The COCONUT API

Version 3.0067

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 2005

Contents

1 Introduction 1

2 Bug List 2

3 Module Index 3

4 Namespace Index 3

5 Class Index 3

6 Class Index 15

7 File Index 22

8 Module Documentation 28

9 Namespace Documentation 85

10 Class Documentation 170

11 File Documentation 1315

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 matrix template library (`MTL`) 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::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.

Member coco::c_maybe It should be possible to specify `c_not` explicitly to identify expressions proven to be `not` convex(concave) and distinguish that from `c_maybe`.

3 Module Index

3.1 Modules

Here is a list of all modules:

Classes and types for external use [external]
Generic algorithms for external use [external]
Classes and types for internal use [external]
Generic algorithms for internal use [external]

Classes and types for basic data management 28

Basic API utilities 40

Models 45

Search Graph 58

Deltas 61

Evaluators 63

Module Base Classes 68

Database Interface 73

Advanced Utilities 74

4 Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

coco (Main namespace of the COCONUT API) 85

coco_api_internal 145

linalg (The namespace for all linear algebra types and functions) 145

std (The standard namespace) 147

5 Class Index

5.1 Class Hierarchy

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

linalg::__linalg_cvec< _TS > 170

linalg::__linalg_vec< _TS > 176
_DG_base< _Tp, _Ctr, _Iterator, _CIterator, _Alloc >[external]
_DG_base< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::const_iterator, _Alloc >[external]
__DG< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator, _SequenceCtr< void *, _PtrAlloc >::const_iterator, _SequenceCtr< void *, _PtrAlloc >::iterator, _Alloc >[external]
dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >[external]
dag< expression_node >[external]

coco::model 918
dag< search_node * >[external]

coco::search_graph 1134

coco::_evaluator_base< _Tp, _NData, _Result, _Walker > 185

coco::_evaluator_base< _Tp, _NData, _Result, _Walker > 185

coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker > 287

coco::evaluator_base< _Tp, _NData, _Result, _Walker > 522

coco::_evaluator_base< _Tp, _NData, _Result,_Walker > 185

coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker > 287

coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker > 277

coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker > 296

coco::_evaluator_base< _Tp,_NData,_Result,_Walker > 185

coco::evaluator_base< _Tp, _NData, _Result, _Walker > 522

coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker > 231

coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker > 542

coco::_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker > 185

coco::cached_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker > 287

coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker > 296

coco::analyticd_eval 198

coco::_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker > 185

coco::cached_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker > 287

coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker > 296

coco::b_interval_eval 221

coco::_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker > 185

coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker > 287

coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker > 296

coco::cinterval_eval 317

coco::_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker > 287

coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker > 277

coco::der_eval 470

coco::der_eval 470

coco::_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker > 185

coco::cached_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker > 287

coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker > 296

coco::dfunc_eval< _T, DN > 483

coco::_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_walker > 185

coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_walker > 287

coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker > 296

coco::diffI_eval 500

coco::_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker > 185

coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker > 287

coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker > 296

coco::diffNumber_eval 512

coco::_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker > 185

coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker > 287

coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker > 296

coco::func_d_eval 560

coco::func_d_eval 560

coco::_evaluator_base< func_eval_type, expression_node, double,expression_const_walker > 185

coco::cached_evaluator_base< func_eval_type, expression_node, double,expression_const_walker > 287

coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker > 296

coco::func_eval 574

coco::func_eval 574

coco::_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker > 185

coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker > 287

coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker > 296

coco::func_id_eval 587

coco::_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker > 185

coco::cached_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker > 287

coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker > 296

coco::func_islp_eval 597

coco::_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker > 287

coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker > 277

coco::hessBackwardEvaluator 619

coco::_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker > 185

coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker > 287

coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker > 296

coco::hessForwardEvaluator 631

coco::_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker > 287

coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker > 296

coco::hessPreparationEvaluator 642

coco::_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker > 287

coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker > 277

coco::ider_eval 651

coco::_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker > 185

coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker > 287

coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker > 296

coco::iderf_eval 661

coco::_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker > 287

coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker > 277

coco::ihessBackwardEvaluator 692

coco::_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker > 185

coco::cached_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker > 287

coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker > 296

coco::ihessForwardEvaluator 703

coco::_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker > 287

coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker > 296

coco::ihessPreparationEvaluator 715

coco::_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker > 185

coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker > 287

coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker > 296

coco::infbound_eval 723

coco::_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker > 185

coco::cached_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker > 287

coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker > 296

coco::interval_eval 854

coco::_evaluator_base< Islope_eval_type, expression_node, Islope,expression_const_walker > 185

coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_const_walker > 287

coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker > 296

coco::Islope_eval 876

coco::_evaluator_base< islp_eval_type, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< islp_eval_type, expression_node, bool,expression_const_walker > 287

coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker > 277

coco::islp_eval 886

coco::_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker > 185

coco::cached_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker > 287

coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker > 277

coco::model::model::lincoeff_visitor 981

coco::_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker > 185

coco::cached_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker > 287

coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker > 296

coco::model::model::detect_0chain_visitor 971

coco::_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker > 287

coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker > 296

coco::prep_d_eval 1050

coco::prep_d_eval 1050

coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker > 185

coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker > 287

coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker > 296

coco::prep_id_eval 1061

coco::prep_islp_eval 1069

coco::analyticd_eval_type 206

coco::annotation 208

coco::api_exception 219

- **coco::graph_analyzer_exception** 614
- **coco::inference_engine_exception** 749
- **coco::initializer_exception** 795
- **coco::management_module_exception** 912
- **coco::nyi_exception** 1027
- **coco::report_module_exception** 1128

coco::b_interval_eval_type 229

coco::basic_alltype 240

coco::box_check_intersection 264

linalg::c_matrix< _Tp > 274

coco::calc_pf_star 307

coco::certificate 309

coco::certificate_base 312

- **coco::compound_certificate** 329
- **coco::no_certificate** 1023
- **coco::rigorous_module_certificate** 1130
- **coco::split_certificate** 1215

coco::checking_my< T > 315

coco::cinterval_eval_type 325

coco::coconut_random_f 327

coco::convex_e 384

counted_ptr< _Tp > 389

coco::datamap 404

- **coco::control_data** 333
- **coco::info_contents** 752

coco::dbccmp_false< _TR > 441

coco::dbccmp_true< _TR > 442

coco::dbccmps_absgt< _TR > 443

coco::dbccmps_abslt< _TR > 444

coco::dbccmps_false< _TR > 445

coco::dbccmps_gt< _TR > 446

coco::dbccmps_lt< _TR > 447

coco::dbccmps_true< _TR > 448

coco::dbt_row 449

coco::delta 452

coco::delta_base 456

- **coco::annotation_delta** 209
- **coco::bound_delta** 256
- **coco::dag_delta** 392
- **coco::infeasible_delta** 733
- **coco::semantics_delta** 1195
- **coco::split_delta** 1219
- **coco::table_delta** 1241
  - **coco::boxes_delta** 267
  - **coco::point_delta** 1043

coco::delta_get_action 459

coco::der_cache 469

coco::der_eval_type 481

coco::dfunc_eval_rettype< _T, DN > 491

coco::dfunc_eval_type< _T, DN > 491

diffI 497

coco::diffI_eval_type 508

diffNumber 510

coco::diffNumber_eval_type 520

coco::expression_node 526

coco::expression_node::expression_node::children_compare 533

coco::expression_node::expression_node::parents_compare 534

coco::expression_node::expression_node::parents_compare_eq 534

FlexLexer 538

yyFlexLexer 1305

coco::func_cache 559

coco::func_d_eval_type 571

coco::func_eval_type 585

coco::func_id_eval_type 595

coco::func_islp_eval_type 605

coco::func_islp_return_type 608

gptr< _Tp > 609

gptr< _Tp > 609

ptr< _Tp > 1109

coco::graph_analyzer 610

coco::hessBackwardEvaluatorType 628

coco::hessForwardEvaluatorReturnValue 639

coco::hessForwardEvaluatorType 640

coco::hessPreparationEvaluatorType 650

coco::ider_eval_type 659

coco::iderf_eval_type 669

coco::iderf_ret_type 671

coco::ie_return_type 673

coco::ihessBackwardEvaluatorType 700

coco::ihessForwardEvaluatorReturnValue 711

coco::ihessForwardEvaluatorType 712

coco::ihessPreparationEvaluatorType 723

coco::infbound_eval_type 731

coco::inference_engine 740

coco::initializer 789

coco::interval 798

coco::interval_eval_type 861

coco::interval_set 864

coco::interval_st 871

Islope 873

coco::Islope_eval_type 884

coco::islp_eval_type 894

coco::locopt_ret_record 895

coco::management_module 901

linalg::matrix< _Tp > 914

coco::model::model::__docompare_nodes 970

coco::model::model::__docompare_variables 970

coco::model::model::detect_0chain_visitor_st 980

coco::model::model::lincoeff_visitor_ret 990

coco::model::model::lincoeff_visitor_st 991

coco::model::model::sort_constraints 1000

coco::model_gid 1001

coco::model_iddata 1014

mtl::compressed1D 1022

mtl::matrix::type 1023

coco::my_rounded_math< T > 1023

coco::point_check_feasibility 1040
postorder_visitor< expression_node, const simplify_visitor_m &, const simplify_visitor_m & >[external]

coco::model::model::simplify_visitor_m 997
postorder_visitor< expression_node, simplify_visitor_0, const simplify_visitor_0 & >[external]

coco::model::model::simplify_visitor_0 994
postorder_visitor< expression_node, unsigned int, unsigned int >[external]

coco::graphorder_visitor 616
preorder_visitor< expression_node, const variable_indicator &, const variable_indicator & >[external]

coco::sparsity_visitor 1213
preorder_visitor< expression_node, int > [external]

coco::expression_print_visitor 535
prepost_visitor< search_node *, bool > [external]

coco::__sg_anc_visitor 182
prepost_visitor< search_node *, search_node * > [external]

coco::sum_deltas 1236

sum_deltas 1231

coco::proj_rational 1077

coco::projective_interval< I > 1082

coco::report_module 1112

coco::search_graph_context 1176

coco::search_node 1179

coco::delta_node 462

coco::full_node 550

coco::work_node 1265

coco::semantics 1186

linalg::sparse_vector< _Tp > 1209

coco::statistic_info 1229

coco::termination_reason 1247

std::triple< _T1, _T2, _T3 > 1249

coco::undelta 1252

coco::undelta_base 1254

coco::annotation_undelta 215

coco::bound_undelta 260

coco::dag_undelta 399

coco::infeasible_undelta 737

coco::semantics_undelta 1206

coco::split_undelta 1227

coco::variable_indicator 1257

coco::wnc_hook_base 1261

coco::diameter_comp_hook 493

coco::inference_engine_comp_hook 745

coco::logvol_comp_hook 898

coco::objbounds_comp_hook 1029

coco::pending_status_comp_hook 1033

coco::pfstar_hook 1036

coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI > 1291

coco::work_node_comp_hook 1296

coco::work_node_context 1299

yy_buffer_state 1302

yy_trans_info 1304

6 Class Index

6.1 Class List

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

linalg::__linalg_cvec< _TS > (Adapter class for a const std::vector to a linear algebra vector) 170

linalg::__linalg_vec< _TS > (Adapter class for a std::vector to a linear algebra vector) 176

coco::__sg_anc_visitor 182

coco::_evaluator_base< _Tp, _NData, _Result, _Walker > (Base class of all evaluators) 185

coco::analyticd_eval 198

coco::analyticd_eval_type 206

coco::annotation (Annotations for Models) 208

coco::annotation_delta (Delta class for annotation changes) 209

coco::annotation_undelta (Undelta class for annotation changes) 215

coco::api_exception (API exception class) 219

coco::b_interval_eval 221

coco::b_interval_eval_type 229

coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker > (Base class of all (non-caching) backward evaluators) 231

coco::basic_alltype (The basic alltype which can hold any of a number of basic types) 240

coco::bound_delta (The bound delta class for changing the node bounds within a model) 256

coco::bound_undelta (The bound undelta class for undoing changes to the node bounds in a model) 260

coco::box_check_intersection (Stored procedure checking whether a box intersects the work_node's box) 264

coco::boxes_delta (A delta class which adds new boxes to the search database) 267

linalg::c_matrix< _Tp > (Sparse Matrix Template Class (column major)) 274

coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker > (Base class of all caching backward evaluators) 277

coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker > (Base class of all caching evaluators) 287

coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker > (Base class of all (non-caching) forward evaluators) 296

coco::calc_pf_star (Stored procedure calculating the pf$*$ value of a box) 307

coco::certificate ((certifies deltas for rigorous mode operation)) 309

coco::certificate_base (Base class for the certificates) 312

coco::checking_my< T > 315

coco::cinterval_eval 317

coco::cinterval_eval_type 325

coco::coconut_random_f 327

coco::compound_certificate (The certificate for deltas formed by compressing bound_delta entries) 329

coco::control_data (The class for communicating parameter information to COCONUT modules) 333

coco::convex_e (Convexity information) 384

counted_ptr< _Tp > 389

coco::dag_delta (The DAG delta class for performing changes to the DAG of a model) 392

coco::dag_undelta (The DAG undelta class for undoing changes to the DAG of a model) 399

coco::datamap (The base class for communicating with COCONUT modules) 404

coco::dbccmp_false< _TR > 441

coco::dbccmp_true< _TR > 442

coco::dbccmps_absgt< _TR > 443

coco::dbccmps_abslt< _TR > 444

coco::dbccmps_false< _TR > 445

coco::dbccmps_gt< _TR > 446

coco::dbccmps_lt< _TR > 447

coco::dbccmps_true< _TR > 448

coco::dbt_row (This type is used to hold one row in some table of the search database) 449

coco::delta ((updates to work nodes)) 452

coco::delta_base (Base class for the deltas) 456

coco::delta_get_action (Stored procedure class for computing the delta action specifier) 459

coco::delta_node (Class holding the delta nodes in the search graph) 462

coco::der_cache (Cache data for der_eval) 469

coco::der_eval (Backward gradient evaluation with prepared derivative data) 470

coco::der_eval_type (Visitor data for der_eval) 481

coco::dfunc_eval< _T, DN > 483

coco::dfunc_eval_rettype< _T, DN > 491

coco::dfunc_eval_type< _T, DN > 491

coco::diameter_comp_hook (The log-volume computation hook (work node computation hook)) 493

diffI 497

coco::diffI_eval (Forward function range evaluation) 500

coco::diffI_eval_type (Visitor data for diffI_eval) 508

diffNumber 510

coco::diffNumber_eval (Forward function range evaluation) 512

coco::diffNumber_eval_type (Visitor data for diffNumber_eval) 520

coco::evaluator_base< _Tp, _NData, _Result, _Walker > (Base class of all (non-caching) evaluators) 522

coco::expression_node (The base class for a node in the expression DAGs) 526

coco::expression_node::expression_node::children_compare 533

coco::expression_node::expression_node::parents_compare 534

coco::expression_node::expression_node::parents_compare_eq 534

coco::expression_print_visitor 535

FlexLexer 538

coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker > (Base class of all (non-caching) forward evaluators) 542

coco::full_node (Class holding the full nodes in the search graph) 550

coco::func_cache (Cache data for func_eval) 559

coco::func_d_eval (Forward function evaluation with preparation of derivative data) 560

coco::func_d_eval_type (Visitor data for func_d_eval) 571

coco::func_eval (Forward function evaluation) 574

coco::func_eval_type (Visitor data for func_eval) 585

coco::func_id_eval (Forward function range evaluation with preparation of interval derivative data) 587

coco::func_id_eval_type (Visitor data for func_id_eval) 595

coco::func_islp_eval (Forward function range evaluation with preparation of first order slope data) 597

coco::func_islp_eval_type (Visitor data for func_id_eval) 605

coco::func_islp_return_type (The return type of the func_islp_eval evaluator) 608

gptr< _Tp > (Global pointer class) 609

coco::graph_analyzer (Graph analyzer base class) 610

coco::graph_analyzer_exception (Graph analyzer exception class) 614

coco::graphorder_visitor (This visitor class is used for computing a graph order) 616

coco::hessBackwardEvaluator 619

coco::hessBackwardEvaluatorType 628

coco::hessForwardEvaluator 631

coco::hessForwardEvaluatorReturnValue 639

coco::hessForwardEvaluatorType 640

coco::hessPreparationEvaluator 642

coco::hessPreparationEvaluatorType 650

coco::ider_eval (Backward interval gradient evaluation with prepared interval derivative data) 651

coco::ider_eval_type (Visitor data for ider_eval) 659

coco::iderf_eval (Forward function and derivative range evaluation) 661

coco::iderf_eval_type (Visitor data for iderf_eval) 669

coco::iderf_ret_type (Visitor data for iderf_eval return value) 671

coco::ie_return_type (The return class of all inference engines) 673

coco::ihessBackwardEvaluator 692

coco::ihessBackwardEvaluatorType 700

coco::ihessForwardEvaluator 703

coco::ihessForwardEvaluatorReturnValue 711

coco::ihessForwardEvaluatorType 712

coco::ihessPreparationEvaluator 715

coco::ihessPreparationEvaluatorType 723

coco::infbound_eval 723

coco::infbound_eval_type 731

coco::infeasible_delta (The infeasible delta class for marking a model as infeasible) 733

coco::infeasible_undelta (The infeasible undelta class for undoing changes to the feasibility of a model) 737

coco::inference_engine (Inference engine base class) 740

coco::inference_engine_comp_hook (The inference engine meta computation hook (work node computation hook)) 745

coco::inference_engine_exception (Inference engine exception class) 749

coco::info_contents (The class for returning additional information from inference modules) 752

coco::initializer (Initializer base class) 789

coco::initializer_exception (Initializer exception class) 795

coco::interval (Interval wrapper class) 798

coco::interval_eval (Forward function range evaluation) 854

coco::interval_eval_type (Visitor data for interval_eval) 861

coco::interval_set 864

coco::interval_st (Constructor-free interval) 871

Islope (Class for computing intervalslopes up to order 2) 873

coco::Islope_eval (Forward function range evaluation) 876

coco::Islope_eval_type (Visitor data for Islope_eval) 884

coco::islp_eval (Backward first order slope evaluation with prepared first order slope data) 886

coco::islp_eval_type (Visitor data for islp_eval) 894

coco::locopt_ret_record 895

coco::logvol_comp_hook (The log-volume computation hook (work node computation hook)) 898

coco::management_module (Management module base class) 901

coco::management_module_exception (Management module exception class) 912

linalg::matrix< _Tp > (Sparse Matrix Template Class (row major)) 914

coco::model ((an attributed DAG of expression nodes, lowest class in the model hierarchy)) 918

coco::model::model::__docompare_nodes 970

coco::model::model::__docompare_variables 970

coco::model::model::detect_0chain_visitor 971

coco::model::model::detect_0chain_visitor_st 980

coco::model::model::lincoeff_visitor 981

coco::model::model::lincoeff_visitor_ret 990

coco::model::model::lincoeff_visitor_st 991

coco::model::model::simplify_visitor_0 994

coco::model::model::simplify_visitor_m 997

coco::model::model::sort_constraints 1000

coco::model_gid (Model Group Data Class (middle class in the model hierarchy)) 1001

coco::model_iddata (The model id-data class (the topmost in the model class hierarchy)) 1014

mtl::compressed1D 1022

mtl::matrix::type 1023

coco::my_rounded_math< T > 1023

coco::no_certificate (The not-certified certificate) 1023

coco::nyi_exception (Not Yet Implemented exception class) 1027

coco::objbounds_comp_hook (The objective-bounds computation hook (work node computation hook)) 1029

coco::pending_status_comp_hook (The pending status computation hook (work node computation hook)) 1033

coco::pfstar_hook (The pfstar computation hook (work node computation hook)) 1036

coco::point_check_feasibility (Stored procedure checking the feasibility of a point) 1040

coco::point_delta (A delta class which adds new points to the search database) 1043

coco::prep_d_eval (Preparation Evaluator for derivatives) 1050

coco::prep_id_eval (Preparation Evaluator for interval derivatives) 1061

coco::prep_islp_eval (Preparation Evaluator for first order slopes) 1069

coco::proj_rational 1077

coco::projective_interval< I > 1082

ptr< _Tp > (Local global pointer class) 1109

coco::report_module (Report module base class) 1112

coco::report_module_exception (Report module exception class) 1128

coco::rigorous_module_certificate (The certificate for deltas computed by rigorous inference engines) 1130

coco::search_graph (The search graph) 1134

coco::search_graph_context (An evaluation context when retrieving from the search database) 1176

coco::search_node (Base type of the nodes in the search graph) 1179

coco::semantics (Expression Semantics) 1186

coco::semantics_delta (The semantics delta class for changing the node semantics within a model) 1195

coco::semantics_undelta (The semantics undelta class for undoing changes to the node semantics in a model) 1206

linalg::sparse_vector< _Tp > (Sparse Vector Template Class) 1209

coco::sparsity_visitor (Preorder visitor which calculates sparsity structures) 1213

coco::split_certificate (The certificate for deltas formed by splits) 1215

coco::split_delta (The split delta class for proposing useful splits) 1219

coco::split_undelta (The split undelta class for removing proposed splits from a work_node) 1227

coco::statistic_info (Base class for all inference engine statistics classes) 1229

sum_deltas (Pre-post visitor for summing up all the deltas during work node extraction) 1231

coco::sum_deltas (Pre-post visitor for summing up all the deltas during work node extraction) 1236

coco::table_delta (The base class for all deltas adding information to the search database) 1241

coco::termination_reason (This class holds the reason of termination of inference and management modules) 1247

std::triple< _T1, _T2, _T3 > (Triple holds three objects of arbitrary type) 1249

coco::undelta ((undo of updates to work nodes)) 1252

coco::undelta_base (Base class for the undeltas) 1254

coco::variable_indicator (Bitmap class used to indicate variable occurrence) 1257

coco::wnc_hook_base (Base class for the work node computation hooks) 1261

coco::work_node (Work node, which is passed to the inference engines) 1265

coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI > (The base class for work_node::constraint_iterator and work_node::constraint_const_iterator) 1291

coco::work_node_comp_hook ((work node computation hook)) 1296

coco::work_node_context (The evaluation context when retrieving from the search database) 1299

yy_buffer_state 1302

yy_trans_info 1304

yyFlexLexer 1305

7 File Index

7.1 File List

Here is a list of all files with brief descriptions:

add_algo 1315

addinfo.h 1315

ade_evaluator.h 1316

annotation.h 1317

annotation_delta.cc 1318

annotation_delta.h 1319

api_cert.h 1320

api_certbase.h 1321
api_debug.h 1323
api_delta.h 1325
api_deltabase.h 1326
api_exception.cc 1327
api_exception.h 1328
api_extradocu.h 1329
basic_alltype.cc 1330
basic_alltype.h 1330
bint_evaluator.h 1335
bound_delta.cc 1336
bound_delta.h 1337
boxes_delta.cc 1338
boxes_delta.h 1339
callback_ref.cc 1340
callback_ref.h 1340
certificate.h 1341
cint_evaluator.h 1341
coconut_config.h 1342
coconut_random.h 1343
coconut_types.h 1345
comp_hook.cc 1346
comp_hook.h 1346
control_data.h 1347
control_data.hpp 1348
counted_ptr.h 1349
dag_delta.cc 1350
dag_delta.h 1351
datamap.cc 1352

datamap.h 1352

datamap.hpp 1354

dbbasic.h 1355

dbcompare.cc 1355

dbcompare.h 1357

dbcompare.hpp 1361

dbcompare_expression_parser.cc 1363

dbcompare_expression_scanner.cc 1365

dbcompare_sortorder_parser.cc 1380

dbcompare_sortorder_scanner.cc 1382

dbtables.cc 1396

dbtables.h 1397

dbtools.cc 1398

dbtools.h 1398

delta.cc 1400

delta.h 1400

der_evaluator.h 1401

der_evaluator_n.h 1402

dfunc_evaluator.h 1404

diameter_hook.h 1405

diffI.h 1405

diffI_evaluator.h 1410

diffNumber.h 1411

diffNumber_evaluator.h 1416

eval_main.cc 1417

eval_main.h 1418

evaluator.h 1422

expression.cc 1424

expression.h 1425

expression.hpp 1429
exprnames.cc 1430
FlexLexer.h 1430
func_evaluator.h 1431
func_evaluator_n.h 1432
gptr.h 1433
graph_analyzer.h 1434
graphorder.cc 1435
graphorder.h 1436
hess_color.cc 1437
hess_color.h 1438
hess_evaluator.h 1439
ider_evaluator.h 1441
iderf_evaluator.h 1442
ie_rettype.cc 1443
ie_rettype.h 1444
ie_rettype.hpp 1445
ie_statistic.h 1445
ihess_evaluator.h 1446
infb_evaluator.h 1447
infeasible_delta.h 1448
infeng_hook.h 1449
inference_engine.cc 1450
inference_engine.h 1451
info_contents.h 1452
initializer.h 1453
int_evaluator.h 1454
interval.cc 1455
interval.h 1456

interval_boost.h 1457
interval_filib.h 1460
interval_profil.h 1463
interval_set.cc 1466
interval_set.h 1468
Islope.h 1470
Islope_evaluator.h 1475
islp_evaluator.h 1476
linalg.h 1478
locopt_ret.cc 1481
locopt_ret.h 1482
logvol_hook.h 1485
mgmt_module.h 1486
model.cc 1487
model.h 1489
model.hpp 1490
objbounds_hook.h 1493
pending_status_hook.h 1494
pfstar.cc 1495
pfstar.h 1495
pfstar_hook.h 1496
point_delta.cc 1497
point_delta.h 1498
print_map.h 1499
print_matrix.h 1500
print_seq.h 1501
print_set.h 1502
print_tuple.h 1502
prointerval.h 1503

prointerval.hpp 1504
report_module.h 1507
search_graph.cc 1508
search_graph.h 1510
search_graph.hpp 1511
search_graph_funcs.cc 1512
search_graph_funcs.h 1513
search_node.cc 1514
search_node.h 1515
search_node.hpp 1517
semantics.cc 1518
semantics.h 1519
semantics_delta.cc 1521
semantics_delta.h 1522
sgraphctx.h 1523
sparsity.cc 1524
sparsity.h 1526
split_delta.h 1528
stdafx.h 1529
stlp_addalgo.h 1529
stlp_pairheap.h 1534
stlp_triple.h 1536
structure_defs.h 1537
sum_deltas.h 1553
table_delta.cc 1553
table_delta.h 1554
termreason.cc 1555
termreason.h 1556
triple 1557

wnodectx.h **1558**

writer_utils.cc **1559**

writer_utils.h **1560**

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< _Tp >

  Global pointer class.

