::string
- std::string coco::e2D (double d)

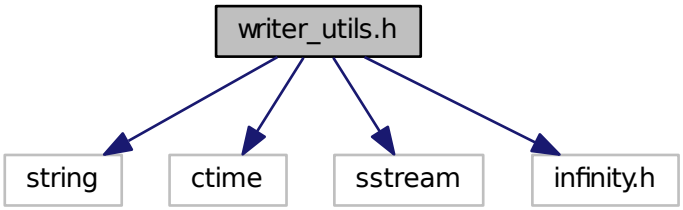
  Fortran real as C++ string.

11.208.1 Detailed Description

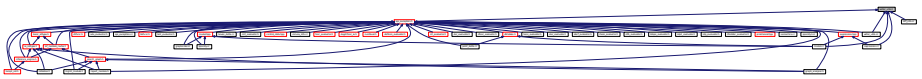
Definition in file writer_utils.cc.

11.209 writer_utils.h File Reference

`#include <string>` `#include <ctime>` `#include <sstream>` `#include <infinity.h>` Include dependency graph for writer_utils.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define double_out(d) (d)
- #define coconut_sscanfdouble(D, S, F,...) sscanf(S, F " %lf", ## __VA_ARGS__, D)
- #define double_out(d) (d)
- #define coconut_sscanfdouble(D, S, F,...) sscanf(S, F " %lf", ## __VA_ARGS__, D)
- #define double_out(d) (d)
- #define coconut_sscanfdouble(D, S, F,...) sscanf(S, F " %lf", ## __VA_ARGS__, D)
- #define double_out(d) (d)
- #define coconut_sscanfdouble(D, S, F,...) sscanf(S, F " %lf", ## __VA_ARGS__, D)
- #define double_out(d) (d)
- #define coconut_sscanfdouble(D, S, F,...) sscanf(S, F " %lf", ## __VA_ARGS__, D)
- #define double_out(d) (d)
- #define coconut_sscanfdouble(D, S, F,...) sscanf(S, F " %lf", ## __VA_ARGS__, D)
- #define double_out(d) (d)
- #define coconut_sscanfdouble(D, S, F,...) sscanf(S, F " %lf", ## __VA_ARGS__, D)

Functions

- template<class _TH >
  std::string coco::coco::convert_to_str (const _TH &_h)

  Convert an object of any printable class to a string.
- std::string coco::i2a (int x)

  converter int -> string
- std::string coco::localdatetime ()

  converter date & time into std::string
- std::string coco::datetime (const struct tm *timeptr)

  local time as C++ string
- std::string coco::e2D (double d)

  Fortran real as C++ string.

11.209.1 Detailed Description

Definition in file writer_utils.h.

11.209.2 Define Documentation

11.209.2.1 #define coconut_sscanfdouble(*D, S, F, ...*) sscanf(S, F " %lf", ## __VA_ARGS__, D)

11.209.2.2 #define coconut_sscanfdouble(*D, S, F, ...*) sscanf(S, F " %lf", ## __VA_ARGS__, D)

11.209.2.3 #define coconut_sscanfdouble(*D, S, F, ...*) sscanf(S, F " %lf", ## __VA_ARGS__, D)

11.209.2.4 #define coconut_sscanfdouble(*D, S, F, ...*) sscanf(S, F " %lf", ## __VA_ARGS__, D)

11.209.2.5 #define coconut_sscanfdouble(*D, S, F, ...*) sscanf(S, F " %lf", ## __VA_ARGS__, D)

11.209.2.6 #define coconut_sscanfdouble(*D, S, F, ...*) sscanf(S, F " %lf", ## __VA_ARGS__, D)

11.209.2.7 #define coconut_sscanfdouble(*D, S, F, ...*) sscanf(S, F " %lf", ## __VA_ARGS__, D)

Definition at line 86 of file writer_utils.h.

11.209.2.8 #define double_out(*d*) (d)

11.209.2.9 #define double_out(*d*) (d)

11.209.2.10 #define double_out(*d*) (d)

11.209.2.11 #define double_out(*d*) (d)

Definition at line 85 of file writer_utils.h.

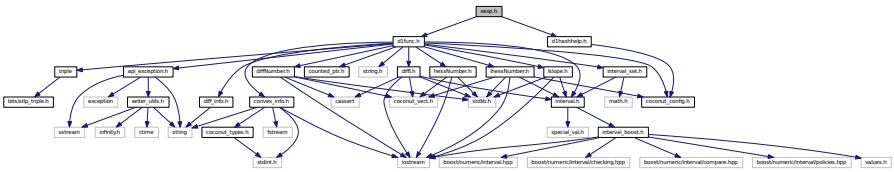
11.209.2.12 #define double_out(*d*) (d)

11.209.2.13 #define double_out(*d*) (d)

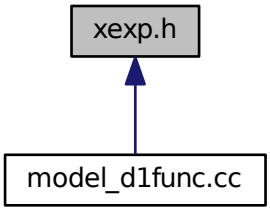
11.209.2.14 #define double_out(*d*) (d)

11.210 xexp.h File Reference

`#include <d1func.h>` `#include <d1hashhelp.h>` Include dependency graph for xexp.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::xexp_func
  The xexp_func class (one dimensional function)

Namespaces

- namespace coco
  the main namespace of the COCONUT API

11.210.1 Detailed Description

Definition in file xexp.h.