- class ptr< _Tp >

  Local global pointer class.

- class counted_ptr< _Tp >
- class linalg::matrix< _Tp >

  Sparse Matrix Template Class (row major).

- class linalg::c_matrix< _Tp >

  Sparse Matrix Template Class (column major).

- class linalg::sparse_vector< _Tp >

  Sparse Vector Template Class.

- class linalg::__linalg_cvec< _TS >

  Adapter class for a const std::vector to a linear algebra vector.

- class linalg::__linalg_vec< _TS >

  Adapter class for a std::vector to a linear algebra vector.

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)
- #define linalg_trans(A) mtl::trans((A))
- #define linalg_scale(X, Y) mtl::scaled(X, Y)

Typedefs

- typedef basic_alltype coco::additional_info_u

  type definition for backwards compatibility with earlier API versions

Enumerations

- enum linalg::linalg_output_format { linalg::linalg_output_sparse, linalg::linalg_output_full }

  The possible output formats of a matrix.

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.

- template<class _Tp >
  matrix< _Tp > linalg::operator+ (const matrix< _Tp > &A, const matrix< _Tp > &B)

  Sparse matrix addition operator.

- template<class _Tp >
  matrix< _Tp > linalg::operator* (_Tp a, const matrix< _Tp > &B)

  Sparse scalar matrix multiplication operator.

- template<class _Tp >
  c_matrix< _Tp > linalg::operator+ (const c_matrix< _Tp > &A, const c_matrix< _Tp > &B)

  Sparse matrix addition operator.

- template<class _Tp >
  c_matrix< _Tp > linalg::operator* (_Tp a, const c_matrix< _Tp > &B)

  Sparse matrix scalar multiplication operator.

- template<class _Tp >
  bool linalg::operator== (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector comparison operator.

- template<class _Tp >
  sparse_vector< _Tp > linalg::operator+ (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector addition operator.

- template<class _Tp >
  _Tp linalg::operator* (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector scalar product operator.

- template<class _Tp >
  sparse_vector< _Tp > linalg::operator* (_Tp a, const sparse_vector< _Tp > &b)

  Sparse vector scaling operator.

- template<class _Ve >
  std::vector< _Ve > & linalg::linalg_ssum (std::vector< _Ve > &a, _Ve b, const std::vector< _Ve > &c)
- template<class _Ve >
  std::vector< _Ve > & linalg::linalg_smult (std::vector< _Ve > &a, _Ve b)

- template<class _TS >
  _TS linalg::linalg_dot (const std::vector< _TS > &a, const std::vector< _TS > &b, _TS s)
- template<class _Va , class _TS >
  _TS linalg::linalg_dot (const _Va &a, const std::vector< _TS > &b, _TS s)
- template<class _Vb , class _TS >
  _TS linalg::linalg_dot (const std::vector< _TS > &a, const _Vb &b, _TS s)
- template<class _Va , class _Vb , class _TS >
  _TS linalg::linalg_dot (const _Va &a, const _Vb &b, _TS s)

- template<class _TS >
  void linalg::linalg_add (const std::vector< _TS > &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Va , class _TS >
  void linalg::linalg_add (const _Va &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Vb , class _TS >
  void linalg::linalg_add (const std::vector< _TS > &a, const _Vb &b, std::vector< _TS > &c)
- template<class _TS , class _Vc >
  void linalg::linalg_add (const std::vector< _TS > &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Va , class _Vb , class _TS >
  void linalg::linalg_add (const _Va &a, const _Vb &b, std::vector< _TS > &c)
- template<class _Va , class _TS , class _Vc >
  void linalg::linalg_add (const _Va &a, const std::vector< _TS > &b, _Vc &c)
- template<class _TS , class _Vb , class _Vc >
  void linalg::linalg_add (const std::vector< _TS > &a, const _Vb &b, _Vc &c)

- template<class _Va , class _Vb , class _Vc >
  void linalg::linalg_add (const _Va &a, const _Vb &b, _Vc &c)

- template<class _Matrix , class _TS >
  void linalg::linalg_matvec (const _Matrix &A, const std::vector< _TS > &a, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Va >
  void linalg::linalg_matvec (const _Matrix &A, const _Va &a, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Vc >
  void linalg::linalg_matvec (const _Matrix &A, const std::vector< _TS > &a, _Vc &c)
- template<class _Matrix , class _Va , class _Vc >
  void linalg::linalg_matvec (const _Matrix &A, const _Va &a, _Vc &c)

- template<class _Matrix , class _TS >
  void linalg::linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Va >
  void linalg::linalg_matvecv (const _Matrix &A, const _Va &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Vb >
  void linalg::linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const _Vb &b, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Vc >
  void linalg::linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Matrix , class _Va , class _Vb , class _TS >
  void linalg::linalg_matvecv (const _Matrix &A, _Va &a, const _Vb &b, const std::vector< _TS > &c)
- template<class _Matrix , class _Va , class _TS , class _Vc >
  void linalg::linalg_matvecv (const _Matrix &A, const _Va &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Matrix , class _TS , class _Vb , class _Vc >
  void linalg::linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const _Vb &b, _Vc &c)
- template<class _Matrix , class _Va , class _Vb , class _Vc >
  void linalg::linalg_matvecv (const _Matrix &A, const _Va &a, const _Vb &b, _Vc &c)

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 87 of file basic_alltype.h.

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 98 of file basic_alltype.h.

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 47 of file basic_alltype.h.

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 117 of file basic_alltype.h.

8.1.2.5 #define linalg_scale(X, Y) mtl::scaled(X, Y)

This function computes the vector/matrix Y scaled by X.

Definition at line 1142 of file linalg.h.

8.1.2.6 #define linalg_trans(A) mtl::trans((A))

This function computes the transposed of the matrix A.

Definition at line 1139 of file linalg.h.

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 Enumeration Type Documentation

8.1.4.1 enum linalg::linalg_output_format

This class describes the possible output formats using the <<-operator for a sparse matrix.

Enumerator:

linalg_output_sparse This is a possible value of the output_format member of the matrix class, which instructs the << operator to print the matrix in a Matlab-like sparse matrix format.

linalg_output_full This is a possible value of the output_format member of the matrix class, which instructs the << operator to print the matrix in a full rectangular matrix format including all zeros.

Definition at line 223 of file linalg.h.

8.1.5 Function Documentation

8.1.5.1 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.

8.1.5.2 template<class _Va , class _Vb , class _Vc > void linalg::linalg_add (const _Va & *a*, const _Vb & *b*, _Vc & *c*) [inline]

This function computes the vector addition c=a+b.

Definition at line 1038 of file linalg.h.

8.1.5.3 template<class _TS , class _Vb , class _Vc > void linalg::linalg_add (const std::vector< _TS > & *a*, const _Vb & *b*, _Vc & *c*) [inline]

This function computes the vector addition c=a+b.

Definition at line 1032 of file linalg.h.

8.1.5.4 template<class _Va , class _TS , class _Vc > void linalg::linalg_add (const _Va & *a*, const std::vector< _TS > & *b*, _Vc & *c*) [inline]

This function computes the vector addition c=a+b.

Definition at line 1026 of file linalg.h.

8.1.5.5 template<class _Va , class _Vb , class _TS > void linalg::linalg_add (const _Va & *a*, const _Vb & *b*, std::vector< _TS > & *c*) [inline]

This function computes the vector addition c=a+b.

Definition at line 1020 of file linalg.h.

8.1.5.6 template<class _TS , class _Vc > void linalg::linalg_add (const std::vector< _TS > & *a*, const std::vector< _TS > & *b*, _Vc & *c*) [inline]

This function computes the vector addition c=a+b.

Definition at line 1014 of file linalg.h.

8.1.5.7 template<class _Vb , class _TS > void linalg::linalg_add (const std::vector< _TS > & *a*, const _Vb & *b*, std::vector< _TS > & *c*) [inline]

This function computes the vector addition c=a+b.

Definition at line 1008 of file linalg.h.

8.1.5.8 template<class _Va , class _TS > void linalg::linalg_add (const _Va & *a*, const std::vector< _TS > & *b*, std::vector< _TS > & *c*) [inline]

This function computes the vector addition c=a+b.

Definition at line 1002 of file linalg.h.

8.1.5.9 template<class _TS > void linalg::linalg_add (const std::vector< _TS > & *a*, const std::vector< _TS > & *b*, std::vector< _TS > & *c*) [inline]

This function computes the vector addition c=a+b.

Definition at line 996 of file linalg.h.

8.1.5.10 template<class _Va , class _Vb , class _TS > _TS linalg::linalg_dot (const _Va & *a*, const _Vb & *b*, _TS *s*) [inline]

This function computes the inner product s∗a∗b.

Definition at line 986 of file linalg.h.

8.1.5.11 template<class _Vb , class _TS > _TS linalg::linalg_dot (const std::vector< _TS > & *a*, const _Vb & *b*, _TS *s*) [inline]

This function computes the inner product s∗a∗b.

Definition at line 980 of file linalg.h.

8.1.5.12 template<class _Va , class _TS > _TS linalg::linalg_dot (const _Va & *a*, const std::vector< _TS > & *b*, _TS *s*) [inline]

This function computes the inner product s∗a∗b.

Definition at line 973 of file linalg.h.

8.1.5.13 template<class _TS > _TS linalg::linalg_dot (const std::vector< _TS > & *a*, const std::vector< _TS > & *b*, _TS *s*) [inline]

This function computes the inner product s∗a∗b.

Definition at line 967 of file linalg.h.

8.1.5.14 template<class _Matrix , class _Va , class _Vc > void linalg::linalg_matvec (const _Matrix & *A*, const _Va & *a*, _Vc & *c*) [inline]

This function computes the matrix-vector product c=Aa.

Definition at line 1070 of file linalg.h.

8.1.5.15 template<class _Matrix , class _TS , class _Vc > void linalg::linalg_matvec (const _Matrix & *A*, const std::vector< _TS > & *a*, _Vc & *c*) [inline]

This function computes the matrix-vector product c=Aa.

Definition at line 1063 of file linalg.h.

8.1.5.16 template<class _Matrix , class _TS , class _Va > void linalg::linalg_matvec (const _Matrix & *A*, const _Va & *a*, std::vector< _TS > & *c*) [inline]

This function computes the matrix-vector product c=Aa.

Definition at line 1056 of file linalg.h.

8.1.5.17 template<class _Matrix , class _TS > void linalg::linalg_matvec (const _Matrix & *A*, const std::vector< _TS > & *a*, std::vector< _TS > & *c*) [inline]

This function computes the matrix-vector product c=Aa.

Definition at line 1049 of file linalg.h.

8.1.5.18 template<class _Matrix , class _Va , class _Vb , class _Vc > void linalg::linalg_matvecv (const _Matrix & *A*, const _Va & *a*, const _Vb & *b*, _Vc & *c*) [inline]

This function computes the generalized matrix-vector product c=Aa+b.

Definition at line 1130 of file linalg.h.

8.1.5.19 template<class _Matrix , class _TS , class _Vb , class _Vc > void linalg::linalg_matvecv (const _Matrix & *A*, const std::vector< _TS > & *a*, const _Vb & *b*, _Vc & *c*) [inline]

This function computes the generalized matrix-vector product c=Aa+b.

Definition at line 1123 of file linalg.h.

8.1.5.20 template<class _Matrix , class _Va , class _TS , class _Vc > void linalg::linalg_matvecv (const _Matrix & *A*, const _Va & *a*, const std::vector< _TS > & *b*, _Vc & *c*) [inline]

This function computes the generalized matrix-vector product c=Aa+b.

Definition at line 1116 of file linalg.h.

8.1.5.21 template<class _Matrix , class _Va , class _Vb , class _TS > void linalg::linalg_matvecv (const _Matrix & *A*, _Va & *a*, const _Vb & *b*, const std::vector< _TS > & *c*) [inline]

This function computes the generalized matrix-vector product c=Aa+b.

Definition at line 1109 of file linalg.h.

8.1.5.22 template<class _Matrix , class _TS , class _Vc > void linalg::linalg_matvecv (const _Matrix & *A*, const std::vector< _TS > & *a*, const std::vector< _TS > & *b*, _Vc & *c*) [inline]

This function computes the generalized matrix-vector product c=Aa+b.

Definition at line 1102 of file linalg.h.

8.1.5.23 template<class _Matrix , class _TS , class _Vb > void linalg::linalg_matvecv (const _Matrix & *A*, const std::vector< _TS > & *a*, const _Vb & *b*, std::vector< _TS > & *c*) [inline]

This function computes the generalized matrix-vector product c=Aa+b.

Definition at line 1095 of file linalg.h.

8.1.5.24 template<class _Matrix , class _TS , class _Va > void linalg::linalg_matvecv (const _Matrix & *A*, const _Va & *a*, const std::vector< _TS > & *b*, std::vector< _TS > & *c*) [inline]

This function computes the generalized matrix-vector product c=Aa+b.

Definition at line 1088 of file linalg.h.

8.1.5.25 template<class _Matrix , class _TS > void linalg::linalg_matvecv (const _Matrix & *A*, const std::vector< _TS > & *a*, const std::vector< _TS > & *b*, std::vector< _TS > & *c*) [inline]

This function computes the generalized matrix-vector product c=Aa+b.

Definition at line 1080 of file linalg.h.

8.1.5.26 template<class _Ve > std::vector<_Ve>& linalg::linalg_smult (std::vector< _Ve > & *a*, _Ve *b*) [inline]

This function computes a=b∗a and returns a.

Definition at line 1165 of file linalg.h.

8.1.5.27 template<class _Ve > std::vector<_Ve>& linalg::linalg_ssum (std::vector< _Ve > & *a*, _Ve *b*, const std::vector< _Ve > & *c*) [inline]

This function computes the generalized sum a=a+b∗c and returns a.

Definition at line 1148 of file linalg.h.

8.1.5.28 template<class _Tp > sparse_vector<_Tp> linalg::operator∗ (_Tp *a*, const sparse_vector< _Tp > & *b*) [inline]

This function computes product of a sparse vector and a scalar.

Definition at line 831 of file linalg.h.

8.1.5.29 template<class _Tp > _Tp linalg::operator∗ (const sparse_vector< _Tp > & *a*, const sparse_vector< _Tp > & *b*) [inline]

This function computes the scalar product of two sparse vectors.

Definition at line 808 of file linalg.h.

8.1.5.30 template<class _Tp > c_matrix<_Tp> linalg::operator* (_Tp *a*, const c_matrix< _Tp > & *B*) [inline]

This function multiplies a sparse matrix with a scalar, returning the result.

Definition at line 549 of file linalg.h.

8.1.5.31 template<class _Tp > matrix<_Tp> linalg::operator* (_Tp *a*, const matrix< _Tp > & *B*) [inline]

This function multiplies a sparse matrix with a scalar, returning the result.

Definition at line 375 of file linalg.h.

8.1.5.32 template<class _Tp > sparse_vector<_Tp> linalg::operator+ (const sparse_vector< _Tp > & *a*, const sparse_vector< _Tp > & *b*) [inline]

This function adds two sparse vectors, returning the result.

Definition at line 781 of file linalg.h.

8.1.5.33 template<class _Tp > c_matrix<_Tp> linalg::operator+ (const c_matrix< _Tp > & *A*, const c_matrix< _Tp > & *B*) [inline]

This function adds two sparse matrices, returning the result.

Definition at line 526 of file linalg.h.

8.1.5.34 template<class _Tp > matrix<_Tp> linalg::operator+ (const matrix< _Tp > & *A*, const matrix< _Tp > & *B*) [inline]

This function adds two sparse matrices, returning the result.

Definition at line 352 of file linalg.h.

8.1.5.35 template<class _Tp > bool linalg::operator== (const sparse_vector< _Tp > & *a*, const sparse_vector< _Tp > & *b*) [inline]

This function returns whether two sparse vectors are equal.

Definition at line 717 of file linalg.h.

8.1.5.36 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.

Functions

- template<class _TH >
  std::string 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 _Tp , int NN, int IND_OFFSET>
  ostream & std::operator<< (ostream &o, const mtl::light1D< _Tp, NN, IND_OFFSET > &a)
- template<typename _Tp >
  ostream & std::operator<< (ostream &s, const linalg::matrix< _Tp > &A)
- template<typename _Tp >
  ostream & std::operator<< (ostream &s, const linalg::c_matrix< _Tp > &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 _Tp >
  ostream & std::operator<< (ostream &s, const linalg::sparse_vector< _Tp > &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.

8.2.4 Function Documentation

8.2.4.1 template<class _TH > std::string 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 45 of file writer_utils.h.

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

This function defines the stream output operator for triples.

Definition at line 48 of file print_tuple.h.

8.2.4.3 template<typename _TA , typename _TB > ostream& std::operator<< (ostream & *o*, const pair< _TA, _TB > & *a*) [inline]

This function defines the stream output operator for pairs.

Definition at line 40 of file print_tuple.h.

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

This function defines the stream output operator for standard multisets.

Definition at line 56 of file print_set.h.

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

This function defines the stream output operator for standard sets.

Definition at line 40 of file print_set.h.

8.2.4.6 template<typename _Tp > ostream& std::operator<< (ostream & *s*, const linalg::sparse_vector< _Tp > & *a*) [inline]

This function defines the stream output operator for sparse vectors.

Definition at line 92 of file print_seq.h.

8.2.4.7 template<typename _Ve , typename _TA > ostream& std::operator<< (ostream & *o*, const deque< _Ve, _TA > & *a*) [inline]

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

Definition at line 76 of file print_seq.h.

8.2.4.8 template<typename _Ve , typename _TA > ostream& std::operator<< (ostream & *o*, const list< _Ve, _TA > & *a*) [inline]

This function defines the stream output operator for standard lists.

Definition at line 59 of file print_seq.h.

8.2.4.9 template<typename _Ve , typename _TA > ostream& std::operator<< (ostream & *o*, const vector< _Ve, _TA > & *a*) [inline]

This function defines the stream output operator for standard vectors.

Definition at line 43 of file print_seq.h.

8.2.4.10 template<typename _Tp > ostream& std::operator<< (ostream & *s*, const linalg::c_matrix< _Tp > & *A*) [inline]

This function defines the stream output operator for sparse matrices.

Definition at line 128 of file print_matrix.h.

8.2.4.11 template<typename _Tp > ostream& std::operator<< (ostream & *s*, const linalg::matrix< _Tp > & *A*) [inline]

This function defines the stream output operator for sparse matrices.

Definition at line 55 of file print_matrix.h.

8.2.4.12 template<typename _Tp , int NN, int IND_OFFSET> ostream& std::operator<< (ostream & *o*, const mtl::light1D< _Tp, NN, IND_OFFSET > & *a*) [inline]

This function defines the stream output operator for light1D vectors.

Definition at line 39 of file print_matrix.h.

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

This function defines the stream output operator for standard multimaps.

Definition at line 58 of file print_map.h.

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

This function defines the stream output operator for standard maps.

Definition at line 42 of file print_map.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.

Functions

- void coco::graphorder (const model &DAG, std::vector< unsigned int > &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_GAUSS -20
- #define EXPRINFO_POLY -21
- #define EXPRINFO_POW -22
- #define EXPRINFO_DIV -23
- #define EXPRINFO_ATAN2 -24
- #define EXPRINFO_LIN -25
- #define EXPRINFO_QUAD -26
- #define EXPRINFO_RE -27
- #define EXPRINFO_IM -28
- #define EXPRINFO_ARG -29
- #define EXPRINFO_CPLXCONJ -30
- #define EXPRINFO_LOOKUP -31
- #define EXPRINFO_PWLIN -32
- #define EXPRINFO_SPLINE -33
- #define EXPRINFO_PWCONSTLC -34
- #define EXPRINFO_PWCONSTRC -35
- #define EXPRINFO_IN -36
- #define EXPRINFO_IF -37
- #define EXPRINFO_AND -38

- #define EXPRINFO_OR -39
- #define EXPRINFO_NOT -40
- #define EXPRINFO_IMPLIES -41
- #define EXPRINFO_COUNT -42
- #define EXPRINFO_ALLDIFF -43
- #define EXPRINFO_HISTOGRAM -44
- #define EXPRINFO_LEVEL -45
- #define EXPRINFO_NEIGHBOR -46
- #define EXPRINFO_NOGOOD -47
- #define EXPRINFO_EXPECTATION -48
- #define EXPRINFO_INTEGRAL -49
- #define EXPRINFO_DET -50
- #define EXPRINFO_COND -51
- #define EXPRINFO_PSD -52
- #define EXPRINFO_MPROD -53
- #define EXPRINFO_FEM -54
- #define EXPRINFO_CMPROD -55
- #define EXPRINFO_CGFEM -56
- #define EXPRINFO_UNDEFINED -57
- #define EXPRINFO_NUMOFPREDEF -(EXPRINFO_UNDEFINED)

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 145 of file expression.h.

8.3.2.2 #define EXPRINFO_ALLDIFF -43

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 288 of file expression.h.

8.3.2.3 #define EXPRINFO_AND -38

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 265 of file expression.h.

8.3.2.4 #define EXPRINFO_ARG -29

This node returns the argument of a complex number.

Definition at line 215 of file expression.h.

8.3.2.5 #define EXPRINFO_ATAN2 -24

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 190 of file expression.h.

8.3.2.6 #define EXPRINFO_CGFEM -56

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

Definition at line 351 of file expression.h.

8.3.2.7 #define EXPRINFO_CMPROD -55

This node computes a matrix vector product with constant A.

Definition at line 347 of file expression.h.

8.3.2.8 #define EXPRINFO_COND -51

This node computes the condition of a matrix.

Definition at line 333 of file expression.h.

8.3.2.9 #define EXPRINFO_CONSTANT -1

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

Definition at line 81 of file expression.h.

8.3.2.10 #define EXPRINFO_COS -19

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

Definition at line 165 of file expression.h.

8.3.2.11 #define EXPRINFO_COUNT -42

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 283 of file expression.h.

8.3.2.12 #define EXPRINFO_CPLXCONJ -30

This node returns the complex conjugate of a complex number.

Definition at line 220 of file expression.h.

8.3.2.13 #define EXPRINFO_DET -50

This node computes the determinant of a matrix.

Definition at line 330 of file expression.h.

8.3.2.14 #define EXPRINFO_DIV -23

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

Definition at line 185 of file expression.h.

8.3.2.15 #define EXPRINFO_EXP -16

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

Definition at line 153 of file expression.h.

8.3.2.16 #define EXPRINFO_EXPECTATION -48

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

Definition at line 318 of file expression.h.

8.3.2.17 #define EXPRINFO_FEM -54

This node computes information relevant for FEM computations.

Definition at line 342 of file expression.h.

8.3.2.18 #define EXPRINFO_GAUSS -20

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 170 of file expression.h.

8.3.2.19 #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 76 of file expression.h.

8.3.2.20 #define EXPRINFO_HISTOGRAM -44

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 294 of file expression.h.

8.3.2.21 #define EXPRINFO_IF -37

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 261 of file expression.h.

8.3.2.22 #define EXPRINFO_IM -28

This node returns the imaginary part of a complex number.

Definition at line 212 of file expression.h.

8.3.2.23 #define EXPRINFO_IMPLIES -41

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 279 of file expression.h.

8.3.2.24 #define EXPRINFO_IN -36

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 250 of file expression.h.

8.3.2.25 #define EXPRINFO_INTEGRAL -49

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

Definition at line 322 of file expression.h.

8.3.2.26 #define EXPRINFO_INTPOWER -15

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

Definition at line 149 of file expression.h.

8.3.2.27 #define EXPRINFO_INVERT -11

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

Definition at line 133 of file expression.h.

8.3.2.28 #define EXPRINFO_LEVEL -45

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 300 of file expression.h.

8.3.2.29 #define EXPRINFO_LIN -25

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 197 of file expression.h.

8.3.2.30 #define EXPRINFO_LOG -17

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

Definition at line 157 of file expression.h.

8.3.2.31 #define EXPRINFO_LOOKUP -31

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

Definition at line 226 of file expression.h.

8.3.2.32 #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 109 of file expression.h.

8.3.2.33 #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 101 of file expression.h.

8.3.2.34 #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 113 of file expression.h.

8.3.2.35 #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 118 of file expression.h.

8.3.2.36 #define EXPRINFO_MPROD -53

This node computes a matrix-vector product.

Definition at line 339 of file expression.h.

8.3.2.37 #define EXPRINFO_NEIGHBOR -46

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 308 of file expression.h.

8.3.2.38 #define EXPRINFO_NOGOOD -47

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

Definition at line 312 of file expression.h.

8.3.2.39 #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 127 of file expression.h.

8.3.2.40 #define EXPRINFO_NOT -40

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

Definition at line 273 of file expression.h.

8.3.2.41 #define EXPRINFO_NUMOFPREDEF -(EXPRINFO_UNDEFINED)

This is the number of standard operator types.

Definition at line 359 of file expression.h.

8.3.2.42 #define EXPRINFO_OR -39

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 269 of file expression.h.

8.3.2.43 #define EXPRINFO_POLY -21

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 175 of file expression.h.

8.3.2.44 #define EXPRINFO_POW -22

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

Definition at line 181 of file expression.h.

8.3.2.45 #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 105 of file expression.h.

8.3.2.46 #define EXPRINFO_PSD -52

This node checks whether a matrix is positive semidefinite.

Definition at line 336 of file expression.h.

8.3.2.47 #define EXPRINFO_PWCONSTLC -34

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

Definition at line 238 of file expression.h.

8.3.2.48 #define EXPRINFO_PWCONSTRC -35

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

Definition at line 242 of file expression.h.

8.3.2.49 #define EXPRINFO_PWLIN -32

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

Definition at line 230 of file expression.h.

8.3.2.50 #define EXPRINFO_QUAD -26

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 203 of file expression.h.

8.3.2.51 #define EXPRINFO_RE -27

This node returns the real part of a complex number.

Definition at line 209 of file expression.h.

8.3.2.52 #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 122 of file expression.h.

8.3.2.53 #define EXPRINFO_SIN -18

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

Definition at line 161 of file expression.h.

8.3.2.54 #define EXPRINFO_SPLINE -33

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

Definition at line 234 of file expression.h.

8.3.2.55 #define EXPRINFO_SQROOT -13

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

Definition at line 141 of file expression.h.

8.3.2.56 #define EXPRINFO_SQUARE -12

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

Definition at line 137 of file expression.h.

8.3.2.57 #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 96 of file expression.h.

8.3.2.58 #define EXPRINFO_UNDEFINED -57

This is the artifical number for undefined nodes.

Definition at line 356 of file expression.h.

8.3.2.59 #define EXPRINFO_VARIABLE -2

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

Definition at line 84 of file expression.h.

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 1054 of file model.h.

8.3.3.2 typedef model::walker coco::expression_walker

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

Definition at line 1049 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 94 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 36 of file exprnames.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::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::sum_deltas
  Pre-post visitor for summing up all the deltas during work node extraction.
- class coco::search_graph
  The search graph.
- 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::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >
  The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

- class sum_deltas

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

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.

- typedef uint32_t coco::search_node_id

  Type of the unique search node identifier.

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.

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 333 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 326 of file search_graph.h.

8.4.2.3 typedef uint32_t coco::search_node_id

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

Definition at line 77 of file search_node.h.

8.4.3 Enumeration Type Documentation

8.4.3.1 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 61 of file search_node.h.

8.4.4 Function Documentation

8.4.4.1 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.

8.4.4.2 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 122 of file search_graph.cc.

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.

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.

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::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.

8.5.3 Function Documentation

8.5.3.1 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.

8.5.3.2 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.

8.6 Evaluators

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.
- struct coco::der_cache
  Cache data for der_eval.
- struct coco::diffI_eval_type
  Visitor data for diffI_eval.
- class coco::diffI_eval
  Forward function range evaluation.
- struct coco::diffNumber_eval_type
  Visitor data for diffNumber_eval.
- class coco::diffNumber_eval
  Forward function range evaluation.
- class coco::variable_indicator
  Bitmap class used to indicate variable occurrence.
- class coco::_evaluator_base< _Tp, _NData, _Result, _Walker >
  Base class of all evaluators.
- class coco::evaluator_base< _Tp, _NData, _Result, _Walker >
  Base class of all (non-caching) evaluators.
- class coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >
  Base class of all caching evaluators.
- class coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >
  Base class of all (non-caching) forward evaluators.
- class coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >
  Base class of all (non-caching) backward evaluators.
- class coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >
  Base class of all (non-caching) forward evaluators.
- class coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >

Base class of all caching backward evaluators.

- struct coco::func_eval_type

  Visitor data for func_eval.

- class coco::func_eval

  Forward function evaluation.

- struct coco::func_cache

  Cache data for func_eval.

- 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

  Visitor data for islp_eval.

- class coco::islp_eval

  Backward first order slope evaluation with prepared first order slope data.

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.

Variables

- struct coco::der_cache * coco::f_data

  Cache data for der_eval.

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 Function Documentation

8.6.2.1 template<class _Walker , class _Visitor > _Visitor::return_value coco::_recursive_short_cut_walk (_Walker __w, _Visitor __f) [inline]

This **internal** function performs 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.

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

Definition at line 1079 of file evaluator.h.

8.6.2.2 template<class _Visitor , class _Walker > _Visitor::return_value coco::evaluate (_Visitor __v, _Walker __start) [inline]

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.

8.6.2.3 template<class _Walker , class _Visitor > _Visitor::return_value coco::recursive_short_cut_walk (_Walker __w, _Visitor __f) [inline]

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

`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`.

Definition at line 989 of file evaluator.h.

8.6.3 Variable Documentation

8.6.3.1 struct coco::der_cache* coco::f_data

This class is the cache data type for the der_eval evaluator. Visitor data for der_eval

This class is the visitor data type for the der_eval evaluator. function evaluation cache

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

- std::ostream & coco::operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.

- std::ostream & 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 45 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 64 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 52 of file callback_ref.cc.

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

Disequality comparison operator with a convex_info.

Definition at line 250 of file semantics.h.

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

Disequality comparison operator with a convex_info.

Definition at line 242 of file semantics.h.

8.7.4.6 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 234 of file semantics.h.

8.7.4.7 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.

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

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

Definition at line 251 of file semantics.cc.

8.7.4.9 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 258 of file semantics.h.

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

Equality comparison operator with a convex_info.

Definition at line 226 of file semantics.h.

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

Equality comparison operator with a convex_info.

Definition at line 218 of file semantics.h.

8.7.4.12 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 210 of file semantics.h.

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::search_graph_context

  An evaluation context when retrieving from the search database.

- class coco::work_node_context

  The 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.

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.

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, 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::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, 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_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 long int coco::rand_t

This is the type returned by the coconut_random function.

Definition at line 90 of file coconut_random.h.

8.9.3.3 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.

8.9.4 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 42 of file search_graph_funcs.cc.

8.9.4.3 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.

8.9.4.4 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 (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.

8.9.4.6 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.

8.9.4.7 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.

8.9.4.8 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 727 of file sparsity.cc.

8.9.4.9 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*)

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 760 of file sparsity.cc.

8.9.4.10 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*)

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 692 of file sparsity.cc.

8.9.4.11 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 736 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*, 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).

8.9.4.13 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 660 of file sparsity.cc.

8.9.4.14 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 649 of file sparsity.cc.

8.9.4.15 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 639 of file sparsity.cc.

8.9.4.16 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 626 of file sparsity.cc.

9 Namespace Documentation

9.1 coco Namespace Reference

the main namespace of the COCONUT API

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.

- 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 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 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.

- 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 der_cache

  Cache data for der_eval.

- struct dfunc_eval_rettype
- struct dfunc_eval_type
- class dfunc_eval
- class diameter_comp_hook

  The log-volume computation hook (work node computation hook).

- struct diffI_eval_type

  Visitor data for diffI_eval.

- class diffI_eval

  Forward function range evaluation.

- struct diffNumber_eval_type

  Visitor data for diffNumber_eval.

- class diffNumber_eval

  Forward function range evaluation.

- 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.

- struct func_cache

  Cache data for func_eval.

- 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 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

- 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 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

  Visitor data for islp_eval.

- class islp_eval

  Backward first order slope evaluation with prepared first order slope data.

- class locopt_ret_record
- class logvol_comp_hook

  The log-volume 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 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 convex_e

  Convexity information.

- 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 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.

- class work_node_context

  The evaluation context when retrieving from the search database.

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 double(∗ func_evaluator)(const std::vector< double > ∗__x, const variable_indicator &__v)
- typedef diffI(∗ diffI_evaluator)(const std::vector< diffI > ∗__x, const variable_indicator &__v)
- typedef diffNumber(∗ diffNumber_evaluator)(const std::vector< diffNumber > ∗__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 **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 uint32_t search_node_id

  Type of the unique search node identifier.

- typedef std::pair< unsigned int, unsigned int > ipair

  Row-column index pair.

- typedef std::set< ipair > splist

  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 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 convex_info { c_convex = 1, c_linear = 0, c_concave = -1, c_maybe = 2 }

  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 sp_indexing { 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.

- const basic_alltype & basic_alltype_empty ()
- std::ostream & operator<< (std::ostream &os, 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 ()
- 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::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 graphorder (const model &DAG, std::vector< unsigned int > &order)

  Compute a graph order.

- void hess_star_color1 (const unsigned int n, const linalg::matrix< int > &H, std::vector< std::vector< unsigned int > > &parts, std::vector< linalg::sparse_vector< unsigned int > > &provides)
- 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, double b)
- bool operator!= (const interval &a, double b)
- interval cancel (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)
- 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)

- 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 log (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 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)
- 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 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 >
  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 > 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.

- bool operator== (const convex_e &__c, const convex_e &__d)

  Equality comparison operator.

- bool operator== (const convex_e &__c, const convex_info &__d)

  Equality comparison operator with a convex_info.

- bool operator== (const convex_info &__c, const convex_e &__d)

  Equality comparison operator with a convex_info.

- bool operator!= (const convex_e &__c, const convex_e &__d)

  Disequality comparison operator.

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

Disequality comparison operator with a convex_info.

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

  Disequality comparison operator with a convex_info.

- std::ostream & operator<< (std::ostream &o, const convex_e &__s)

  C++ stream output operator for convex_e.

- 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, bool org_only, bool H_upper_only, sp_indexing indexing, bool fun_one_based, bool var_one_based)
- 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 _TH >
  std::string convert_to_str (const _TH &_h)

  Convert an object of any printable class to a string.

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

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

- 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 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 division_part1 (const interval &x, const interval &y, bool &b)
- interval division_part2 (const interval &x, const interval &y, bool b=true)

- 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, 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 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, 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 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)
  Sparsity calculation (Jacobian).

Variables

- struct coco::der_cache * f_data
  Cache data for der_eval.

- std::vector< der_cache > * d_data
- std::vector< double > * grad_vec
- const model * mod
- double mult
- double mult_trans
- unsigned int child_n
- std::vector< std::vector< void * > > standard_evaluators
- const char * expr_names [ ]

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 45 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 45 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 45 of file cint_evaluator.h.

9.1.2.4 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 51 of file der_evaluator.h.

9.1.2.5 typedef diffI(∗ coco::diffI_evaluator)(const std::vector< diffI > ∗__x, const variable_indicator &__v)

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

Definition at line 46 of file diffI_evaluator.h.

9.1.2.6 typedef diffNumber(∗ coco::diffNumber_evaluator)(const std::vector< diffNumber > ∗__x, const variable_indicator &__v)

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

Definition at line 46 of file diffNumber_evaluator.h.

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

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

Definition at line 429 of file expression.h.

9.1.2.8 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 48 of file der_evaluator.h.

9.1.2.9 typedef double(∗ 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.

9.1.2.10 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 48 of file ider_evaluator.h.

9.1.2.11 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 59 of file islp_evaluator.h.

9.1.2.12 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 51 of file ider_evaluator.h.

9.1.2.13 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 55 of file iderf_evaluator.h.

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

Definition at line 45 of file infb_evaluator.h.

9.1.2.15 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 45 of file int_evaluator.h.

9.1.2.16 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.

9.1.2.17 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 62 of file islp_evaluator.h.

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

Definition at line 63 of file locopt_ret.h.

9.1.2.19 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.

9.1.2.20 typedef bool(∗ coco::prep_d_evaluator)()

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

Definition at line 47 of file der_evaluator.h.

9.1.2.21 typedef bool(∗ coco::prep_id_evaluator)()

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

Definition at line 47 of file ider_evaluator.h.

9.1.2.22 typedef bool(∗ coco::prep_islp_evaluator)()

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

Definition at line 58 of file islp_evaluator.h.

9.1.2.23 typedef interval coco::rhs_t

This type is the allowed type for constraint restrictions.

Definition at line 426 of file expression.h.

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

Definition at line 35 of file search_graph_funcs.h.

9.1.2.25 typedef enum coco::tristate_e coco::tristate

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

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

Definition at line 33 of file annotation_delta.cc.

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 94 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 or the expression is neither convex nor concave.

Bug

It should be possible to specify c_not explicitly to identify expressions proven to be not convex(concave) and distinguish that from c_maybe.

Definition at line 45 of file semantics.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 369 of file expression.h.

9.1.3.5 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

Definition at line 54 of file sparsity.h.

9.1.3.6 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 34 of file coconut_types.h.

9.1.3.7 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 59 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 279 of file expression.hpp.

9.1.4.2 template<class I > projective_interval<I> coco::abs (const projective_interval< I > & *x*) [inline]

Definition at line 880 of file prointerval.hpp.

9.1.4.3 proj_rational coco::abs (const proj_rational & *r*)

Definition at line 120 of file prointerval.h.

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

Definition at line 542 of file interval_set.cc.

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

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

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

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

Definition at line 756 of file interval_boost.h.

9.1.4.7 template<class I > projective_interval<I> coco::acos (const projective_interval< I > & *x*) [inline]

Definition at line 871 of file prointerval.hpp.

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

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

9.1.4.9 template<class I > projective_interval<I> coco::acosh (const projective_interval< I > & *x*) [inline]

Definition at line 888 of file prointerval.hpp.

9.1.4.10 interval coco::acosh (const interval & *x*)

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

9.1.4.11 template<class I > projective_interval<I> coco::acot (const projective_interval< I > & *x*) [inline]

Definition at line 897 of file prointerval.hpp.

9.1.4.12 interval coco::acot (const interval & *x*)

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

9.1.4.13 template<class I > projective_interval<I> coco::acoth (const projective_interval< I > & *x*) [inline]

Definition at line 906 of file prointerval.hpp.

9.1.4.14 interval coco::acoth (const interval & *x*)

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

9.1.4.15 template<class I > projective_interval<I> coco::asin (const projective_interval< I > & *x*) [inline]

Definition at line 915 of file prointerval.hpp.

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

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

9.1.4.17 template<class I > projective_interval<I> coco::asinh (const projective_interval< I > & *x*) [inline]

Definition at line 924 of file prointerval.hpp.

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

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

9.1.4.19 template<class I > projective_interval<I> coco::atan (const projective_interval< I > & *x*) [inline]

Definition at line 933 of file prointerval.hpp.

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

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

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

Definition at line 580 of file interval_set.cc.

9.1.4.22 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 749 of file interval_boost.h.

9.1.4.23 template<class I > projective_interval<I> coco::atanh (const projective_interval< I > & *x*) [inline]

Definition at line 942 of file prointerval.hpp.

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

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

9.1.4.25 const basic_alltype & coco::basic_alltype_empty ()

Definition at line 43 of file basic_alltype.cc.

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

9.1.4.27 template<class I > projective_interval<I> coco::cos (const projective_interval< I > & *x*) [inline]

Definition at line 951 of file prointerval.hpp.

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

Definition at line 560 of file interval_set.cc.

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

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

9.1.4.30 template<class I > projective_interval<I> coco::cosh (const projective_interval< I > & *x*) [inline]

Definition at line 960 of file prointerval.hpp.

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

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

9.1.4.32 template<class I > projective_interval<I> coco::cot (const projective_interval< I > & *x*) [inline]

Definition at line 969 of file prointerval.hpp.

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

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

9.1.4.34 template<class I > projective_interval<I> coco::coth (const projective_interval< I > & *x*) [inline]

Definition at line 978 of file prointerval.hpp.

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

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

9.1.4.36 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.

9.1.4.37 template<typename _TR > _TR coco::dbccmps_abs (const _TR & *x*) [inline]

Definition at line 118 of file dbcompare.hpp.

9.1.4.38 double coco::diam (const interval &)

diam(a) == a.diam()

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

Same as x.dist(y)

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

Definition at line 425 of file interval_set.h.

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

Definition at line 421 of file interval_set.h.

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

Definition at line 452 of file interval_set.cc.

9.1.4.43 template<class I > projective_interval<I> coco::division_part1 (const projective_interval< I > & *x*, const projective_interval< I > & *y*, bool & *b*) [inline]

Definition at line 1152 of file prointerval.hpp.

9.1.4.44 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.

9.1.4.45 template<class I > projective_interval<I> coco::division_part2 (const projective_interval< I > & *x*, const projective_interval< I > & *y*, bool *b*) [inline]

Definition at line 1162 of file prointerval.hpp.

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

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.

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

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

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

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

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

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

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

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

9.1.4.51 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.

9.1.4.52 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.

9.1.4.53 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*) [inline]

Definition at line 39 of file dbcompare.cc.

9.1.4.54 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*) [inline]

9.1.4.55 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*) [inline]

9.1.4.56 template<typename _TR > bool coco::eval_s (const vdbl::rowid & *rid*, const std::string & *colname*, dbc_method *tp*, vdbl::view * *iv*, _TR *cmpval*) [inline]

Definition at line 127 of file dbcompare.cc.

9.1.4.57 template<> bool coco::eval_s< interval > (const vdbl::rowid & *rid*, const std::string & *colname*, dbc_method *tp*, vdbl::view * *iv*, interval *cmpval*) [inline]

9.1.4.58 template<typename _TR > bool coco::eval_u (const vdbl::rowid & *rid*, const std::string & *colname*, dbc_method *tp*, vdbl::view * *iv*, _TR *cmpval*) [inline]

Definition at line 263 of file dbcompare.cc.

9.1.4.59 template<class I > projective_interval<I> coco::exp (const projective_interval< I > & *x*) [inline]

Definition at line 987 of file prointerval.hpp.

9.1.4.60 interval_set coco::exp (const interval_set & *a*)

Definition at line 524 of file interval_set.cc.

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

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

9.1.4.62 template<class I > projective_interval<I> coco::exp10 (const projective_interval< I > & *x*) [inline]

Definition at line 996 of file prointerval.hpp.

9.1.4.63 interval coco::exp10 (const interval & *x*)

Returns an interval enclosure of the exponential (base 10) of the interval x.

9.1.4.64 template<class I > projective_interval<I> coco::exp2 (const projective_interval< I > & *x*) [inline]

Definition at line 1005 of file prointerval.hpp.

9.1.4.65 interval coco::exp2 (const interval & *x*)

Returns an interval enclosure of the exponential (base 2) of the interval x.

9.1.4.66 template<class I > projective_interval<I> coco::expm1 (const projective_interval< I > & *x*) [inline]

Definition at line 1014 of file prointerval.hpp.

9.1.4.67 interval coco::expm1 (const interval & *x*)

Returns an interval enclosure of exp(x)-1

9.1.4.68 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*) [inline]

Definition at line 403 of file dbcompare.hpp.

9.1.4.69 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*) [inline]

Definition at line 41 of file dbcompare.hpp.

9.1.4.70 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*) [inline]

Definition at line 239 of file dbcompare.hpp.

9.1.4.71 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) [inline]

Definition at line 326 of file dbcompare.hpp.

9.1.4.72 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 771 of file interval_boost.h.

9.1.4.73 interval coco::gauss (const interval & *x*) [inline]

This function computes the gaussian function e^{-x^2} for intervals.

Definition at line 742 of file interval_boost.h.

9.1.4.74 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 58 of file search_graph_funcs.cc.

9.1.4.75 void coco::get_ids (const std::vector< si_ri_pair > & *riv*, std::vector< search_node_id > & *v*)

Definition at line 33 of file search_graph_funcs.cc.

9.1.4.76 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.

9.1.4.77 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.

9.1.4.78 tableid coco::get_point_table (database * *ptr*, standard_table *& *ptb*, userid *_uid*)

Definition at line 41 of file dbtables.cc.

9.1.4.79 void coco::hess_star_color1 (const unsigned int *n*, const linalg::matrix< int > & *H*, std::vector< std::vector< unsigned int > > & *parts*, std::vector< linalg::sparse_vector< unsigned int > > & *provides*)

Definition at line 35 of file hess_color.cc.

9.1.4.80 interval coco::hull (const interval & *__x*, const interval & *__y*) [inline]

Definition at line 899 of file interval_filib.h.

9.1.4.81 interval coco::hulldiff (const interval & *__x*, const interval & *__y*) [inline]

Definition at line 905 of file interval_filib.h.

9.1.4.82 std::string coco::i2a (int *x*)

This function converts integers to a C++ string.

Definition at line 37 of file writer_utils.cc.

9.1.4.83 template<class I > projective_interval<I> coco::imax (const projective_interval< I > & *x*, const projective_interval< I > & *y*) [inline]

Definition at line 1134 of file prointerval.hpp.

9.1.4.84 interval_set coco::imax (const interval_set & *a*, const interval_set & *b*)

Definition at line 485 of file interval_set.cc.

9.1.4.85 interval coco::imax (const interval & *x*, const interval & *y*)

Returns an interval enclosure of the maximum of two intervals x and y.

9.1.4.86 template<class I > projective_interval<I> coco::imin (const projective_interval< I > & *x*, const projective_interval< I > & *y*) [inline]

Definition at line 1143 of file prointerval.hpp.

9.1.4.87 interval_set coco::imin (const interval_set & *a*, const interval_set & *b*)

Definition at line 474 of file interval_set.cc.

9.1.4.88 interval coco::imin (const interval & *x*, const interval & *y*)

Returns an interval enclosure of the minimum of two intervals x and y.

9.1.4.89 interval coco::intersect (const interval & __*x*, const interval & __*y*) [inline]

Definition at line 893 of file interval_filib.h.

9.1.4.90 interval coco::ipow (const interval & *x*, int *n*) [inline]

This function computes the integer power operation x^n for intervals.

Definition at line 735 of file interval_boost.h.

9.1.4.91 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) [inline]

Definition at line 441 of file dbcompare.hpp.

9.1.4.92 std::string coco::localdatetime ()

This function converts a date/time structure to a C++ string.

Definition at line 68 of file writer_utils.cc.

9.1.4.93 template<class I > projective_interval<I> coco::log (const projective_interval< I > & *x*) [inline]

Definition at line 1023 of file prointerval.hpp.

9.1.4.94 interval_set coco::log (const interval_set & *a*)

Definition at line 533 of file interval_set.cc.

9.1.4.95 interval coco::log (const interval & *x*)

Returns an interval enclosure of the natural logarithm of the interval x.

9.1.4.96 template<class I > projective_interval<I> coco::log10 (const projective_interval< I > & *x*) [inline]

Definition at line 1032 of file prointerval.hpp.

9.1.4.97 interval coco::log10 (const interval & *x*)

Returns an interval enclosure of the logarithm (base 10) of the interval x.

9.1.4.98 template<class I > projective_interval<I> coco::log1p (const projective_interval< I > & *x*) [inline]

Definition at line 1041 of file prointerval.hpp.

9.1.4.99 interval coco::log1p (const interval & *x*)

Returns an interval enclosure of log(1+x).

9.1.4.100 template<class I > projective_interval<I> coco::log2 (const projective_interval< I > & *x*) [inline]

Definition at line 1050 of file prointerval.hpp.

9.1.4.101 interval coco::log2 (const interval & *x*)

Returns an interval enclosure of the logarithm (base 2) of the interval x.

9.1.4.102 double coco::mag (const interval &)

mag(a) == a.mag()

9.1.4.103 double coco::mid (const interval &)

mid(a) == a.mid()

9.1.4.104 double coco::mig (const interval &)

mig(a) == a.mig()

9.1.4.105 bool coco::operator!= (int *n*, const proj_rational & *R*)

Definition at line 119 of file prointerval.h.

9.1.4.106 bool coco::operator!= (const interval & *a*, double *b*) [inline]

Definition at line 492 of file interval_boost.h.

9.1.4.107 bool coco::operator!= (const interval & *__i*, const interval & *__d*) [inline]

Comparison operator (not equal)

Definition at line 490 of file interval_boost.h.

9.1.4.108 template<class _TC > bool coco::operator!= (const interval & *__i*, const _TC & *__d*) [inline]

Comparison operator (not equal)

Definition at line 794 of file interval_boost.h.

9.1.4.109 template<class I > projective_interval<I> coco::operator* (double *b*, const projective_interval< I > & *a*) [inline]

Definition at line 837 of file prointerval.hpp.

9.1.4.110 template<class I > projective_interval<I> coco::operator* (const projective_interval< I > & *a*, double *b*) [inline]

Definition at line 829 of file prointerval.hpp.

9.1.4.111 template<class I > projective_interval<I> coco::operator* (const projective_interval< I > & *a*, const projective_interval< I > & *b*) [inline]

Definition at line 819 of file prointerval.hpp.

9.1.4.112 interval_set coco::operator* (double *a*, const interval_set & *b*) [inline]

Definition at line 412 of file interval_set.h.

9.1.4.113 interval_set coco::operator∗ (const interval_set & *a*, double *b*) [inline]

Definition at line 405 of file interval_set.h.

9.1.4.114 interval_set coco::operator∗ (const interval & *a*, const interval_set & *b*) [inline]

Definition at line 398 of file interval_set.h.

9.1.4.115 interval_set coco::operator∗ (const interval_set & *a*, const interval & *b*) [inline]

Definition at line 391 of file interval_set.h.

9.1.4.116 interval_set coco::operator∗ (const interval_set & *a*, const interval_set & *b*) [inline]

Definition at line 383 of file interval_set.h.

9.1.4.117 interval coco::operator∗ (double *b*, const interval & *a*)

This operator computes a∗b as interval.

9.1.4.118 interval coco::operator∗ (const interval & *a*, double *b*)

This operator computes a∗b as interval.

9.1.4.119 interval coco::operator∗ (const interval & *a*, const interval & *b*)

This operator computes a∗b as interval.

9.1.4.120 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 324 of file search_node.hpp.

9.1.4.121 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 297 of file search_node.hpp.

9.1.4.122 template<class I > projective_interval<I> coco::operator+ (double *b*, const projective_interval< I > & *a*) [inline]

Definition at line 742 of file prointerval.hpp.

9.1.4.123 template<class I > projective_interval<I> coco::operator+ (const projective_interval< I > & *a*, double *b*) [inline]

Definition at line 723 of file prointerval.hpp.

9.1.4.124 template<class I > projective_interval<I> coco::operator+ (const projective_interval< I > & *a*, const projective_interval< I > & *b*) [inline]

Definition at line 703 of file prointerval.hpp.

9.1.4.125 interval_set coco::operator+ (double *a*, const interval_set & *b*) [inline]

Definition at line 333 of file interval_set.h.

9.1.4.126 interval_set coco::operator+ (const interval_set & *a*, double *b*) [inline]

Definition at line 326 of file interval_set.h.

9.1.4.127 interval_set coco::operator+ (const interval & *a*, const interval_set & *b*) [inline]

Definition at line 319 of file interval_set.h.

9.1.4.128 interval_set coco::operator+ (const interval_set & *a*, const interval & *b*) [inline]

Definition at line 312 of file interval_set.h.

9.1.4.129 interval_set coco::operator+ (const interval_set & *a*, const interval_set & *b*) [inline]

Definition at line 304 of file interval_set.h.

9.1.4.130 interval coco::operator+ (double *b*, const interval & *a*)

This operator computes a+b as interval.

9.1.4.131 interval coco::operator+ (const interval & *a*, double *b*)

This operator computes a+b as interval.

9.1.4.132 interval coco::operator+ (const interval & *a*, const interval & *b*)

This operator computes a+b as interval.

9.1.4.133 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 340 of file search_node.hpp.

9.1.4.134 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 310 of file search_node.hpp.

9.1.4.135 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 364 of file search_node.hpp.

9.1.4.136 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 35 of file search_node.cc.

9.1.4.137 template<class I > projective_interval<I> coco::operator- (double *b*, const projective_interval< I > & *a*) [inline]

Definition at line 800 of file prointerval.hpp.

9.1.4.138 template<class I > projective_interval<I> coco::operator- (const projective_interval< I > & *a*, double *b*) [inline]

Definition at line 781 of file prointerval.hpp.

9.1.4.139 template<class I > projective_interval<I> coco::operator- (const projective_interval< I > & *a*, const projective_interval< I > & *b*) [inline]

Definition at line 761 of file prointerval.hpp.

9.1.4.140 interval_set coco::operator- (const interval_set & *a*) [inline]

Definition at line 377 of file interval_set.h.

9.1.4.141 interval_set coco::operator- (double *a*, const interval_set & *b*) [inline]

Definition at line 370 of file interval_set.h.

9.1.4.142 interval_set coco::operator- (const interval_set & *a*, double *b*) [inline]

Definition at line 363 of file interval_set.h.

9.1.4.143 interval_set coco::operator- (const interval & *a*, const interval_set & *b*) [inline]

Definition at line 356 of file interval_set.h.

9.1.4.144 interval_set coco::operator- (const interval_set & *a*, const interval & *b*) [inline]

Definition at line 349 of file interval_set.h.

9.1.4.145 interval_set coco::operator- (const interval_set & *a*, const interval_set & *b*) [inline]

Definition at line 341 of file interval_set.h.

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

This operator computes a-b as interval.

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

This operator computes a-b as interval.

9.1.4.148 interval coco::operator- (const interval & *a*, const interval & *b*)

This operator computes a-b as interval.

9.1.4.149 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 382 of file search_node.hpp.

9.1.4.150 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 76 of file search_node.cc.

9.1.4.151 template<class I > projective_interval<I> coco::operator/ (double *b*, const projective_interval< I > & *a*) [inline]

Definition at line 863 of file prointerval.hpp.

9.1.4.152 template<class I > projective_interval<I> coco::operator/ (const projective_interval< I > & *a*, double *b*) [inline]

Definition at line 855 of file prointerval.hpp.

9.1.4.153 template<class I > projective_interval<I> coco::operator/ (const projective_interval< I > & *a*, const projective_interval< I > & *b*) [inline]

Definition at line 845 of file prointerval.hpp.

9.1.4.154 interval_set coco::operator/ (double *a*, const interval_set & *b*) [inline]

Definition at line 459 of file interval_set.h.

9.1.4.155 interval_set coco::operator/ (const interval_set & *a*, double *b*) [inline]

Definition at line 452 of file interval_set.h.

9.1.4.156 interval_set coco::operator/ (const interval & *a*, const interval_set & *b*) [inline]

Definition at line 445 of file interval_set.h.

9.1.4.157 interval_set coco::operator/ (const interval_set & *a*, const interval & *b*) [inline]

Definition at line 438 of file interval_set.h.

9.1.4.158 interval_set coco::operator/ (const interval_set & *a*, const interval_set & *b*) [inline]

Definition at line 430 of file interval_set.h.

9.1.4.159 interval coco::operator/ (double *b*, const interval & *a*)

This operator computes a/b as interval.

9.1.4.160 interval coco::operator/ (const interval & *a*, double *b*)

This operator computes a/b as interval.

9.1.4.161 interval coco::operator/ (const interval & *a*, const interval & *b*)

This operator computes a/b as interval.

9.1.4.162 bool coco::operator< (int *n*, const proj_rational & *R*)

Definition at line 114 of file prointerval.h.

9.1.4.163 bool coco::operator< (double *a*, const interval & *b*) [inline]

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 496 of file interval_boost.h.

9.1.4.164 bool coco::operator< (const interval & *a*, double *b*) [inline]

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 495 of file interval_boost.h.

9.1.4.165 bool coco::operator< (const interval & *a*, const interval & *b*) [inline]

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 494 of file interval_boost.h.

9.1.4.166 std::ostream & coco::operator<< (std::ostream & *s*, const interval & *a*) [inline]

Definition at line 963 of file interval_filib.h.

9.1.4.167 std::ostream& coco::operator<< (std::ostream & *o*, const expression_node & *__x*)

Definition at line 207 of file expression.cc.

9.1.4.168 std::ostream & coco::operator<< (std::ostream & *os*, const basic_alltype & *b*)

Definition at line 49 of file basic_alltype.cc.

9.1.4.169 bool coco::operator<= (int *n*, const proj_rational & *R*)

Definition at line 115 of file prointerval.h.

9.1.4.170 bool coco::operator== (int *n*, const proj_rational & *R*)

Definition at line 118 of file prointerval.h.

9.1.4.171 bool coco::operator== (const interval_set & *a*, const interval_set & *b*)

Definition at line 463 of file interval_set.cc.

9.1.4.172 bool coco::operator== (const interval & *a*, double *b*) [inline]

Definition at line 488 of file interval_boost.h.

9.1.4.173 bool coco::operator== (const interval & *__i*, const interval & *__d*) [inline]

Comparison operator (equal)

Definition at line 486 of file interval_boost.h.

9.1.4.174 template<class _TC > bool coco::operator== (const interval & *__i*, const _TC & *__d*) [inline]

Comparison operator (equal)

Definition at line 786 of file interval_boost.h.

9.1.4.175 bool coco::operator> (int *n*, const proj_rational & *R*)

Definition at line 116 of file prointerval.h.

9.1.4.176 bool coco::operator>= (int *n*, const proj_rational & *R*)

Definition at line 117 of file prointerval.h.

9.1.4.177 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.

9.1.4.178 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.

9.1.4.179 template<class I > projective_interval<I> coco::pow (const projective_interval< I > & *x*, const projective_interval< I > & *y*) [inline]

Definition at line 1068 of file prointerval.hpp.

9.1.4.180 interval_set coco::pow (const interval_set & *a*, const interval_set & *b*)

Definition at line 569 of file interval_set.cc.

9.1.4.181 interval coco::pow (const interval & *x*, const interval & *y*)

Returns an interval enclosure of x^y = exp(y∗log(x)).

9.1.4.182 template<class I > projective_interval<I> coco::power (const projective_interval< I > & *x*, int *n*) [inline]

Definition at line 1059 of file prointerval.hpp.

9.1.4.183 interval_set coco::power (const interval_set & *a*, int *n*)

Definition at line 496 of file interval_set.cc.

9.1.4.184 interval coco::power (const interval & *x*, int *n*)

Returns an interval enclosure of x^n.

9.1.4.185 double coco::r_random_h_eval (double *t*, double *b*, double *a*) [inline]

Definition at line 123 of file coconut_random.h.

9.1.4.186 double coco::r_random_hinv_eval (double *x*, double *b*, double *a*) [inline]

Definition at line 136 of file coconut_random.h.

9.1.4.187 double coco::rad (const interval &)

rad(a) == a.rad()

9.1.4.188 double coco::relDiam (const interval &)

relDiam(a) == a.relDiam()

9.1.4.189 interval_set coco::round_to_integer (const interval_set & *a*)

This function rounds the interval inward to integer borders.

Definition at line 592 of file interval_set.cc.

9.1.4.190 interval coco::round_to_integer (const interval & *x*)

This function rounds the interval inward to integer borders.

9.1.4.191 double coco::safeguarded_mid (const interval & *__i*) [inline]

This function computes the safeguarded midpoint of ___i.

Definition at line 464 of file interval_boost.h.

9.1.4.192 template<class I > projective_interval<I> coco::sin (const projective_interval< I > & *x*) [inline]

Definition at line 1080 of file prointerval.hpp.

9.1.4.193 interval_set coco::sin (const interval_set & *a*)

Definition at line 551 of file interval_set.cc.

9.1.4.194 interval coco::sin (const interval & *x*)

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

9.1.4.195 template<class I > projective_interval<I> coco::sinh (const projective_interval< I > & *x*) [inline]

Definition at line 1089 of file prointerval.hpp.

9.1.4.196 interval coco::sinh (const interval & *x*)

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

9.1.4.197 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 705 of file sparsity.cc.

9.1.4.198 template<class I > projective_interval<I> coco::sqr (const projective_interval< I > & *x*) [inline]

Definition at line 1098 of file prointerval.hpp.

9.1.4.199 interval_set coco::sqr (const interval_set & *a*)

Definition at line 506 of file interval_set.cc.

9.1.4.200 interval coco::sqr (const interval & *x*)

Returns an interval enclosure of x^2.

9.1.4.201 template<class I > projective_interval<I> coco::sqrt (const projective_interval< I > & *x*) [inline]

Definition at line 1107 of file prointerval.hpp.

9.1.4.202 interval_set coco::sqrt (const interval_set & *a*)

Definition at line 515 of file interval_set.cc.

9.1.4.203 interval coco::sqrt (const interval & *x*)

Returns an interval enclosure of the square root of the interval x.

9.1.4.204 std::vector<std::vector<void *> > coco::standard_evaluators (NUM_EVALUATORS)

This variables contains pointers to all standard evaluation routines for user defined nodes.

9.1.4.205 template<class I > projective_interval<I> coco::tan (const projective_interval< I > & *x*) [inline]

Definition at line 1116 of file prointerval.hpp.

9.1.4.206 interval coco::tan (const interval & *x*)

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

9.1.4.207 template<class I > projective_interval<I> coco::tanh (const projective_interval< I > & *x*) [inline]

Definition at line 1125 of file prointerval.hpp.

9.1.4.208 interval coco::tanh (const interval & *x*)

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

9.1.4.209 double coco::upward_divides (const double & *a*, const double & *b*)

This function divides a by b, rounding upwards.

9.1.4.210 double coco::upward_minus (const double & *a*, const double & *b*)

This function subtracts b from a, rounding upwards.

9.1.4.211 double coco::upward_multiplies (const double & *a*, const double & *b*)

This function multiplies a by b, rounding upwards.

9.1.4.212 double coco::upward_plus (const double & *a*, const double & *b*)

This function adds a and b, rounding upwards.

9.1.4.213 double coco::width (const interval_set & *a*)

Definition at line 449 of file interval_set.cc.

9.1.4.214 double coco::width (const interval &)

9.1.5 Variable Documentation

9.1.5.1 unsigned int coco::child_n

children counter

Definition at line 1001 of file der_evaluator_n.h.

9.1.5.2 std::vector<der_cache>* coco::d_data

derivative cache

Definition at line 996 of file der_evaluator_n.h.

9.1.5.3 std::vector<double>* coco::grad_vec

the gradient vector

Definition at line 997 of file der_evaluator_n.h.

9.1.5.4 const model* coco::mod

the DAG

Definition at line 998 of file der_evaluator_n.h.

9.1.5.5 double coco::mult

the current value on the path

Definition at line 999 of file der_evaluator_n.h.

9.1.5.6 double coco::mult_trans

transfer variable for mult

Definition at line 1000 of file der_evaluator_n.h.

9.1.5.7 std::vector<std::vector<void *> > coco::standard_evaluators

9.2 coco_api_internal Namespace Reference

9.3 linalg Namespace Reference

The namespace for all linear algebra types and functions.

Classes

- class matrix
  Sparse Matrix Template Class (row major).
- class c_matrix
  Sparse Matrix Template Class (column major).
- class sparse_vector
  Sparse Vector Template Class.
- class __linalg_cvec
  Adapter class for a const std::vector to a linear algebra vector.
- class __linalg_vec
  Adapter class for a std::vector to a linear algebra vector.

Enumerations

- enum linalg_output_format { linalg_output_sparse, linalg_output_full }
  The possible output formats of a matrix.

Functions

- template<class _Tp >
  matrix< _Tp > operator+ (const matrix< _Tp > &A, const matrix< _Tp > &B)
  Sparse matrix addition operator.

- template<class _Tp >
  matrix< _Tp > operator* (_Tp a, const matrix< _Tp > &B)

  Sparse scalar matrix multiplication operator.

- template<class _Tp >
  c_matrix< _Tp > operator+ (const c_matrix< _Tp > &A, const c_matrix< _Tp > &B)

  Sparse matrix addition operator.

- template<class _Tp >
  c_matrix< _Tp > operator* (_Tp a, const c_matrix< _Tp > &B)

  Sparse matrix scalar multiplication operator.

- template<class _Tp >
  bool operator== (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector comparison operator.

- template<class _Tp >
  sparse_vector< _Tp > operator+ (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector addition operator.

- template<class _Tp >
  _Tp operator* (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector scalar product operator.

- template<class _Tp >
  sparse_vector< _Tp > operator* (_Tp a, const sparse_vector< _Tp > &b)

  Sparse vector scaling operator.

- template<class _Ve >
  std::vector< _Ve > & linalg_ssum (std::vector< _Ve > &a, _Ve b, const std::vector< _Ve > &c)
- template<class _Ve >
  std::vector< _Ve > & linalg_smult (std::vector< _Ve > &a, _Ve b)

  - template<class _TS >
    _TS linalg_dot (const std::vector< _TS > &a, const std::vector< _TS > &b, _TS s)
  - template<class _Va , class _TS >
    _TS linalg_dot (const _Va &a, const std::vector< _TS > &b, _TS s)
  - template<class _Vb , class _TS >
    _TS linalg_dot (const std::vector< _TS > &a, const _Vb &b, _TS s)
  - template<class _Va , class _Vb , class _TS >
    _TS linalg_dot (const _Va &a, const _Vb &b, _TS s)

  - template<class _TS >
    void linalg_add (const std::vector< _TS > &a, const std::vector< _TS > &b, std::vector< _TS > &c)
  - template<class _Va , class _TS >
    void linalg_add (const _Va &a, const std::vector< _TS > &b, std::vector< _TS > &c)
  - template<class _Vb , class _TS >
    void linalg_add (const std::vector< _TS > &a, const _Vb &b, std::vector< _TS > &c)

- template<class _TS , class _Vc >
  void linalg_add (const std::vector< _TS > &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Va , class _Vb , class _TS >
  void linalg_add (const _Va &a, const _Vb &b, std::vector< _TS > &c)
- template<class _Va , class _TS , class _Vc >
  void linalg_add (const _Va &a, const std::vector< _TS > &b, _Vc &c)
- template<class _TS , class _Vb , class _Vc >
  void linalg_add (const std::vector< _TS > &a, const _Vb &b, _Vc &c)
- template<class _Va , class _Vb , class _Vc >
  void linalg_add (const _Va &a, const _Vb &b, _Vc &c)

- template<class _Matrix , class _TS >
  void linalg_matvec (const _Matrix &A, const std::vector< _TS > &a, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Va >
  void linalg_matvec (const _Matrix &A, const _Va &a, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Vc >
  void linalg_matvec (const _Matrix &A, const std::vector< _TS > &a, _Vc &c)
- template<class _Matrix , class _Va , class _Vc >
  void linalg_matvec (const _Matrix &A, const _Va &a, _Vc &c)

- template<class _Matrix , class _TS >
  void linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Va >
  void linalg_matvecv (const _Matrix &A, const _Va &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Vb >
  void linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const _Vb &b, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Vc >
  void linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Matrix , class _Va , class _Vb , class _TS >
  void linalg_matvecv (const _Matrix &A, _Va &a, const _Vb &b, const std::vector< _TS > &c)
- template<class _Matrix , class _Va , class _TS , class _Vc >
  void linalg_matvecv (const _Matrix &A, const _Va &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Matrix , class _TS , class _Vb , class _Vc >
  void linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const _Vb &b, _Vc &c)
- template<class _Matrix , class _Va , class _Vb , class _Vc >
  void linalg_matvecv (const _Matrix &A, const _Va &a, const _Vb &b, _Vc &c)

9.3.1 Detailed Description

This namespace is for all linear algebra related types and operations.

9.4 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 _Tp , int NN, int IND_OFFSET>
  ostream & operator<< (ostream &o, const mtl::light1D< _Tp, NN, IND_OFFSET > &a)
- template<typename _Tp >
  ostream & operator<< (ostream &s, const linalg::matrix< _Tp > &A)
- template<typename _Tp >
  ostream & operator<< (ostream &s, const linalg::c_matrix< _Tp > &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 _Tp >
  ostream & operator<< (ostream &s, const linalg::sparse_vector< _Tp > &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.

9.4.1 Detailed Description

This is the standard namespace of the C++ library.

9.4.2 Enumeration Type Documentation

9.4.2.1 anonymous enum

Enumerator:

_M_p_chunk_size

Definition at line 55 of file stlp_addalgo.h.

9.4.2.2 anonymous enum

Enumerator:

_M_p_threshold

Definition at line 56 of file stlp_addalgo.h.

9.4.3 Function Documentation

9.4.3.1 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*) [inline]

Definition at line 230 of file stlp_pairheap.h.

9.4.3.2 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*) [inline]

Definition at line 155 of file stlp_pairheap.h.

9.4.3.3 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*) [inline]

Definition at line 1187 of file stlp_addalgo.h.

9.4.3.4 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Distance > void std::__chunk_insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Distance *__chunk_size*) [inline]

Definition at line 1171 of file stlp_addalgo.h.

9.4.3.5 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > void std::__final_insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Compare *__comp*) [inline]

Definition at line 311 of file stlp_addalgo.h.

9.4.3.6 template<typename _RandomAccessIter , typename _RandomAccessIterp > void std::__final_insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*) [inline]

Definition at line 289 of file stlp_addalgo.h.

9.4.3.7 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > void std::__inplace_stable_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Compare *__comp*) [inline]

Definition at line 1087 of file stlp_addalgo.h.

9.4.3.8 template<typename _RandomAccessIter , typename _RandomAccessIterp > void std::__inplace_stable_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*) [inline]

Definition at line 1062 of file stlp_addalgo.h.

9.4.3.9 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > void std::__insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Compare *__comp*) [inline]

Definition at line 216 of file stlp_addalgo.h.

9.4.3.10 template<typename _RandomAccessIter , typename _RandomAccessIterp > void std::__insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*) [inline]

Definition at line 186 of file stlp_addalgo.h.

9.4.3.11 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*) [inline]

Definition at line 365 of file stlp_addalgo.h.

9.4.3.12 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size > void std::__intro_p_sort_loop (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Size *__depth_limit*) [inline]

Definition at line 333 of file stlp_addalgo.h.

9.4.3.13 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*) [inline]

Definition at line 428 of file stlp_addalgo.h.

9.4.3.14 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Size > void std::__introsort_p_loop (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Size *__depth_limit*) [inline]

This is a helper function for the pair_sort routine.

Definition at line 396 of file stlp_addalgo.h.

9.4.3.15 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*) [inline]

Definition at line 1140 of file stlp_addalgo.h.

9.4.3.16 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*) [inline]

Definition at line 1110 of file stlp_addalgo.h.

9.4.3.17 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*) [inline]

Definition at line 1232 of file stlp_addalgo.h.

9.4.3.18 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*) [inline]

Definition at line 1203 of file stlp_addalgo.h.

9.4.3.19 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*) [inline]

Definition at line 617 of file stlp_addalgo.h.

9.4.3.20 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*) [inline]

Definition at line 555 of file stlp_addalgo.h.

9.4.3.21 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*) [inline]

Definition at line 751 of file stlp_addalgo.h.

9.4.3.22 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*) [inline]

Definition at line 679 of file stlp_addalgo.h.

9.4.3.23 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*) [inline]

Definition at line 1015 of file stlp_addalgo.h.

9.4.3.24 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*) [inline]

Definition at line 966 of file stlp_addalgo.h.

9.4.3.25 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.

9.4.3.26 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.

9.4.3.27 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*) [inline]

Definition at line 105 of file stlp_pairheap.h.

9.4.3.28 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*) [inline]

Definition at line 53 of file stlp_pairheap.h.

9.4.3.29 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*) [inline]

Definition at line 1291 of file stlp_addalgo.h.

9.4.3.30 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*) [inline]

Definition at line 1262 of file stlp_addalgo.h.

9.4.3.31 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 268 of file stlp_addalgo.h.

9.4.3.32 template<typename _RandomAccessIter , typename _RandomAccessIterp > void std::__unguarded_insertion_p_sort (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*) [inline]

Definition at line 246 of file stlp_addalgo.h.

9.4.3.33 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*) [inline]

Definition at line 160 of file stlp_addalgo.h.

9.4.3.34 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp , typename _Tpp > void std::__unguarded_linear_p_insert (_RandomAccessIter *__last1*, _Tp *__val1*, _RandomAccessIterp *__last2*, _Tpp *__val2*) [inline]

Definition at line 133 of file stlp_addalgo.h.

9.4.3.35 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*) [inline]

Definition at line 99 of file stlp_addalgo.h.

9.4.3.36 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Tp > pair<_RandomAccessIter, _RandomAccessIterp> std::__unguarded_p_partition (_RandomAccessIter *__first*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Tp *__pivot*) [inline]

Definition at line 65 of file stlp_addalgo.h.

9.4.3.37 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.

9.4.3.38 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp > void std::make_pair_heap (_RandomAccessIterator *__first*, _RandomAccessIterator *__last*, _RandomAccessIteratorp *__firstp*, _RandomAccessIteratorp *__lastp*) [inline]

Definition at line 303 of file stlp_pairheap.h.

9.4.3.39 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.

9.4.3.40 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.

9.4.3.41 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.

9.4.3.42 template<class I > ostream& std::operator<< (ostream & *o*, const coco::projective_interval< I > & *a*) [inline]

Definition at line 189 of file prointerval.hpp.

9.4.3.43 ostream& std::operator<< (ostream & *o*, const coco::proj_rational & *x*)

Definition at line 180 of file prointerval.hpp.

9.4.3.44 ostream & std::operator<< (ostream & *s*, const coco::interval_set & *a*)

Definition at line 601 of file interval_set.cc.

9.4.3.45 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.

9.4.3.46 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.

9.4.3.47 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.

9.4.3.48 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.

9.4.3.49 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*) [inline]

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 907 of file stlp_addalgo.h.

9.4.3.50 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*) [inline]

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 834 of file stlp_addalgo.h.

9.4.3.51 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 523 of file stlp_addalgo.h.

9.4.3.52 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 475 of file stlp_addalgo.h.

9.4.3.53 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > bool std::partial_pair_sort (_RandomAccessIter *__first*, _RandomAccessIter *__middle*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*, _Compare *__comp*) [inline]

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 1507 of file stlp_addalgo.h.

9.4.3.54 template<typename _RandomAccessIter , typename _RandomAccessIterp > bool std::partial_pair_sort (_RandomAccessIter *__first*, _RandomAccessIter *__middle*, _RandomAccessIter *__last*, _RandomAccessIterp *__firstp*, _RandomAccessIterp *__lastp*) [inline]

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 1449 of file stlp_addalgo.h.

9.4.3.55 template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare > bool std::partial_sort (_RandomAccessIter *__first*, _RandomAccessIter *__middle*, _RandomAccessIter *__last*, _RandomAccessIter *__firstp*, _RandomAccessIter *__lastp*, _Compare *__comp*) [inline]

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 1570 of file stlp_addalgo.h.

9.4.3.56 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.

9.4.3.57 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.

9.4.3.58 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.

9.4.3.59 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.

9.4.3.60 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp , typename _Compare > void std::sort_pair_heap (_RandomAccessIterator *__first*, _RandomAccessIterator *__last*, _RandomAccessIteratorp *__firstp*, _RandomAccessIteratorp *__lastp*, _Compare *__comp*) [inline]

Definition at line 403 of file stlp_pairheap.h.

9.4.3.61 template<typename _RandomAccessIterator , typename _RandomAccessIteratorp > void std::sort_pair_heap (_RandomAccessIterator *__first*, _RandomAccessIterator *__last*, _RandomAccessIteratorp *__firstp*, _RandomAccessIteratorp *__lastp*) [inline]

Definition at line 383 of file stlp_pairheap.h.

9.4.3.62 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 1342 of file stlp_addalgo.h.

9.4.3.63 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 1397 of file stlp_addalgo.h.

10 Class Documentation

10.1 linalg::__linalg_cvec< _TS > Class Template Reference

Adapter class for a const std::vector to a linear algebra vector.

`#include <linalg.h>`

Public Types

- enum { N = 0 }
- typedef std::vector< _TS >::value_type value_type
- typedef std::vector< _TS >::reference reference
- typedef std::vector< _TS >::pointer pointer
- typedef std::vector< _TS >::const_reference const_reference
- typedef std::vector< _TS >::const_pointer const_pointer
- typedef std::vector< _TS >::size_type size_type
- typedef std::vector< _TS >::difference_type difference_type
- typedef std::vector< _TS >::allocator_type allocator_type
- typedef std::vector< _TS >::iterator iterator
- typedef std::vector< _TS >::const_iterator const_iterator
- typedef std::vector< _TS >::reverse_iterator reverse_iterator
- typedef std::vector< _TS >::const_reverse_iterator const_reverse_iterator
- typedef mtl::dense_tag sparsity
- typedef mtl::oned_tag dimension
- typedef mtl::scaled1D< _Self > scaled_type
- typedef mtl::light1D< _TS > subrange_type
- typedef _Self IndexArray
- typedef _Self IndexArrayRef

Public Member Functions

- __linalg_cvec (const std::vector< _TS > &s)
- ~__linalg_cvec ()
- unsigned int size () const

- iterator begin ()
- iterator end ()
- const_iterator begin () const
- const_iterator end () const

10.1.1 Detailed Description

template<class _TS> class linalg::__linalg_cvec< _TS >

This adapter class is used to transform a const std::vector to a vector which can be used by the linear algebra operations of the MTL.

Definition at line 847 of file linalg.h.

10.1.2 Member Typedef Documentation

10.1.2.1 template<class _TS> typedef std::vector<_TS>::allocator_type linalg::__linalg_cvec< _TS >::allocator_type

These definitions are all required for the linear algebra vector type.

Definition at line 866 of file linalg.h.

10.1.2.2 template<class _TS> typedef std::vector<_TS>::const_iterator linalg::__linalg_cvec< _TS >::const_iterator

These definitions are all required for the linear algebra vector type.

Definition at line 868 of file linalg.h.

10.1.2.3 template<class _TS> typedef std::vector<_TS>::const_pointer linalg::__linalg_cvec< _TS >::const_pointer

These definitions are all required for the linear algebra vector type.

Definition at line 863 of file linalg.h.

10.1.2.4 template<class _TS> typedef std::vector<_TS>::const_reference linalg::__linalg_cvec< _TS >::const_reference

These definitions are all required for the linear algebra vector type.

Definition at line 862 of file linalg.h.

10.1.2.5 template<class _TS> typedef std::vector<_TS>::const_reverse_iterator linalg::__linalg_cvec< _TS >::const_reverse_iterator

These definitions are all required for the linear algebra vector type.

Definition at line 870 of file linalg.h.

10.1.2.6 template<class _TS> typedef std::vector<_TS>::difference_type linalg::__linalg_cvec< _TS >::difference_type

These definitions are all required for the linear algebra vector type.

Definition at line 865 of file linalg.h.

10.1.2.7 template<class _TS> typedef mtl::oned_tag linalg::__linalg_cvec< _TS >::dimension

These definitions are all required for the linear algebra vector type.

Definition at line 873 of file linalg.h.

10.1.2.8 template<class _TS> typedef _Self linalg::__linalg_cvec< _TS >::IndexArray

These definitions are all required for the linear algebra vector type.

Definition at line 878 of file linalg.h.

10.1.2.9 template<class _TS> typedef _Self linalg::__linalg_cvec< _TS >::IndexArrayRef

These definitions are all required for the linear algebra vector type.

Definition at line 879 of file linalg.h.

10.1.2.10 template<class _TS> typedef std::vector<_TS>::iterator linalg::__linalg_cvec< _TS >::iterator

These definitions are all required for the linear algebra vector type.

Definition at line 867 of file linalg.h.

10.1.2.11 template<class _TS> typedef std::vector<_TS>::pointer linalg::__linalg_cvec< _TS >::pointer

These definitions are all required for the linear algebra vector type.

Definition at line 861 of file linalg.h.

10.1.2.12 template<class _TS> typedef std::vector<_TS>::reference linalg::__linalg_cvec< _TS >::reference

These definitions are all required for the linear algebra vector type.

Definition at line 860 of file linalg.h.

10.1.2.13 template<class _TS> typedef std::vector<_TS>::reverse_iterator linalg::__linalg_cvec< _TS >::reverse_iterator

These definitions are all required for the linear algebra vector type.

Definition at line 869 of file linalg.h.

10.1.2.14 template<class _TS> typedef mtl::scaled1D<_Self> linalg::__linalg_cvec< _TS >::scaled_type

These definitions are all required for the linear algebra vector type.

Definition at line 875 of file linalg.h.

10.1.2.15 template<class _TS> typedef std::vector<_TS>::size_type linalg::__linalg_cvec< _TS >::size_type

These definitions are all required for the linear algebra vector type.

Definition at line 864 of file linalg.h.

10.1.2.16 template<class _TS> typedef mtl::dense_tag linalg::__linalg_cvec< _TS >::sparsity

These definitions are all required for the linear algebra vector type.

Definition at line 872 of file linalg.h.

10.1.2.17 template<class _TS> typedef mtl::light1D<_TS> linalg::__linalg_cvec< _TS >::subrange_type

These definitions are all required for the linear algebra vector type.

Definition at line 877 of file linalg.h.

10.1.2.18 template<class _TS> typedef std::vector<_TS>::value_type linalg::__linalg_cvec< _TS >::value_type

These definitions are all required for the linear algebra vector type.

Definition at line 859 of file linalg.h.

10.1.3 Member Enumeration Documentation

10.1.3.1 template<class _TS> anonymous enum

These definitions are all required for the linear algebra vector type.

Enumerator:

N

Definition at line 857 of file linalg.h.

10.1.4 Constructor & Destructor Documentation

10.1.4.1 template<class _TS> linalg::__linalg_cvec< _TS >::__linalg_cvec (const std::vector< _TS > & *s*) [inline]

This is the Standard Constructor

Definition at line 883 of file linalg.h.

10.1.4.2 template<class _TS> linalg::__linalg_cvec< _TS >::~__linalg_cvec () [inline]

This is the Standard Destructor

Definition at line 885 of file linalg.h.

10.1.5 Member Function Documentation

10.1.5.1 template<class _TS> const_iterator linalg::__linalg_cvec< _TS >::begin () const [inline]

The basic iterator methods are needed for a linear algebra vector and are directly translated.

Definition at line 893 of file linalg.h.

10.1.5.2 template<class _TS> iterator linalg::__linalg_cvec< _TS >::begin () [inline]

The basic iterator methods are needed for a linear algebra vector and are directly translated.

Definition at line 890 of file linalg.h.

10.1.5.3 template<class _TS> const_iterator linalg::__linalg_cvec< _TS >::end () const [inline]

The basic iterator methods are needed for a linear algebra vector and are directly translated.

Definition at line 894 of file linalg.h.

10.1.5.4 template<class _TS> iterator linalg::__linalg_cvec< _TS >::end () [inline]

The basic iterator methods are needed for a linear algebra vector and are directly translated.

Definition at line 891 of file linalg.h.

10.1.5.5 template<class _TS> unsigned int linalg::__linalg_cvec< _TS >::size () const [inline]

The size method is needed for a linear algebra vector and is directly translated.

Definition at line 899 of file linalg.h.

The documentation for this class was generated from the following file:

- linalg.h

10.2 linalg::__linalg_vec< _TS > Class Template Reference

Adapter class for a std::vector to a linear algebra vector.

`#include <linalg.h>`

Public Types

- enum { N = 0 }
- typedef std::vector< _TS >::value_type value_type
- typedef std::vector< _TS >::reference reference
- typedef std::vector< _TS >::pointer pointer
- typedef std::vector< _TS >::const_reference const_reference
- typedef std::vector< _TS >::const_pointer const_pointer
- typedef std::vector< _TS >::size_type size_type
- typedef std::vector< _TS >::difference_type difference_type
- typedef std::vector< _TS >::allocator_type allocator_type
- typedef std::vector< _TS >::iterator iterator
- typedef std::vector< _TS >::const_iterator const_iterator
- typedef std::vector< _TS >::reverse_iterator reverse_iterator
- typedef std::vector< _TS >::const_reverse_iterator const_reverse_iterator
- typedef mtl::dense_tag sparsity
- typedef mtl::oned_tag dimension
- typedef mtl::scaled1D< _Self > scaled_type
- typedef mtl::light1D< _TS > subrange_type
- typedef _Self IndexArray
- typedef _Self IndexArrayRef

Public Member Functions

- __linalg_vec (std::vector< _TS > &s)
- ~__linalg_vec ()
- unsigned int size () const

- iterator begin ()
- iterator end ()
- const_iterator begin () const
- const_iterator end () const

10.2.1 Detailed Description

template<class _TS> class linalg::__linalg_vec< _TS >

This adapter class is used to transform a std::vector to a vector which can be used by the linear algebra operations of the MTL.

Definition at line 908 of file linalg.h.

10.2.2 Member Typedef Documentation

10.2.2.1 template<class _TS > typedef std::vector<_TS>::allocator_type linalg::__linalg_vec< _TS >::allocator_type

These definitions are all required for the linear algebra vector type.

Definition at line 927 of file linalg.h.

10.2.2.2 template<class _TS > typedef std::vector<_TS>::const_iterator linalg::__linalg_vec< _TS >::const_iterator

These definitions are all required for the linear algebra vector type.

Definition at line 929 of file linalg.h.

10.2.2.3 template<class _TS > typedef std::vector<_TS>::const_pointer linalg::__linalg_vec< _TS >::const_pointer

These definitions are all required for the linear algebra vector type.

Definition at line 924 of file linalg.h.

10.2.2.4 template<class _TS > typedef std::vector<_TS>::const_reference linalg::__linalg_vec< _TS >::const_reference

These definitions are all required for the linear algebra vector type.

Definition at line 923 of file linalg.h.

10.2.2.5 template<class _TS > typedef std::vector<_TS>::const_reverse_iterator linalg::__linalg_vec< _TS >::const_reverse_iterator

These definitions are all required for the linear algebra vector type.

Definition at line 931 of file linalg.h.

10.2.2.6 template<class _TS > typedef std::vector<_TS>::difference_type linalg::__linalg_vec< _TS >::difference_type

These definitions are all required for the linear algebra vector type.

Definition at line 926 of file linalg.h.

10.2.2.7 template<class _TS > typedef mtl::oned_tag linalg::__linalg_vec< _TS >::dimension

These definitions are all required for the linear algebra vector type.

Definition at line 934 of file linalg.h.

10.2.2.8 template<class _TS > typedef _Self linalg::__linalg_vec< _TS >::IndexArray

These definitions are all required for the linear algebra vector type.

Definition at line 939 of file linalg.h.

10.2.2.9 template<class _TS > typedef _Self linalg::__linalg_vec< _TS >::IndexArrayRef

These definitions are all required for the linear algebra vector type.

Definition at line 940 of file linalg.h.

10.2.2.10 template<class _TS > typedef std::vector<_TS>::iterator linalg::__linalg_vec< _TS >::iterator

These definitions are all required for the linear algebra vector type.

Definition at line 928 of file linalg.h.

10.2.2.11 template<class _TS > typedef std::vector<_TS>::pointer linalg::__linalg_vec< _TS >::pointer

These definitions are all required for the linear algebra vector type.

Definition at line 922 of file linalg.h.

10.2.2.12 template<class _TS > typedef std::vector<_TS>::reference linalg::__linalg_vec< _TS >::reference

These definitions are all required for the linear algebra vector type.

Definition at line 921 of file linalg.h.

10.2.2.13 template<class _TS > typedef std::vector<_TS>::reverse_iterator linalg::__linalg_vec< _TS >::reverse_iterator

These definitions are all required for the linear algebra vector type.

Definition at line 930 of file linalg.h.

10.2.2.14 template<class _TS > typedef mtl::scaled1D<_Self> linalg::__linalg_vec< _TS >::scaled_type

These definitions are all required for the linear algebra vector type.

Definition at line 936 of file linalg.h.

10.2.2.15 template<class _TS > typedef std::vector<_TS>::size_type linalg::__linalg_vec< _TS >::size_type

These definitions are all required for the linear algebra vector type.

Definition at line 925 of file linalg.h.

10.2.2.16 template<class _TS > typedef mtl::dense_tag linalg::__linalg_vec< _TS >::sparsity

These definitions are all required for the linear algebra vector type.

Definition at line 933 of file linalg.h.

10.2.2.17 template<class _TS > typedef mtl::light1D<_TS> linalg::__linalg_vec< _TS >::subrange_type

These definitions are all required for the linear algebra vector type.

Definition at line 938 of file linalg.h.

10.2.2.18 template<class _TS > typedef std::vector<_TS>::value_type linalg::__linalg_vec< _TS >::value_type

These definitions are all required for the linear algebra vector type.

Definition at line 920 of file linalg.h.

10.2.3 Member Enumeration Documentation

10.2.3.1 template<class _TS > anonymous enum

These definitions are all required for the linear algebra vector type.

Enumerator:

N

Definition at line 918 of file linalg.h.

10.2.4 Constructor & Destructor Documentation

10.2.4.1 template<class _TS > linalg::__linalg_vec< _TS >::__linalg_vec (std::vector< _TS > & *s*) [inline]

This is the Standard Constructor

Definition at line 944 of file linalg.h.

10.2.4.2 template<class _TS > linalg::__linalg_vec< _TS >::~__linalg_vec () [inline]

This is the Standard Destructor

Definition at line 946 of file linalg.h.

10.2.5 Member Function Documentation

10.2.5.1 template<class _TS > const_iterator linalg::__linalg_vec< _TS >::begin () const [inline]

The basic iterator methods are needed for a linear algebra vector and are directly translated.

Definition at line 954 of file linalg.h.

10.2.5.2 template<class _TS > iterator linalg::__linalg_vec< _TS >::begin () [inline]

The basic iterator methods are needed for a linear algebra vector and are directly translated.

Definition at line 951 of file linalg.h.

10.2.5.3 template<class _TS > const_iterator linalg::__linalg_vec< _TS >::end () const [inline]

The basic iterator methods are needed for a linear algebra vector and are directly translated.

Definition at line 955 of file linalg.h.

10.2.5.4 template<class _TS > iterator linalg::__linalg_vec< _TS >::end () [inline]

The basic iterator methods are needed for a linear algebra vector and are directly translated.

Definition at line 952 of file linalg.h.

10.2.5.5 template<class _TS > unsigned int linalg::__linalg_vec< _TS >::size () const [inline]

The size method is needed for a linear algebra vector and is directly translated.

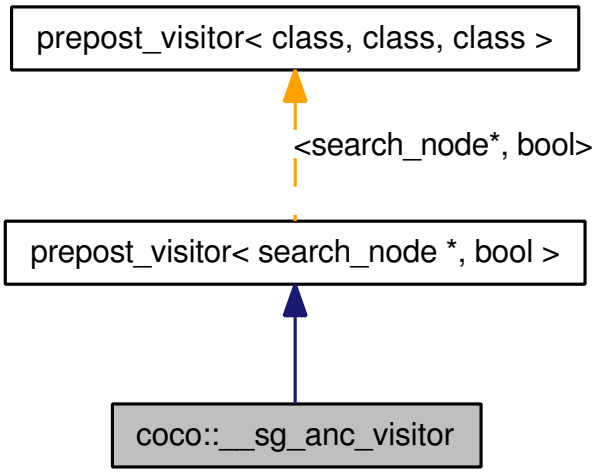
Definition at line 960 of file linalg.h.

The documentation for this class was generated from the following file:

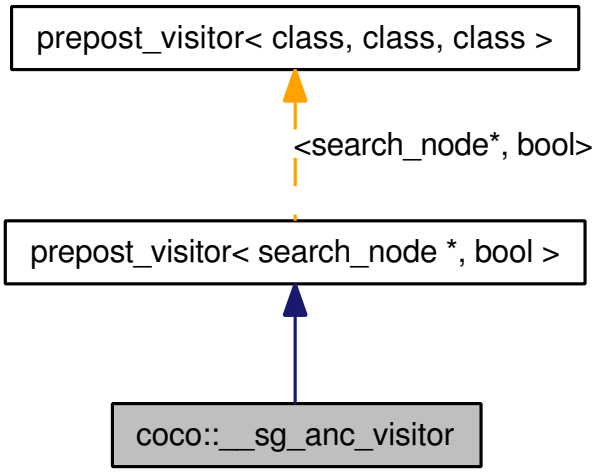
- linalg.h

10.3 coco::__sg_anc_visitor Class Reference

Inheritance diagram for coco::__sg_anc_visitor:



Collaboration diagram for coco::__sg_anc_visitor:



Public Member Functions

- __sg_anc_visitor (std::set< search_node_id > &anc)
- ~__sg_anc_visitor ()
- virtual bool **preorder** (const _Node &__n)
- virtual bool **postorder** (const _Node &__n)
- virtual void **collect** (const _Node &__n, collect_value __r)

- 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 ()

10.3.1 Detailed Description

Definition at line 261 of file search_graph.cc.

10.3.2 Constructor & Destructor Documentation

10.3.2.1 coco::__sg_anc_visitor::__sg_anc_visitor (std::set< search_node_id > & *anc*) [inline]

Constructor

Definition at line 268 of file search_graph.cc.

10.3.2.2 coco::__sg_anc_visitor::~__sg_anc_visitor () [inline]

Destructor

Definition at line 271 of file search_graph.cc.

10.3.3 Member Function Documentation

10.3.3.1 virtual void prepost_visitor< class, class, class >::collect (const _Node & *__n*, collect_value *__r*) [virtual, inherited]

10.3.3.2 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 306 of file search_graph.cc.

10.3.3.3 virtual bool prepost_visitor< class, class, class >::postorder (const _Node & *__n*) [virtual, inherited]

10.3.3.4 bool coco::__sg_anc_visitor::postorder (search_node *const & *r*) [inline]

This method is required by a **prepost_visitor**.

Definition at line 300 of file search_graph.cc.

10.3.3.5 virtual bool prepost_visitor< class, class, class >::preorder (const _Node & *__n*) [virtual, inherited]

10.3.3.6 bool coco::__sg_anc_visitor::preorder (search_node *const & *r*) [inline]

This method is required by a **prepost_visitor**.

Definition at line 293 of file search_graph.cc.

10.3.3.7 return_value coco::__sg_anc_visitor::value () [inline]

This method is required by a **prepost_visitor**.

Definition at line 311 of file search_graph.cc.

10.3.3.8 void coco::__sg_anc_visitor::vcollect (return_value const &) [inline]

This method is required by a **prepost_visitor**.

Definition at line 280 of file search_graph.cc.

10.3.3.9 void coco::__sg_anc_visitor::vinit () [inline, virtual]

This method is required by a **prepost_visitor**.

Reimplemented from **prepost_visitor< search_node *, bool >**.

Definition at line 275 of file search_graph.cc.

10.3.3.10 return_value coco::__sg_anc_visitor::vvalue () [inline, virtual]

This method is required by a **prepost_visitor**.

Reimplemented from **prepost_visitor< search_node *, bool >**.

Definition at line 287 of file search_graph.cc.

The documentation for this class was generated from the following file:

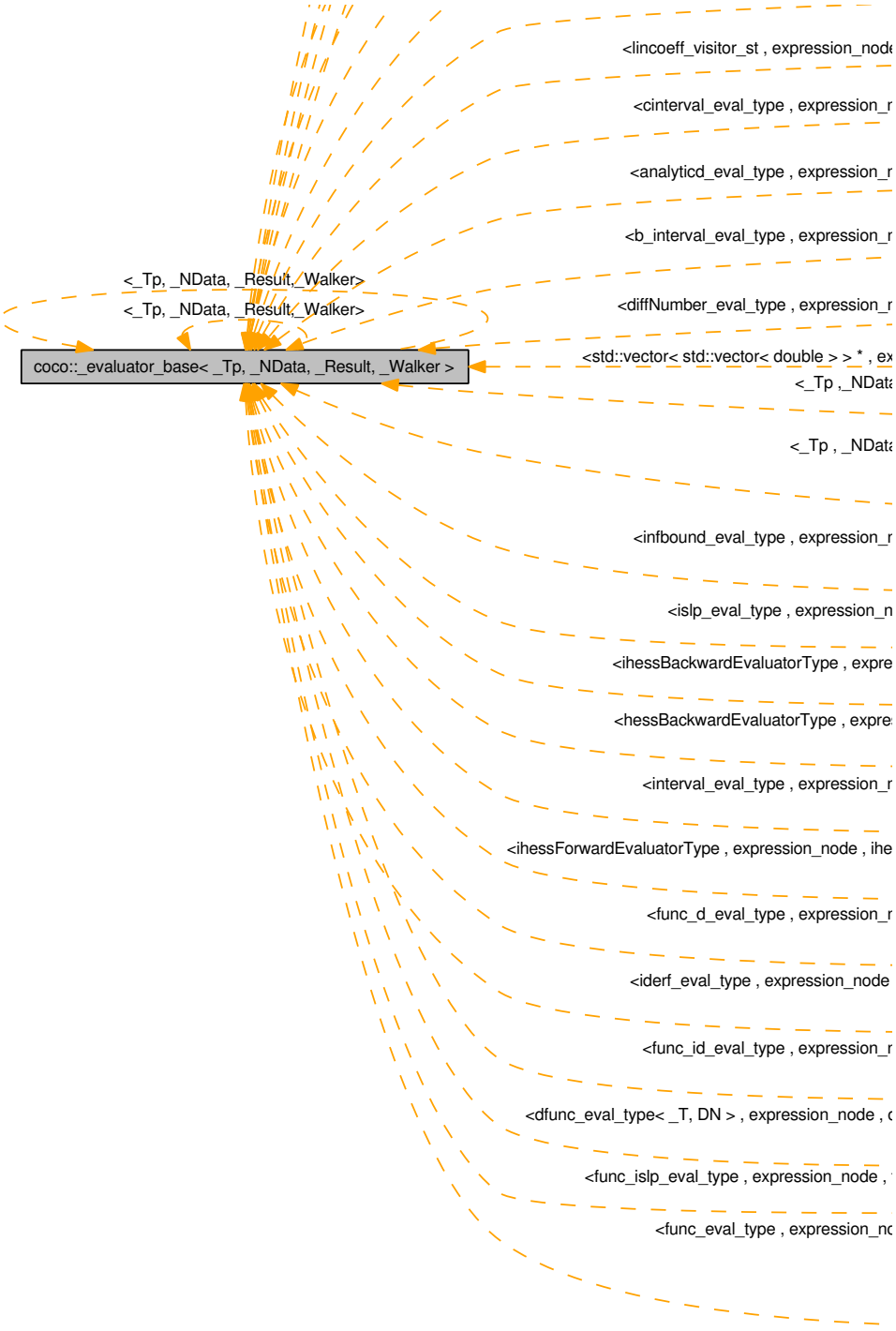
- search_graph.cc

10.4 coco::_evaluator_base< _Tp, _NData, _Result, _Walker > Class Template Reference

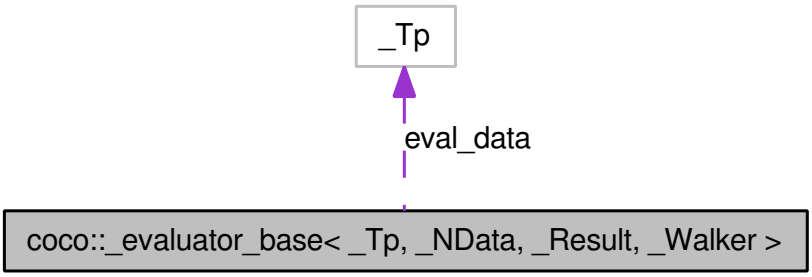
Base class of all evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::_evaluator_base< _Tp, _NData, _Result, _Walker >:



Collaboration diagram for coco::_evaluator_base< _Tp, _NData, _Result, _Walker >:



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

10.4.1 Detailed Description

template<class _Tp, class _NData, class _Result, class _Walker> class coco::_evaluator_base< _Tp, _NData, _Result, _Walker >

This class is the base class of all evaluators. Basically, it is a visitor to expression_node nodes in a DAG.

Definition at line 298 of file evaluator.h.

10.4.2 Member Typedef Documentation

10.4.2.1 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Walker coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented in coco::evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_walker >, coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >, coco::cached_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >, coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_const_walker >, coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >, coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >, coco::cached_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >, coco::cached_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >, coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >, coco::cached_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >, coco::cached_evaluator_base< islp_eval_type, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >, coco::cached_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >, coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >, coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >, coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker >, coco::cached_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >, coco::cached_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >, coco::cached_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >, coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hess-

ForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 312 of file evaluator.h.

10.4.2.2 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Tp coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::data_type

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.4.2.3 template<class _Tp, class _NData, class _Result, class _Walker> typedef _NData coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented in coco::evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_walker >, coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >, coco::cached_evaluator_base<

model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_-walker >, coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_-const_walker >, coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_-const_walker >, coco::cached_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< lincoeff_visitor_-st, expression_node, lincoeff_visitor_ret,model::const_walker >, coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >, coco::cached_evaluator_-base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >, coco::cached_-evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >, coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_-walker >, coco::cached_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >, coco::cached_evaluator_base< islp_eval_type, expression_-node, bool,expression_const_walker >, coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< hessBackwardEvalu-atorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< interval_-eval_type, expression_node, interval,expression_const_walker >, coco::cached_evaluator_base< ihess-ForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >, coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >, coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_-walker >, coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_-const_walker >, coco::cached_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >, coco::cached_evaluator_base< func_islp_-eval_type, expression_node, func_islp_return_type,expression_const_walker >, coco::cached_evaluator_-base< func_eval_type, expression_node, double,expression_const_walker >, coco::cached_forward_-evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_-forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_-eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hess-ForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_-walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_-node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_-0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessFor-wardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_-evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_-forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_-node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_-forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >,

coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_-const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hess-BackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_-evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_-backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 307 of file evaluator.h.

10.4.2.4 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Result coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::return_value

This type is the result type of the evaluator.

Reimplemented in coco::evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_-evaluator_base< _Tp, _NData, _Result, _Walker >, coco::forward_evaluator_base< _Tp, _-NData, _Result, _Walker >, coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_backward_-evaluator_base< _Tp, _NData, _Result, _Walker >, coco::evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_-const_walker >, coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >, coco::cached_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_-walker >, coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_-const_walker >, coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_-const_walker >, coco::cached_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< lincoeff_visitor_-st, expression_node, lincoeff_visitor_ret,model::const_walker >, coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >, coco::cached_evaluator_-base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >, coco::cached_-evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >, coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_-walker >, coco::cached_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >, coco::cached_evaluator_base< islp_eval_type, expression_-node, bool,expression_const_walker >, coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< hessBackwardEvalu-atorType, expression_node, bool,expression_const_walker >, coco::cached_evaluator_base< interval_-eval_type, expression_node, interval,expression_const_walker >, coco::cached_evaluator_base< ihess-ForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >, coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >, coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_-walker >, coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_-const_walker >, coco::cached_evaluator_base< dfunc_eval_type< _T, DN >, expression_node,

dfunc_eval_rettype< _T, DN >,expression_const_walker >, coco::cached_evaluator_base< func_islp_-eval_type, expression_node, func_islp_return_type,expression_const_walker >, coco::cached_evaluator_-base< func_eval_type, expression_node, double,expression_const_walker >, coco::cached_forward_-evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_-forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_-eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hess-ForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_-walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_-node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_-0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessFor-wardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_-evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_-forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_-node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_-forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_-const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hess-BackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_-evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_-backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 310 of file evaluator.h.

10.4.3 Constructor & Destructor Documentation

10.4.3.1 template<class _Tp, class _NData, class _Result, class _Walker> coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::_evaluator_base () [inline]

Standard Constructor

Definition at line 320 of file evaluator.h.

10.4.3.2 template<class _Tp, class _NData, class _Result, class _Walker> coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::_evaluator_base (const _Tp & *__x*) [inline]

Constructor, which initializes the internal data

Definition at line 322 of file evaluator.h.

10.4.3.3 template<class _Tp, class _NData, class _Result, class _Walker> coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::_evaluator_base (const _Self & *__x*) [inline]

Standard Copy Constructor

Definition at line 324 of file evaluator.h.

10.4.3.4 template<class _Tp, class _NData, class _Result, class _Walker> virtual coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::~_evaluator_base () [inline, virtual]

Standard Destructor

Definition at line 327 of file evaluator.h.

10.4.4 Member Function Documentation

10.4.4.1 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >,

coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 359 of file evaluator.h.

10.4.4.2 template<class _Tp, class _NData, class _Result, class _Walker> virtual void coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType,

expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 364 of file evaluator.h.

10.4.4.3 template<class _Tp, class _NData, class _Result, class _Walker> virtual return_value coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,

expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 336 of file evaluator.h.

10.4.4.4 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_-

eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 348 of file evaluator.h.

10.4.4.5 template<class _Tp, class _NData, class _Result, class _Walker> virtual return_value coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval,

expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 332 of file evaluator.h.

10.4.5 Member Data Documentation

10.4.5.1 template<class _Tp, class _NData, class _Result, class _Walker> _Tp coco::_evaluator_base< _Tp, _NData, _Result, _Walker >::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

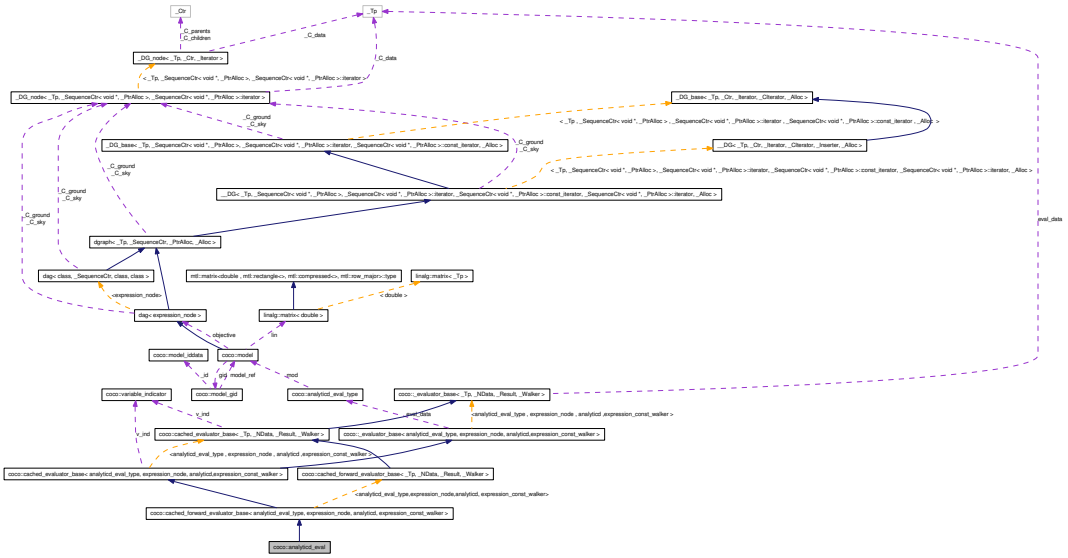
10.5 coco::analyticd_eval Class Reference

`#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
- typedef analyticd_eval_type data_type

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 void cleanup (const node_data_type &__data)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- analyticd_eval_type eval_data

10.5.1 Detailed Description

Definition at line 60 of file ade_evaluator.h.

10.5.2 Member Typedef Documentation

10.5.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.5.2.2 typedef analyticd_eval_type coco::_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.5.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.5.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.5.3 Constructor & Destructor Documentation

10.5.3.1 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 115 of file ade_evaluator.h.

10.5.3.2 coco::analyticd_eval::analyticd_eval (const analyticd_eval & *__v*) [inline]

Definition at line 127 of file ade_evaluator.h.

10.5.3.3 coco::analyticd_eval::~analyticd_eval () [inline]

Definition at line 129 of file ade_evaluator.h.

10.5.4 Member Function Documentation

10.5.4.1 void coco::analyticd_eval::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 192 of file ade_evaluator.h.

10.5.4.2 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::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 709 of file ade_evaluator.h.

10.5.4.3 virtual void coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::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 804 of file evaluator.h.

10.5.4.4 int coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.5.4.5 int coco::analyticd_eval::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 136 of file ade_evaluator.h.

10.5.4.6 void coco::analyticd_eval::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 134 of file ade_evaluator.h.

10.5.4.7 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::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 69 of file ade_evaluator.h.

10.5.4.8 void coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.5.4.9 int coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.5.4.10 void coco::analyticd_eval::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 204 of file ade_evaluator.h.

10.5.4.11 expression_const_walker coco::analyticd_eval::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 131 of file ade_evaluator.h.

10.5.4.12 int coco::analyticd_eval::update (const expression_node & *__data*, const analyticd & *__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.

Reimplemented from coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 229 of file ade_evaluator.h.

10.5.4.13 int coco::analyticd_eval::update (const analyticd & *__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.

Reimplemented from coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >.

Definition at line 223 of file ade_evaluator.h.

10.5.4.14 return_value coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.5.4.15 int coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.5.4.16 void coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::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 771 of file evaluator.h.

10.5.4.17 return_value coco::cached_forward_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< analyticd_eval_type, expression_node, analyticd,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.5.5 Member Data Documentation

10.5.5.1 analyticd_eval_type coco::_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.5.5.2 const variable_indicator* coco::cached_evaluator_base< analyticd_eval_type , expression_node , analyticd , expression_const_walker >::v_ind [protected, inherited]

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:

- ade_evaluator.h

10.6 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

10.6.1 Detailed Description

Definition at line 48 of file ade_evaluator.h.

10.6.2 Member Data Documentation

10.6.2.1 std::vector<analyticd>* coco::analyticd_eval_type::cache

Definition at line 51 of file ade_evaluator.h.

10.6.2.2 analyticd coco::analyticd_eval_type::d

Definition at line 54 of file ade_evaluator.h.

10.6.2.3 int coco::analyticd_eval_type::info

Definition at line 55 of file ade_evaluator.h.

10.6.2.4 const model* coco::analyticd_eval_type::mod

Definition at line 52 of file ade_evaluator.h.

10.6.2.5 unsigned int coco::analyticd_eval_type::n

Definition at line 57 of file ade_evaluator.h.

10.6.2.6 void* coco::analyticd_eval_type::p

Definition at line 53 of file ade_evaluator.h.

10.6.2.7 analyticd coco::analyticd_eval_type::r

Definition at line 56 of file ade_evaluator.h.

10.6.2.8 const std::vector<analyticd>* coco::analyticd_eval_type::x

Definition at line 50 of file ade_evaluator.h.

The documentation for this struct was generated from the following file:

- ade_evaluator.h

10.7 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

10.7.1 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 annotation.h.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 coco::annotation::annotation (const vdbl::tableid & *_ti*, const vdbl::rowid & *_ri*) [inline]

standard constructor

Definition at line 55 of file annotation.h.

10.7.2.2 virtual coco::annotation::~annotation () [inline, virtual]

standard destructor

Definition at line 58 of file annotation.h.

10.7.3 Member Function Documentation

10.7.3.1 vdbl::rowid coco::annotation::get_entry () const [inline]

retrieve the row_id part from the annotation

Definition at line 63 of file annotation.h.

10.7.3.2 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:

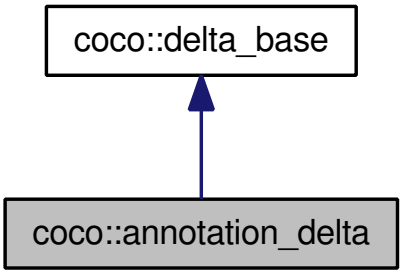
- annotation.h

10.8 coco::annotation_delta Class Reference

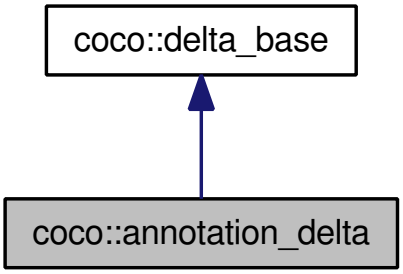
the delta class for annotation changes

`#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) 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)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Public Attributes

- std::vector< annotation > add
- std::vector< annotation > rm

Protected Attributes

- std::string _action

10.8.1 Detailed Description

The annotation_delta class describes the apply information for changes to the annotation structure in the work_node.

Definition at line 95 of file annotation_delta.h.

10.8.2 Constructor & Destructor Documentation

10.8.2.1 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.

10.8.2.2 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.

10.8.2.3 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.

10.8.2.4 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.

10.8.2.5 coco::annotation_delta::annotation_delta (const annotation_delta & *__d*) [inline]

standard copy constructor

Definition at line 131 of file annotation_delta.h.

10.8.2.6 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.

10.8.2.7 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.

10.8.2.8 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.

10.8.3 Member Function Documentation

10.8.3.1 bool coco::annotation_delta::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_did*) const [virtual]

apply operation: apply the delta with identity _did to work_node _x and construct in _u the undelta information.

Reimplemented from coco::delta_base.

Definition at line 35 of file annotation_delta.cc.

10.8.3.2 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *&& *_u*, const delta_id & *_d*) 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.

10.8.3.3 virtual void 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.

Reimplemented in coco::table_delta.

Definition at line 189 of file api_deltabase.h.

10.8.3.4 virtual void coco::annotation_delta::destroy_copy (delta_base * *__d*) const [inline, virtual]

clone destructor

Definition at line 164 of file annotation_delta.h.

10.8.3.5 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.8.3.6 delta 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 api_deltabase.h.

10.8.3.7 virtual annotation_delta* coco::annotation_delta::new_copy () const [inline, virtual]

clone operation

Reimplemented from coco::delta_base.

Definition at line 161 of file annotation_delta.h.

10.8.3.8 bool coco::annotation_delta::operator!= (const annotation_delta & _c) const [inline]

Definition at line 183 of file annotation_delta.h.

10.8.3.9 bool coco::annotation_delta::operator!= (const delta_base & _c) const [inline]

Definition at line 176 of file annotation_delta.h.

10.8.3.10 bool coco::annotation_delta::operator== (const annotation_delta & _c) const [inline]

Comparison operators

Definition at line 181 of file annotation_delta.h.

10.8.3.11 bool coco::annotation_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Definition at line 173 of file annotation_delta.h.

10.8.3.12 virtual void 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 api_deltabase.h.

10.8.4 Member Data Documentation

10.8.4.1 std::string coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file api_deltabase.h.

10.8.4.2 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.

10.8.4.3 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:

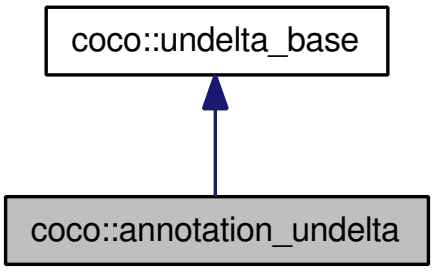
- annotation_delta.h
- annotation_delta.cc

10.9 coco::annotation_undelta Class Reference

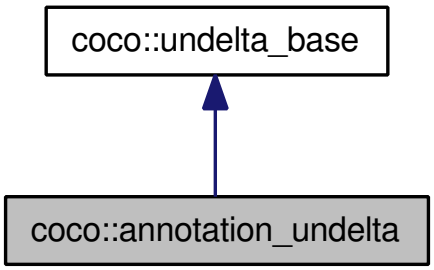
the undelta class for annotation changes

`#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 ()
- 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

10.9.1 Detailed Description

The annotation_undelta class describes the unapply information for changes to the annotation structure in the work_node.

Definition at line 40 of file annotation_delta.h.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 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.

10.9.2.2 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.

10.9.2.3 coco::annotation_undelta::annotation_undelta (const annotation & __a) [inline]

constructor for a single added annotation

Definition at line 63 of file annotation_delta.h.

10.9.2.4 coco::annotation_undelta::annotation_undelta (const annotation_undelta & __d) [inline]

standard copy constructor

Definition at line 68 of file annotation_delta.h.

10.9.3 Member Function Documentation

10.9.3.1 void coco::annotation_undelta::destroy_copy (undelta_base * __d) const [inline]

clone destructor

Definition at line 80 of file annotation_delta.h.

10.9.3.2 undelta coco::undelta_base::make_undelta () [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 283 of file api_deltabase.h.

10.9.3.3 annotation_undelta* coco::annotation_undelta::new_copy () const [inline, virtual]

cloning operation

Reimplemented from coco::undelta_base.

Definition at line 78 of file annotation_delta.h.

10.9.3.4 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::undelta_base.

Definition at line 62 of file annotation_delta.cc.

10.9.3.5 bool 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.

10.9.4 Friends And Related Function Documentation

10.9.4.1 friend class annotation_delta [friend]

Definition at line 86 of file annotation_delta.h.

10.9.5 Member Data Documentation

10.9.5.1 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.

10.9.5.2 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:

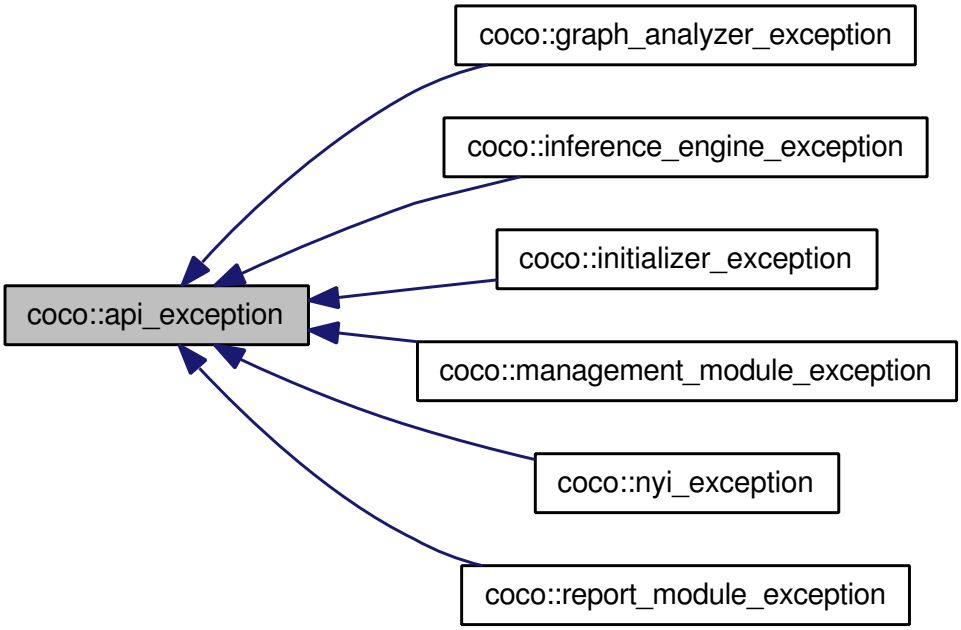
- annotation_delta.h
- annotation_delta.cc

10.10 coco::api_exception Class Reference

API exception class.

`#include <api_exception.h>`

Inheritance diagram for 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 ()

10.10.1 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 api_exception.h.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 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.

10.10.2.2 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.

10.10.2.3 virtual coco::api_exception::~api_exception () throw () [inline, virtual]

Standard Destructor

Definition at line 86 of file api_exception.h.

10.10.3 Member Function Documentation

10.10.3.1 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.

10.10.3.2 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.

10.10.3.3 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.

10.10.3.4 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 files:

- api_exception.h
- api_exception.cc

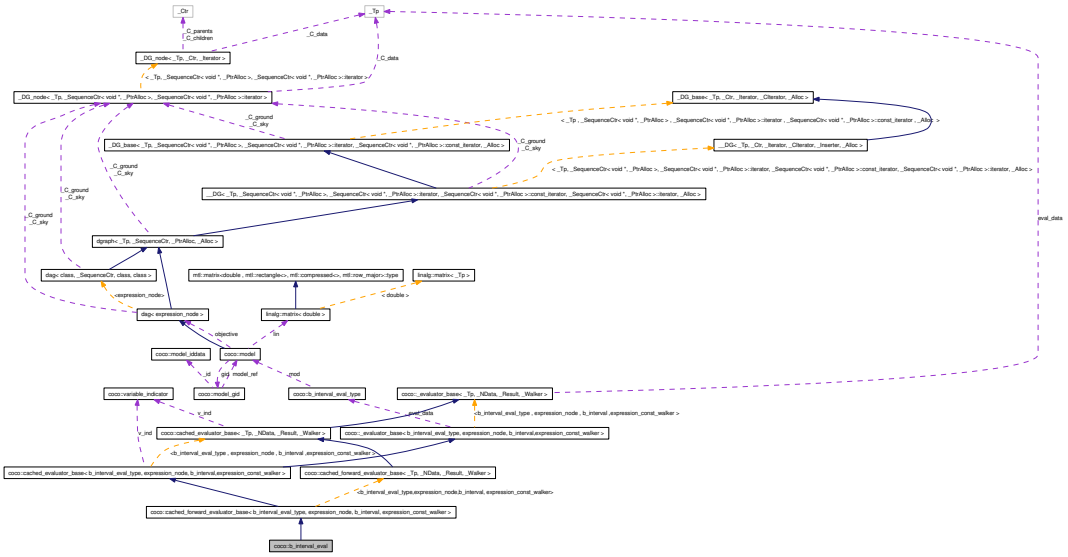
10.11 coco::b_interval_eval Class Reference

`#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
- typedef b_interval_eval_type data_type

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 void cleanup (const node_data_type &__data)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- b_interval_eval_type eval_data

10.11.1 Detailed Description

Definition at line 60 of file bint_evaluator.h.

10.11.2 Member Typedef Documentation

10.11.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.11.2.2 typedef b_interval_eval_type coco::_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.11.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.11.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.11.3 Constructor & Destructor Documentation

10.11.3.1 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 115 of file bint_evaluator.h.

10.11.3.2 coco::b_interval_eval::b_interval_eval (const b_interval_eval & *__v*) [inline]

Definition at line 127 of file bint_evaluator.h.

10.11.3.3 coco::b_interval_eval::~b_interval_eval () [inline]

Definition at line 129 of file bint_evaluator.h.

10.11.4 Member Function Documentation

10.11.4.1 void coco::b_interval_eval::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 192 of file bint_evaluator.h.

10.11.4.2 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::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 709 of file bint_evaluator.h.

10.11.4.3 virtual void coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::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 804 of file evaluator.h.

10.11.4.4 int coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.11.4.5 int coco::b_interval_eval::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 136 of file bint_evaluator.h.

10.11.4.6 void coco::b_interval_eval::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 134 of file bint_evaluator.h.

10.11.4.7 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::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 69 of file bint_evaluator.h.

10.11.4.8 void coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.11.4.9 int coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.11.4.10 void coco::b_interval_eval::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 204 of file bint_evaluator.h.

10.11.4.11 expression_const_walker coco::b_interval_eval::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 131 of file bint_evaluator.h.

10.11.4.12 int coco::b_interval_eval::update (const expression_node & *__data*, const b_interval & *__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.

Reimplemented from coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 229 of file bint_evaluator.h.

10.11.4.13 int coco::b_interval_eval::update (const b_interval & *__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.

Reimplemented from coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >.

Definition at line 223 of file bint_evaluator.h.

10.11.4.14 return_value coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.11.4.15 int coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.11.4.16 void coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::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 771 of file evaluator.h.

10.11.4.17 return_value coco::cached_forward_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< b_interval_eval_type, expression_node, b_interval,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.11.5 Member Data Documentation

10.11.5.1 b_interval_eval_type coco::_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.11.5.2 const variable_indicator* coco::cached_evaluator_base< b_interval_eval_type , expression_node , b_interval , expression_const_walker >::v_ind [protected, inherited]

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:

- bint_evaluator.h

10.12 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

10.12.1 Detailed Description

Definition at line 48 of file bint_evaluator.h.

10.12.2 Member Data Documentation

10.12.2.1 std::vector<b_interval>* coco::b_interval_eval_type::cache

Definition at line 51 of file bint_evaluator.h.

10.12.2.2 b_interval coco::b_interval_eval_type::d

Definition at line 54 of file bint_evaluator.h.

10.12.2.3 int coco::b_interval_eval_type::info

Definition at line 55 of file bint_evaluator.h.

10.12.2.4 const model* coco::b_interval_eval_type::mod

Definition at line 52 of file bint_evaluator.h.

10.12.2.5 unsigned int coco::b_interval_eval_type::n

Definition at line 57 of file bint_evaluator.h.

10.12.2.6 void* coco::b_interval_eval_type::p

Definition at line 53 of file bint_evaluator.h.

10.12.2.7 b_interval coco::b_interval_eval_type::r

Definition at line 56 of file bint_evaluator.h.

10.12.2.8 const std::vector<b_interval>* coco::b_interval_eval_type::x

Definition at line 50 of file bint_evaluator.h.

The documentation for this struct was generated from the following file:

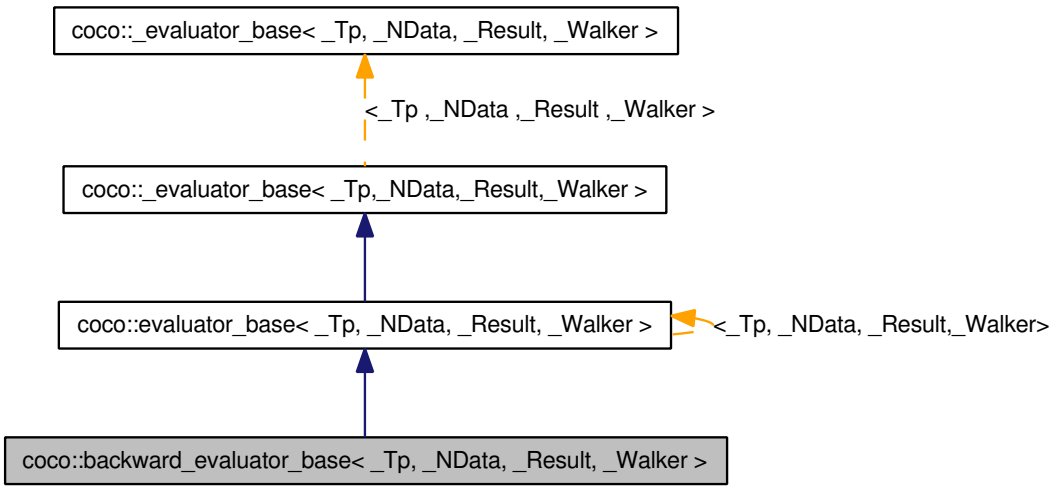
- bint_evaluator.h

10.13 coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker > Class Template Reference

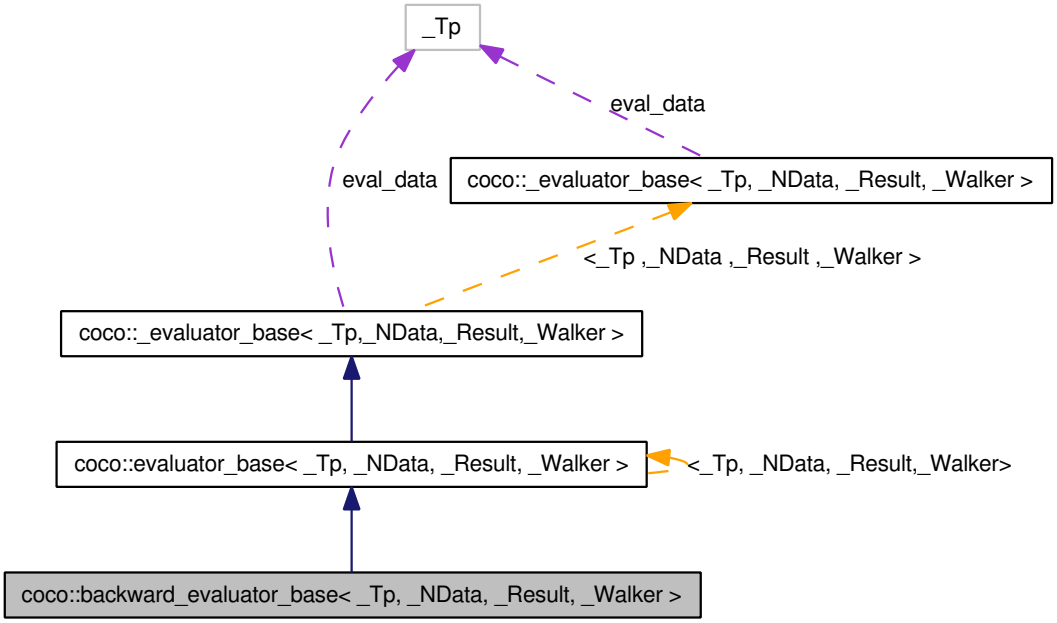
Base class of all (non-caching) backward evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >:



Collaboration diagram for coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >:



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

- 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)
- 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

10.13.1 Detailed Description

template<class _Tp, class _NData, class _Result, class _Walker> class coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >

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.

Definition at line 588 of file evaluator.h.

10.13.2 Member Typedef Documentation

10.13.2.1 template<class _Tp , class _NData , class _Result , class _Walker > typedef _Base::const_walker coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 601 of file evaluator.h.

10.13.2.2 typedef _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.13.2.3 template<class _Tp , class _NData , class _Result , class _Walker > typedef _Base::node_data_type coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 597 of file evaluator.h.

10.13.2.4 template<class _Tp , class _NData , class _Result , class _Walker > typedef _Base::return_value coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::return_value

This type is the result type of the evaluator.

Reimplemented from coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 599 of file evaluator.h.

10.13.3 Constructor & Destructor Documentation

10.13.3.1 template<class _Tp , class _NData , class _Result , class _Walker > coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::backward_evaluator_base () [inline]

Standard Constructor

Definition at line 606 of file evaluator.h.

10.13.3.2 template<class _Tp , class _NData , class _Result , class _Walker > coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::backward_evaluator_base (const _Tp & *__x*) [inline]

Constructor

Definition at line 608 of file evaluator.h.

10.13.3.3 template<class _Tp , class _NData , class _Result , class _Walker > coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::backward_evaluator_base (const _Self & *__x*) [inline]

Standard Copy Constructor

Definition at line 610 of file evaluator.h.

10.13.3.4 template<class _Tp , class _NData , class _Result , class _Walker > virtual coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::~backward_evaluator_base () [inline, virtual]

Standard Destructor

Definition at line 612 of file evaluator.h.

10.13.4 Member Function Documentation

10.13.4.1 template<class _Tp , class _NData , class _Result , class _Walker > virtual int coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.2 template<class _Tp , class _NData , class _Result , class _Walker > virtual return_value coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.3 template<class _Tp , class _NData , class _Result , class _Walker > virtual void coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.4 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 359 of file evaluator.h.

10.13.4.5 template<class _Tp , class _NData , class _Result , class _Walker > int coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.6 template<class _Tp , class _NData , class _Result , class _Walker > virtual void coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.7 template<class _Tp , class _NData , class _Result , class _Walker > virtual void coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Definition at line 654 of file evaluator.h.

10.13.4.8 virtual void coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 364 of file evaluator.h.

10.13.4.9 template<class _Tp , class _NData , class _Result , class _Walker > void coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.10 virtual int coco::evaluator_base< _Tp , _NData , _Result , _Walker >::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 401 of file evaluator.h.

10.13.4.11 template<class _Tp , class _NData , class _Result , class _Walker > int coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.12 virtual const_walker coco::evaluator_base< _Tp , _NData , _Result , _Walker >::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 405 of file evaluator.h.

10.13.4.13 template<class _Tp , class _NData , class _Result , class _Walker > virtual int coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.14 template<class _Tp , class _NData , class _Result , class _Walker > virtual int coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.15 template<class _Tp , class _NData , class _Result , class _Walker > return_value coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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. It is translated to a call to calculate_value with parameter false.

Reimplemented from coco::_evaluator_base< _Tp,_NData,_Result,_Walker >.

Definition at line 640 of file evaluator.h.

10.13.4.16 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 348 of file evaluator.h.

10.13.4.17 template<class _Tp , class _NData , class _Result , class _Walker > int coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.18 template<class _Tp , class _NData , class _Result , class _Walker > void coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.13.4.19 template<class _Tp , class _NData , class _Result , class _Walker > return_value coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< _Tp,_NData,_Result,_Walker >.

Definition at line 645 of file evaluator.h.

10.13.5 Member Data Documentation

10.13.5.1 _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::eval_data [protected, inherited]

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:

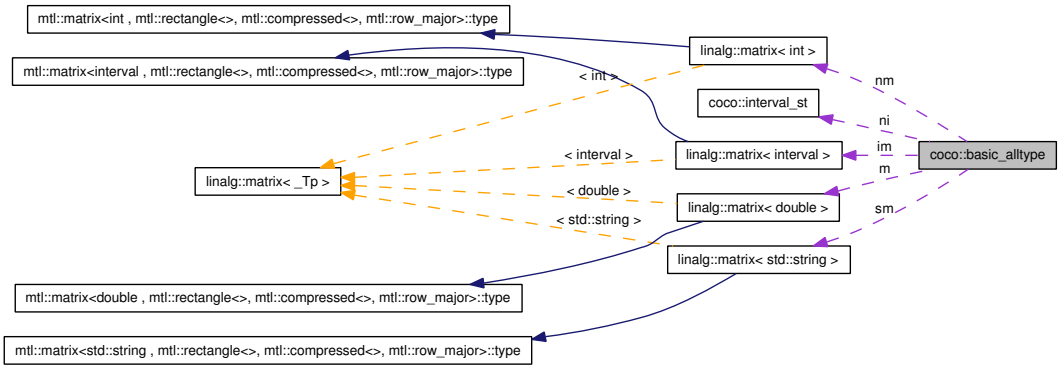
- evaluator.h

10.14 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 (const char *__cp)
- basic_alltype (const std::string &__x)
- 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 linalg::matrix< double > &__x)
- basic_alltype (const linalg::matrix< int > &__x)
- basic_alltype (const linalg::matrix< interval > &__x)
- basic_alltype (const linalg::matrix< std::string > &__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= (const std::string &__x)
- basic_alltype & operator= (const char *__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 linalg::matrix< double > &__x)
- basic_alltype & operator= (const linalg::matrix< int > &__x)
- basic_alltype & operator= (const linalg::matrix< interval > &__x)
- basic_alltype & operator= (const linalg::matrix< std::string > &__x)
- basic_alltype & operator= (const basic_alltype &__a)
- basic_alltype & clear ()
- basic_alltype & set_m (linalg::matrix< double > *__m)
- basic_alltype & set_nm (linalg::matrix< int > *__m)
- basic_alltype & set_im (linalg::matrix< interval > *__m)
- basic_alltype & set_sm (linalg::matrix< std::string > *__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
- std::string & s () const BASIC_ALLTYPE_THROW
- std::vector< bool > & b () const BASIC_ALLTYPE_THROW
- std::vector< int > & n () const BASIC_ALLTYPE_THROW
- std::vector< unsigned int > & u () const BASIC_ALLTYPE_THROW
- std::vector< double > & d () const BASIC_ALLTYPE_THROW
- std::vector< interval > & i () const BASIC_ALLTYPE_THROW
- std::vector< std::string > & sv () const BASIC_ALLTYPE_THROW
- linalg::matrix< double > & m () const BASIC_ALLTYPE_THROW
- linalg::matrix< int > & nm () const BASIC_ALLTYPE_THROW
- linalg::matrix< interval > & im () const BASIC_ALLTYPE_THROW
- linalg::matrix< std::string > & sm () const BASIC_ALLTYPE_THROW
- bool is_allocated () const
- bool is_scalar () const
- bool is_vector () const
- bool is_empty_vector () const
- bool is_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

10.14.1 Detailed Description

This class is the base of the data handling in the API. It can hold values of 16 different types: bool, int, unsigned int, double, interval, string, vector<bool>, vector<int>, vector<unsigned int>, vector<double>, vector<interval>, vector<string>, matrix<double>, matrix<int>, matrix<interval>, matrix<string>. All of these types can be stored and retrieved conveniently.

Definition at line 133 of file basic_alltype.h.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 coco::basic_alltype::basic_alltype () [inline]

Standard Constructor yielding an empty basic_alltype

Definition at line 274 of file basic_alltype.h.

10.14.2.2 coco::basic_alltype::basic_alltype (bool __x) [inline]

Constructor storing the bool __x

Definition at line 276 of file basic_alltype.h.

10.14.2.3 coco::basic_alltype::basic_alltype (int __x) [inline]

Constructor storing the int __x

Definition at line 279 of file basic_alltype.h.

10.14.2.4 coco::basic_alltype::basic_alltype (unsigned int __x) [inline]

Constructor storing the unsigned int __x

Definition at line 282 of file basic_alltype.h.

10.14.2.5 coco::basic_alltype::basic_alltype (double __x) [inline]

Constructor storing the double __x

Definition at line 285 of file basic_alltype.h.

10.14.2.6 coco::basic_alltype::basic_alltype (interval __x) [inline]

Constructor storing the interval __x

Definition at line 288 of file basic_alltype.h.

10.14.2.7 coco::basic_alltype::basic_alltype (const char * __cp) [inline]

Constructor storing the string __cp

Definition at line 291 of file basic_alltype.h.

10.14.2.8 coco::basic_alltype::basic_alltype (const std::string & __x) [inline]

Constructor storing the string __x

Definition at line 294 of file basic_alltype.h.

10.14.2.9 coco::basic_alltype::basic_alltype (const std::vector< bool > & __x) [inline]

Constructor storing the vector<bool> __x

Definition at line 297 of file basic_alltype.h.

10.14.2.10 coco::basic_alltype::basic_alltype (const std::vector< int > & __x) [inline]

Constructor storing the vector<int> __x

Definition at line 300 of file basic_alltype.h.

10.14.2.11 coco::basic_alltype::basic_alltype (const std::vector< unsigned int > & __x) [inline]

Constructor storing the vector<unsigned int> __x

Definition at line 303 of file basic_alltype.h.

10.14.2.12 coco::basic_alltype::basic_alltype (const std::vector< double > & __x) [inline]

Constructor storing the vector<double> __x

Definition at line 306 of file basic_alltype.h.

10.14.2.13 coco::basic_alltype::basic_alltype (const std::vector< interval > & __x) [inline]

Constructor storing the vector<interval> __x

Definition at line 310 of file basic_alltype.h.

10.14.2.14 coco::basic_alltype::basic_alltype (const std::vector< std::string > & __x) [inline]

Constructor storing the vector<string> __x

Definition at line 314 of file basic_alltype.h.

10.14.2.15 coco::basic_alltype::basic_alltype (const linalg::matrix< double > & __x) [inline]

Constructor storing the matrix<double> __x

Definition at line 318 of file basic_alltype.h.

10.14.2.16 coco::basic_alltype::basic_alltype (const linalg::matrix< int > & __x) [inline]

Constructor storing the matrix<int> __x

Definition at line 322 of file basic_alltype.h.

10.14.2.17 coco::basic_alltype::basic_alltype (const linalg::matrix< interval > & __x) [inline]

Constructor storing the matrix<interval> __x

Definition at line 326 of file basic_alltype.h.

10.14.2.18 coco::basic_alltype::basic_alltype (const linalg::matrix< std::string > & __x) [inline]

Constructor storing the matrix<string> __x

Definition at line 330 of file basic_alltype.h.

10.14.2.19 coco::basic_alltype::~basic_alltype () [inline]

Standard Destructor

Definition at line 335 of file basic_alltype.h.

10.14.2.20 coco::basic_alltype::basic_alltype (const basic_alltype & __a) [inline]

Standard Copy Constructor

Definition at line 339 of file basic_alltype.h.

10.14.3 Member Function Documentation

10.14.3.1 std::vector<bool>& coco::basic_alltype::b () const [inline]

retrieve a vector<bool> from the basic_alltype

Definition at line 549 of file basic_alltype.h.

10.14.3.2 basic_alltype& coco::basic_alltype::clear () [inline]

Reset the basic_alltype to an empty value

Definition at line 465 of file basic_alltype.h.

10.14.3.3 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 649 of file basic_alltype.h.

10.14.3.4 std::vector<double>& coco::basic_alltype::d () const [inline]

retrieve a vector<double> from the basic_alltype

Definition at line 597 of file basic_alltype.h.

10.14.3.5 bool coco::basic_alltype::empty () const [inline]

Check, whether the basic_alltype is empty.

Definition at line 646 of file basic_alltype.h.

10.14.3.6 std::vector<interval>& coco::basic_alltype::i () const [inline]

retrieve a vector<interval> from the basic_alltype

Definition at line 601 of file basic_alltype.h.

10.14.3.7 linalg::matrix<interval>& coco::basic_alltype::im () const [inline]

retrieve a matrix<interval> from the basic_alltype

Definition at line 615 of file basic_alltype.h.

10.14.3.8 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 624 of file basic_alltype.h.

10.14.3.9 bool coco::basic_alltype::is_empty_vector () const [inline]

Check, whether the basic_alltype contains a size 0 vector type.

Definition at line 633 of file basic_alltype.h.

10.14.3.10 bool coco::basic_alltype::is_matrix () const [inline]

Check, whether the basic_alltype contains a matrix type.

Definition at line 644 of file basic_alltype.h.

10.14.3.11 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 627 of file basic_alltype.h.

10.14.3.12 bool coco::basic_alltype::is_vector () const [inline]

Check, whether the basic_alltype contains a vector type.

Definition at line 630 of file basic_alltype.h.

10.14.3.13 linalg::matrix<double>& coco::basic_alltype::m () const [inline]

retrieve a matrix<double> from the basic_alltype

Definition at line 609 of file basic_alltype.h.

10.14.3.14 std::vector<int>& coco::basic_alltype::n () const [inline]

retrieve a vector<int> from the basic_alltype

Definition at line 553 of file basic_alltype.h.

10.14.3.15 bool coco::basic_alltype::nb () const [inline]

retrieve a bool from the basic_alltype

Definition at line 519 of file basic_alltype.h.

10.14.3.16 double coco::basic_alltype::nd () const [inline]

retrieve a double from the basic_alltype

Definition at line 540 of file basic_alltype.h.

10.14.3.17 interval coco::basic_alltype::ni () const [inline]

retrieve an interval from the basic_alltype

Definition at line 543 of file basic_alltype.h.

10.14.3.18 linalg::matrix<int>& coco::basic_alltype::nm () const [inline]

retrieve a matrix<int> from the basic_alltype

Definition at line 612 of file basic_alltype.h.

10.14.3.19 int coco::basic_alltype::nn () const [inline]

retrieve an int from the basic_alltype

Definition at line 522 of file basic_alltype.h.

10.14.3.20 unsigned int coco::basic_alltype::nu () const [inline]

retrieve an unsigned int from the basic_alltype

Definition at line 531 of file basic_alltype.h.

10.14.3.21 bool coco::basic_alltype::operator!= (const basic_alltype & *b*) const

Standard comparison operator

Definition at line 173 of file basic_alltype.cc.

10.14.3.22 basic_alltype& coco::basic_alltype::operator= (const basic_alltype & __a) [inline]

Standard Assignment Operator

Definition at line 457 of file basic_alltype.h.

10.14.3.23 basic_alltype& coco::basic_alltype::operator= (const linalg::matrix< std::string > & __x) [inline]

Assignment Operator, assigning the matrix<string> value __x

Definition at line 450 of file basic_alltype.h.

10.14.3.24 basic_alltype& coco::basic_alltype::operator= (const linalg::matrix< interval > & __x) [inline]

Assignment Operator, assigning the matrix<interval> value __x

Definition at line 443 of file basic_alltype.h.

10.14.3.25 basic_alltype& coco::basic_alltype::operator= (const linalg::matrix< int > & __x) [inline]

Assignment Operator, assigning the matrix<int> value __x

Definition at line 436 of file basic_alltype.h.

10.14.3.26 basic_alltype& coco::basic_alltype::operator= (const linalg::matrix< double > & __x) [inline]

Assignment Operator, assigning the matrix<double> value __x

Definition at line 429 of file basic_alltype.h.

10.14.3.27 basic_alltype& coco::basic_alltype::operator= (const std::vector< std::string > & __x) [inline]

Assignment Operator, assigning the vector<string> value __x

Definition at line 422 of file basic_alltype.h.

10.14.3.28 basic_alltype& coco::basic_alltype::operator= (const std::vector< interval > & __x) [inline]

Assignment Operator, assigning the vector<interval> value __x

Definition at line 415 of file basic_alltype.h.

10.14.3.29 basic_alltype& coco::basic_alltype::operator= (const std::vector< double > & __x) [inline]

Assignment Operator, assigning the vector<double> value __x

Definition at line 408 of file basic_alltype.h.

10.14.3.30 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 401 of file basic_alltype.h.

10.14.3.31 basic_alltype& coco::basic_alltype::operator= (const std::vector< int > & __x) [inline]

Assignment Operator, assigning the vector<int> value __x

Definition at line 394 of file basic_alltype.h.

10.14.3.32 basic_alltype& coco::basic_alltype::operator= (const std::vector< bool > & __x) [inline]

Assignment Operator, assigning the vector<bool> value __x

Definition at line 387 of file basic_alltype.h.

10.14.3.33 basic_alltype& coco::basic_alltype::operator= (const char * __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 380 of file basic_alltype.h.

10.14.3.34 basic_alltype& coco::basic_alltype::operator= (const std::string & __x) [inline]

Assignment Operator, assigning the string value __x

Definition at line 373 of file basic_alltype.h.

10.14.3.35 basic_alltype& coco::basic_alltype::operator= (interval __x) [inline]

Assignment Operator, assigning the interval value __x

Definition at line 367 of file basic_alltype.h.

10.14.3.36 basic_alltype& coco::basic_alltype::operator= (double __x) [inline]

Assignment Operator, assigning the double value __x

Definition at line 361 of file basic_alltype.h.

10.14.3.37 basic_alltype& coco::basic_alltype::operator= (unsigned int __x) [inline]

Assignment Operator, assigning the unsigned int value __x

Definition at line 355 of file basic_alltype.h.

10.14.3.38 basic_alltype& coco::basic_alltype::operator= (int __x) [inline]

Assignment Operator, assigning the int value __x

Definition at line 349 of file basic_alltype.h.

10.14.3.39 basic_alltype& coco::basic_alltype::operator= (bool __x) [inline]

Assignment Operator, assigning the bool value __x

Definition at line 343 of file basic_alltype.h.

10.14.3.40 bool coco::basic_alltype::operator== (const basic_alltype & *b*) const

Standard comparison operator

Definition at line 111 of file basic_alltype.cc.

10.14.3.41 std::string& coco::basic_alltype::s () const [inline]

retrieve a string from the basic_alltype

Definition at line 546 of file basic_alltype.h.

10.14.3.42 basic_alltype& coco::basic_alltype::set_im (linalg::matrix< interval > * *__m*) [inline]

This method assigns a matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new matrix<interval>.

Definition at line 498 of file basic_alltype.h.

10.14.3.43 basic_alltype& coco::basic_alltype::set_m (linalg::matrix< double > * *__m*) [inline]

This method assigns a matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new matrix<double>.

Definition at line 476 of file basic_alltype.h.

10.14.3.44 basic_alltype& coco::basic_alltype::set_nm (linalg::matrix< int > * *__m*) [inline]

This method assigns a matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new matrix<int>.

Definition at line 487 of file basic_alltype.h.

10.14.3.45 basic_alltype& coco::basic_alltype::set_sm (linalg::matrix< std::string > * *__m*) [inline]

This method assigns a matrix<double>* to the basic alltype. This method must not be used unless __m was allocated by new matrix<string>.

Definition at line 509 of file basic_alltype.h.

10.14.3.46 linalg::matrix<std::string>& coco::basic_alltype::sm () const [inline]

retrieve a matrix<string> from the basic_alltype

Definition at line 618 of file basic_alltype.h.

10.14.3.47 std::vector<std::string>& coco::basic_alltype::sv () const [inline]

retrieve a vector<string> from the basic_alltype

Definition at line 605 of file basic_alltype.h.

10.14.3.48 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 654 of file basic_alltype.h.

10.14.3.49 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 651 of file basic_alltype.h.

10.14.3.50 std::vector<unsigned int>& coco::basic_alltype::u () const [inline]

retrieve a vector<unsigned int> from the basic_alltype

Definition at line 575 of file basic_alltype.h.

10.14.4 Member Data Documentation

10.14.4.1 std::vector<bool>* coco::basic_alltype::b

Definition at line 147 of file basic_alltype.h.

10.14.4.2 std::vector<double>* coco::basic_alltype::d

Definition at line 150 of file basic_alltype.h.

10.14.4.3 std::vector<interval>* coco::basic_alltype::i

Definition at line 151 of file basic_alltype.h.

10.14.4.4 linalg::matrix<interval>* coco::basic_alltype::im

Definition at line 155 of file basic_alltype.h.

10.14.4.5 linalg::matrix<double>* coco::basic_alltype::m

Definition at line 153 of file basic_alltype.h.

10.14.4.6 std::vector<int>* coco::basic_alltype::n

Definition at line 148 of file basic_alltype.h.

10.14.4.7 bool coco::basic_alltype::nb

Definition at line 141 of file basic_alltype.h.

10.14.4.8 double coco::basic_alltype::nd

Definition at line 144 of file basic_alltype.h.

10.14.4.9 interval_st coco::basic_alltype::ni

Definition at line 145 of file basic_alltype.h.

10.14.4.10 linalg::matrix<int>* coco::basic_alltype::nm

Definition at line 154 of file basic_alltype.h.

10.14.4.11 int coco::basic_alltype::nn

Definition at line 142 of file basic_alltype.h.

10.14.4.12 unsigned int coco::basic_alltype::nu

Definition at line 143 of file basic_alltype.h.

10.14.4.13 std::string* coco::basic_alltype::s

Definition at line 146 of file basic_alltype.h.

10.14.4.14 linalg::matrix<std::string>* coco::basic_alltype::sm

Definition at line 156 of file basic_alltype.h.

10.14.4.15 std::vector<std::string>* coco::basic_alltype::sv

Definition at line 152 of file basic_alltype.h.

10.14.4.16 std::vector<unsigned int>* coco::basic_alltype::u

Definition at line 149 of file basic_alltype.h.

The documentation for this class was generated from the following files:

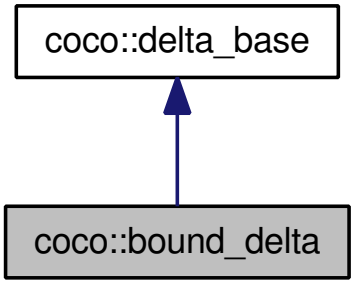
- basic_alltype.h
- basic_alltype.cc

10.15 coco::bound_delta Class Reference

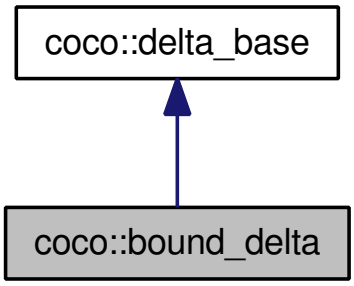
The bound delta class for changing the node bounds within a model.

`#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) 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)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Public Attributes

- std::vector< unsigned int > indices
- std::vector< interval > new_f_bounds

Protected Attributes

- std::string _action

10.15.1 Detailed Description

This class is used to specify information for changing the node bounds within a model.

Definition at line 98 of file bound_delta.h.

10.15.2 Constructor & Destructor Documentation

10.15.2.1 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.

10.15.2.2 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.

10.15.2.3 coco::bound_delta::bound_delta (const bound_delta & *__d*) [inline]

Standard Copy Constructor

Definition at line 121 of file bound_delta.h.

10.15.3 Member Function Documentation

10.15.3.1 bool coco::bound_delta::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_did*) const [virtual]

Apply the delta with delta_id _d to work node _x, hereby changing the bounds as requested.

Reimplemented from coco::delta_base.

Definition at line 33 of file bound_delta.cc.

10.15.3.2 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*) 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.

10.15.3.3 virtual void 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.

Reimplemented in coco::table_delta.

Definition at line 189 of file api_deltabase.h.

10.15.3.4 void coco::bound_delta::destroy_copy (delta_base * *__d*) const [inline]

Clone Destructor

Definition at line 132 of file bound_delta.h.

10.15.3.5 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.15.3.6 delta 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 api_deltabase.h.

10.15.3.7 bound_delta* coco::bound_delta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::delta_base.

Definition at line 130 of file bound_delta.h.

10.15.3.8 bool coco::bound_delta::operator!= (const bound_delta & _c) const [inline]

Definition at line 150 of file bound_delta.h.

10.15.3.9 bool coco::bound_delta::operator!= (const delta_base & _c) const [inline]

Definition at line 142 of file bound_delta.h.

10.15.3.10 bool coco::bound_delta::operator== (const bound_delta & _c) const [inline]

Comparison operators

Definition at line 147 of file bound_delta.h.

10.15.3.11 bool coco::bound_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Definition at line 139 of file bound_delta.h.

10.15.3.12 virtual void 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 api_deltabase.h.

10.15.4 Member Data Documentation

10.15.4.1 std::string coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file api_deltabase.h.

10.15.4.2 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.

10.15.4.3 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:

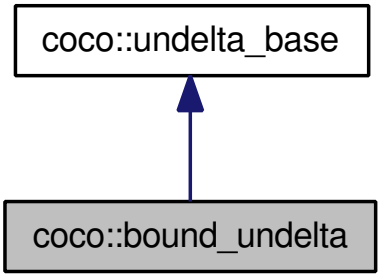
- bound_delta.h
- bound_delta.cc

10.16 coco::bound_undelta Class Reference

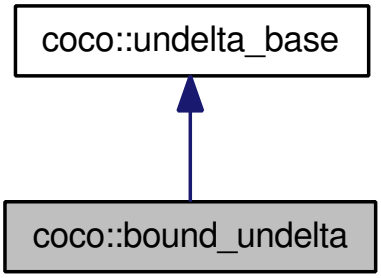
The bound undelta class for undoing changes to the node bounds in a model.

`#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 ()
- 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

10.16.1 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.

Definition at line 41 of file bound_delta.h.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 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.

10.16.2.2 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.

10.16.2.3 coco::bound_undelta::bound_undelta (const bound_undelta & *__d*) [inline]

Standard Copy Constructor

Definition at line 70 of file bound_delta.h.

10.16.3 Member Function Documentation

10.16.3.1 void coco::bound_undelta::destroy_copy (undelta_base * *__d*) const [inline]

Clone Destructor

Definition at line 84 of file bound_delta.h.

10.16.3.2 undelta coco::undelta_base::make_undelta () [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 283 of file api_deltabase.h.

10.16.3.3 bound_undelta* coco::bound_undelta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::undelta_base.

Definition at line 82 of file bound_delta.h.

10.16.3.4 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::undelta_base.

Definition at line 74 of file bound_delta.cc.

10.16.3.5 bool 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.

10.16.4 Friends And Related Function Documentation

10.16.4.1 friend class bound_delta [friend]

Definition at line 89 of file bound_delta.h.

10.16.5 Member Data Documentation

10.16.5.1 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.

10.16.5.2 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.

10.16.5.3 double coco::bound_undelta::old_gain

the gain in the work_node before the application

Definition at line 51 of file bound_delta.h.

10.16.5.4 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:

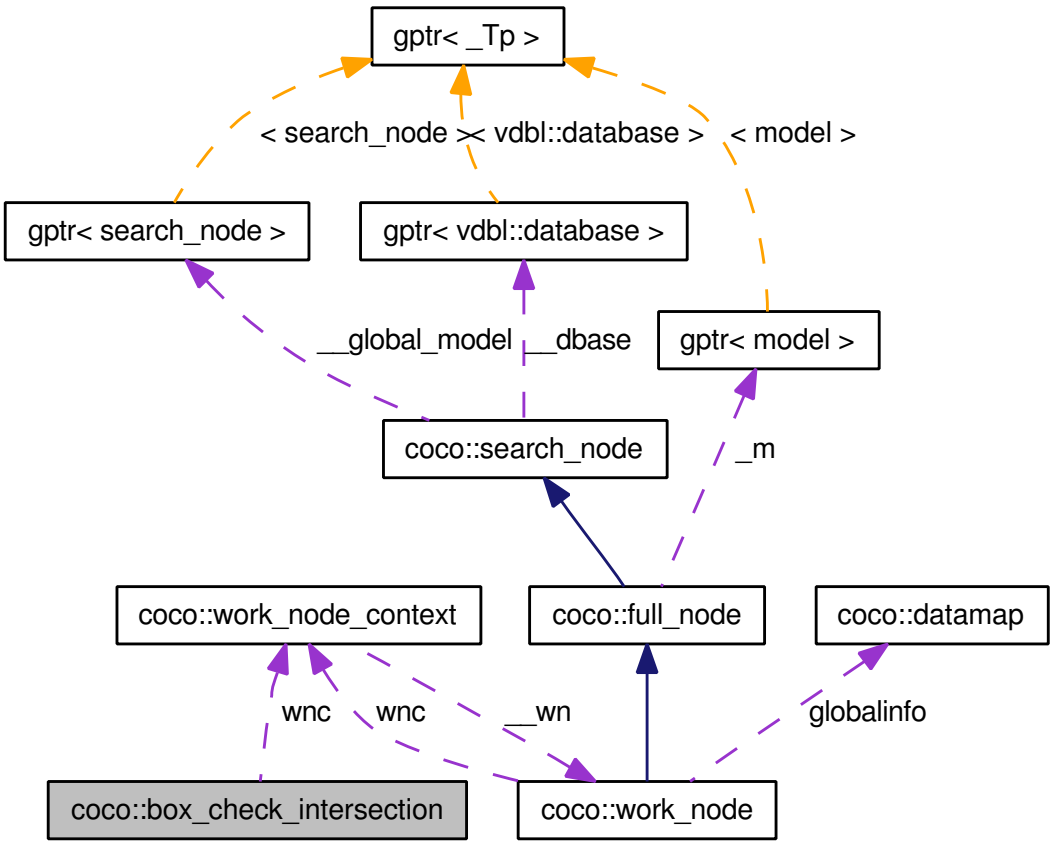
- bound_delta.h
- bound_delta.cc

10.17 coco::box_check_intersection Class Reference

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

```
#include <dbtools.h>
```

Collaboration diagram for coco::box_check_intersection:



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)

10.17.1 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.

Definition at line 95 of file dbtools.h.

10.17.2 Member Typedef Documentation

10.17.2.1 typedef work_node_context coco::box_check_intersection::context

The evaluation context type

Definition at line 99 of file dbtools.h.

10.17.2.2 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.

10.17.3 Constructor & Destructor Documentation

10.17.3.1 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.

10.17.3.2 coco::box_check_intersection::box_check_intersection (const box_check_intersection & *_i*) [inline]

Standard Copy Constructor

Definition at line 119 of file dbtools.h.

10.17.3.3 virtual coco::box_check_intersection::~box_check_intersection () [inline, virtual]

Standard Destructor

Definition at line 122 of file dbtools.h.

10.17.4 Member Function Documentation

10.17.4.1 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.

10.17.4.2 bool coco::box_check_intersection::operator() () const

Evaluation operator, which computes the value of the stored procedure

Definition at line 75 of file dbtools.cc.

10.17.4.3 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 files:

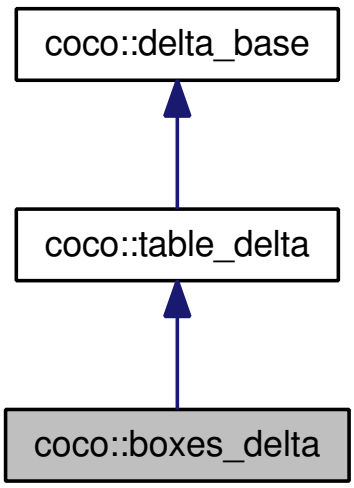
- dbtools.h
- dbtools.cc

10.18 coco::boxes_delta Class Reference

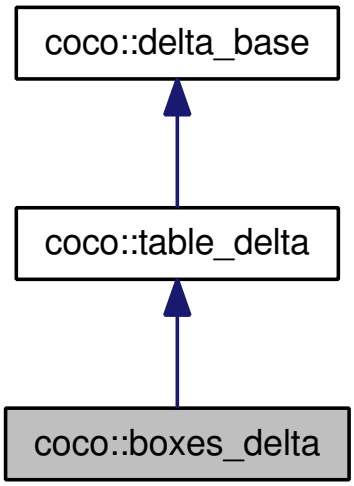
A delta class which adds new boxes to the search database.

`#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) 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)
- void convert (work_node &_x, delta_base *&_u)
- bool operator== (const table_delta &_c) const
- bool operator!= (const table_delta &_c) const
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Protected Attributes

- std::string _action

10.18.1 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.

Definition at line 44 of file boxes_delta.h.

10.18.2 Member Typedef Documentation

10.18.2.1 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.

10.18.2.2 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.

10.18.3 Constructor & Destructor Documentation

10.18.3.1 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.

10.18.3.2 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.

10.18.3.3 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.

10.18.3.4 coco::boxes_delta::boxes_delta (const boxes_delta & *__d*) `[inline]`

Standard Copy Constructor

Definition at line 89 of file boxes_delta.h.

10.18.4 Member Function Documentation

10.18.4.1 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.

10.18.4.2 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.

10.18.4.3 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.

10.18.4.4 bool coco::boxes_delta::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_did*) 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.

10.18.4.5 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*) 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.

10.18.4.6 void coco::table_delta::convert (work_node & *_x*, delta_base *& *_u*) [virtual, 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.

Reimplemented from coco::delta_base.

Definition at line 42 of file table_delta.cc.

10.18.4.7 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 43 of file boxes_delta.cc.

10.18.4.8 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.

10.18.4.9 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.18.4.10 delta 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 api_deltabase.h.

10.18.4.11 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.

10.18.4.12 bool coco::table_delta::operator!= (const table_delta & *_c*) const [inline, inherited]

Definition at line 143 of file table_delta.h.

10.18.4.13 bool coco::boxes_delta::operator!= (const boxes_delta & *_c*) const [inline]

Definition at line 119 of file boxes_delta.h.

10.18.4.14 bool coco::boxes_delta::operator!= (const delta_base & *_c*) const [inline]

Reimplemented from coco::table_delta.

Definition at line 112 of file boxes_delta.h.

10.18.4.15 bool coco::table_delta::operator== (const table_delta & *_c*) const [inline, inherited]

Comparison operators

Definition at line 142 of file table_delta.h.

10.18.4.16 bool coco::boxes_delta::operator== (const boxes_delta & *_c*) const [inline]

Comparison operators

Definition at line 117 of file boxes_delta.h.

10.18.4.17 bool coco::boxes_delta::operator== (const delta_base & *_c*) const [inline]

Comparison operators

Reimplemented from coco::table_delta.

Definition at line 109 of file boxes_delta.h.

10.18.4.18 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.

10.18.4.19 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.

10.18.4.20 virtual void 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 api_deltabase.h.

10.18.5 Member Data Documentation

10.18.5.1 std::string coco::delta_base::_action [protected, inherited]

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 files:

- boxes_delta.h
- boxes_delta.cc

10.19 linalg::c_matrix< _Tp > Class Template Reference

Sparse Matrix Template Class (column major).

`#include <linalg.h>`

Public Types

- typedef mtl::matrix< _Tp, mtl::rectangle<>, mtl::compressed<>, mtl::column_major >::type _Base
- typedef _Base::OneD Col

Public Member Functions

- c_matrix ()
- c_matrix (const _Self &__m)
- c_matrix (const size_t &n, const size_t &m)
- void setOutputFormat (const linalg_output_format &f)
- const linalg_output_format & getOutputFormat () const
- _Self & operator+= (const _Self &a)
- _Self & operator*= (_Tp a)
- bool operator== (const _Self &a) const
- bool operator!= (const _Self &a) const

Protected Attributes

- linalg_output_format output_format

Friends

- template<typename _Tq >
  std::ostream & std::operator<< (std::ostream &s, const c_matrix< _Tq > &A)

10.19.1 Detailed Description

template<class _Tp> class linalg::c_matrix< _Tp >

This class is the base class for all matrix storage/computation in the COCONUT environment. It is based on the MTL, but wraps the MTL classes to make changing the matrix library possible (and hopefully not too awkward. The matrix is general rectangular column-major.

Definition at line 439 of file linalg.h.

10.19.2 Member Typedef Documentation

10.19.2.1 template<class _Tp> typedef mtl::matrix<_Tp, mtl::rectangle<>, mtl::compressed<>, mtl::column_major>::type linalg::c_matrix< _Tp >::_Base

The base class

Definition at line 444 of file linalg.h.

10.19.2.2 template<class _Tp> typedef _Base::OneD linalg::c_matrix< _Tp >::Col

This is the row type of the matrix class.

Definition at line 446 of file linalg.h.

10.19.3 Constructor & Destructor Documentation

10.19.3.1 template<class _Tp> linalg::c_matrix< _Tp >::c_matrix () [inline]

The Standard Constructor

Definition at line 456 of file linalg.h.

10.19.3.2 template<class _Tp> linalg::c_matrix< _Tp >::c_matrix (const _Self & __m) [inline]

Definition at line 457 of file linalg.h.

10.19.3.3 template<class _Tp> linalg::c_matrix< _Tp >::c_matrix (const size_t & n, const size_t & m) [inline]

Definition at line 460 of file linalg.h.

10.19.4 Member Function Documentation

10.19.4.1 template<class _Tp> const linalg_output_format& linalg::c_matrix< _Tp >::getOutputFormat () const [inline]

This method returns the output format of the matrix.

Definition at line 467 of file linalg.h.

10.19.4.2 template<class _Tp> bool linalg::c_matrix< _Tp >::operator!= (const _Self & a) const

Standard comparison operator.

10.19.4.3 template<class _Tp> _Self& linalg::c_matrix< _Tp >::operator*= (_Tp a) [inline]

This operator multiplies this matrix by a.

Definition at line 500 of file linalg.h.

10.19.4.4 template<class _Tp> _Self& linalg::c_matrix< _Tp >::operator+= (const _Self & a) [inline]

This operator adds a to this matrix.

Definition at line 492 of file linalg.h.

10.19.4.5 template<class _Tp> bool linalg::c_matrix< _Tp >::operator== (const _Self & *a*) const

Standard comparison operator.

10.19.4.6 template<class _Tp> void linalg::c_matrix< _Tp >::setOutputFormat (const linalg_output_format & *f*) [inline]

This method sets the output format of the matrix.

Definition at line 464 of file linalg.h.

10.19.5 Friends And Related Function Documentation

10.19.5.1 template<class _Tp> template<typename _Tq > std::ostream& std::operator<< (std::ostream & *s*, const c_matrix< _Tq > & *A*) [friend]

10.19.6 Member Data Documentation

10.19.6.1 template<class _Tp> linalg_output_format linalg::c_matrix< _Tp >::output_format [protected]

This variable describes the output format for matrix output. Its possible values are linalg_output_sparse and linalg_output_full.

Definition at line 452 of file linalg.h.

The documentation for this class was generated from the following file:

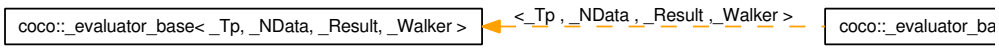
- linalg.h

10.20 coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker > Class Template Reference

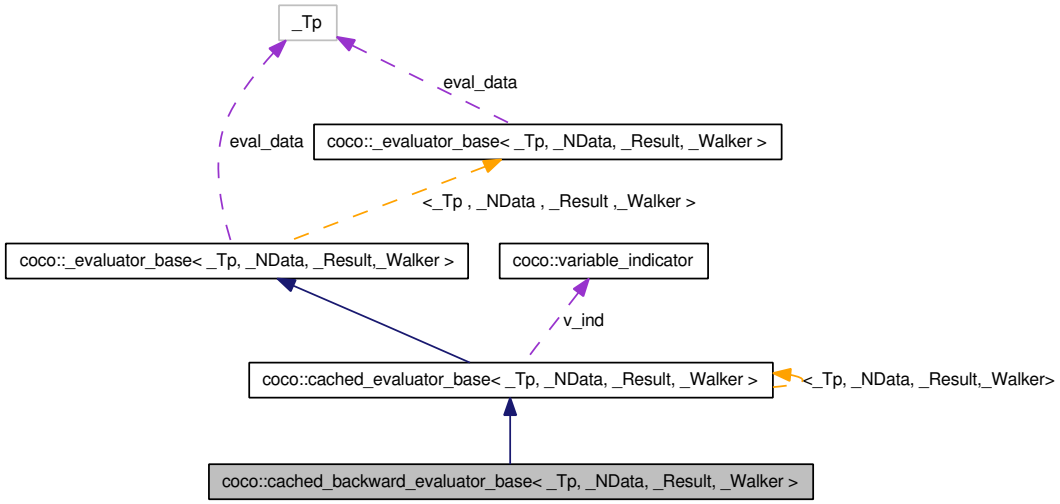
Base class of all caching backward evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >:



Collaboration diagram for coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >:



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

- 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)
- 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

10.20.1 Detailed Description

template<class _Tp, class _NData, class _Result, class _Walker> class coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >

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.

Definition at line 841 of file evaluator.h.

10.20.2 Member Typedef Documentation

10.20.2.1 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::const_walker coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 853 of file evaluator.h.

10.20.2.2 typedef _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.20.2.3 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::node_data_type coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 849 of file evaluator.h.

10.20.2.4 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::return_value coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::return_value

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 851 of file evaluator.h.

10.20.3 Member Function Documentation

10.20.3.1 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::der_eval, coco::der_eval, coco::hessBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, coco::islp_eval, and coco::model::model::lincoeff_visitor.

Definition at line 921 of file evaluator.h.

10.20.3.2 template<class _Tp, class _NData, class _Result, class _Walker> virtual return_value coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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::der_eval, coco::der_eval, coco::hessBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, coco::islp_eval, and coco::model::model::lincoeff_visitor.

Definition at line 956 of file evaluator.h.

10.20.3.3 template<class _Tp, class _NData, class _Result, class _Walker> virtual void coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::der_eval, coco::der_eval, coco::hessBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, and coco::islp_eval.

Definition at line 925 of file evaluator.h.

10.20.3.4 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 359 of file evaluator.h.

10.20.3.5 template<class _Tp, class _NData, class _Result, class _Walker> int coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.20.3.6 template<class _Tp, class _NData, class _Result, class _Walker> virtual void coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented in coco::der_eval, coco::der_eval, coco::hessBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, coco::islp_eval, and coco::model::model::lincoeff_visitor.

Definition at line 910 of file evaluator.h.

10.20.3.7 template<class _Tp, class _NData, class _Result, class _Walker> virtual bool coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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::der_eval, coco::der_eval, coco::hessBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, coco::islp_eval, and coco::model::model::lincoeff_visitor.

Definition at line 905 of file evaluator.h.

10.20.3.8 virtual void coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 364 of file evaluator.h.

10.20.3.9 template<class _Tp, class _NData, class _Result, class _Walker> void coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.20.3.10 virtual int coco::cached_evaluator_base< _Tp , _NData , _Result , _Walker >::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 446 of file evaluator.h.

10.20.3.11 template<class _Tp, class _NData, class _Result, class _Walker> int coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.20.3.12 template<class _Tp, class _NData, class _Result, class _Walker> virtual void coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::der_eval, coco::der_eval, coco::ider_eval, coco::islp_eval, and coco::model::model::lincoeff_visitor.

Definition at line 929 of file evaluator.h.

10.20.3.13 virtual const_walker coco::cached_evaluator_base< _Tp , _NData , _Result , _Walker >::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 450 of file evaluator.h.

10.20.3.14 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::der_eval, coco::der_eval, coco::hessBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, and coco::islp_eval.

Definition at line 951 of file evaluator.h.

10.20.3.15 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::der_eval, coco::der_eval, coco::hessBackwardEvaluator, coco::ider_eval, coco::ihessBackwardEvaluator, and coco::islp_eval.

Definition at line 940 of file evaluator.h.

10.20.3.16 template<class _Tp, class _NData, class _Result, class _Walker> return_value coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result,_Walker >.

Definition at line 891 of file evaluator.h.

10.20.3.17 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 348 of file evaluator.h.

10.20.3.18 template<class _Tp, class _NData, class _Result, class _Walker> int coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.20.3.19 template<class _Tp, class _NData, class _Result, class _Walker> void coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.20.3.20 template<class _Tp, class _NData, class _Result, class _Walker> return_value coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result,_Walker >.

Definition at line 896 of file evaluator.h.

10.20.4 Member Data Documentation

10.20.4.1 _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.20.4.2 const variable_indicator* coco::cached_evaluator_base< _Tp , _NData , _Result , _Walker >::v_ind [protected, inherited]

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:

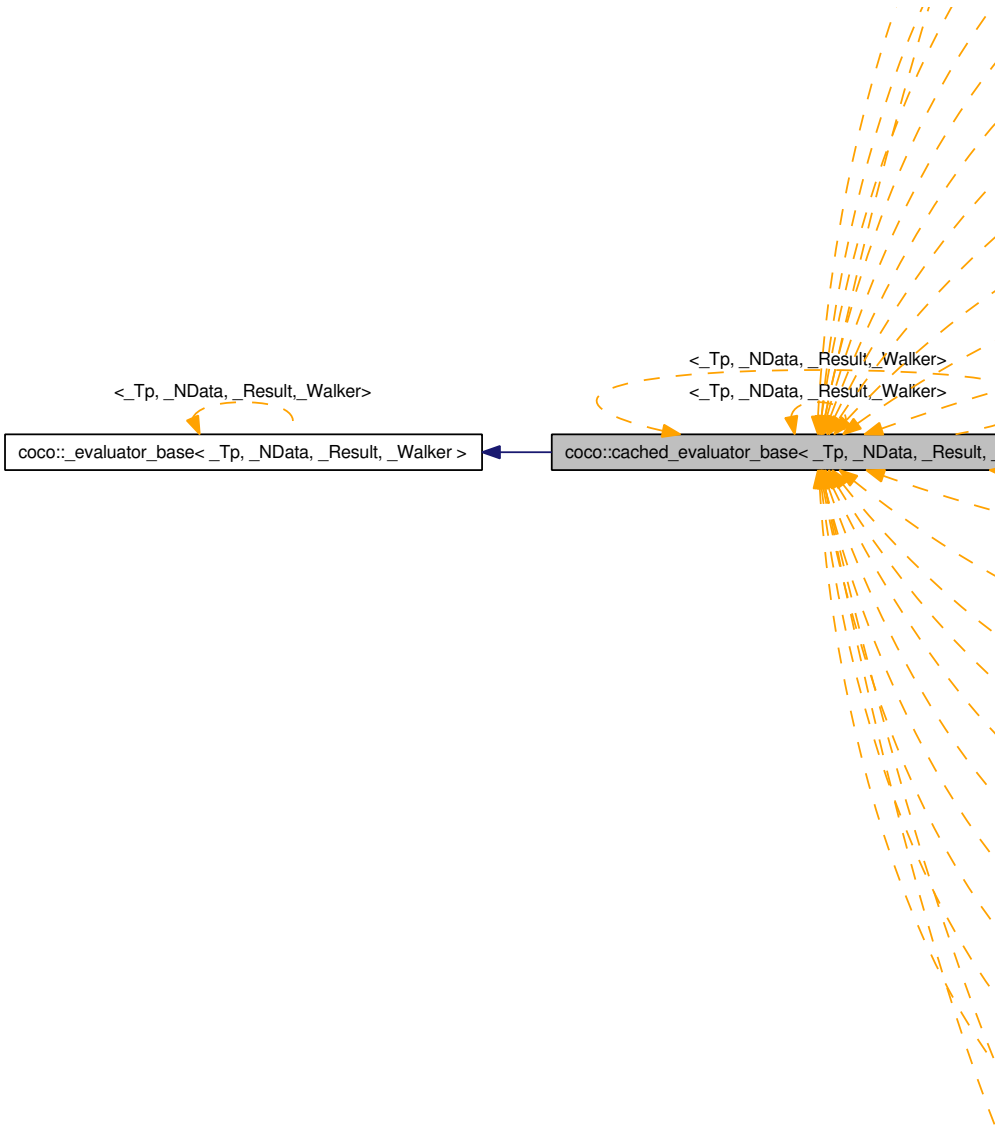
- evaluator.h

10.21 coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker > Class Template Reference

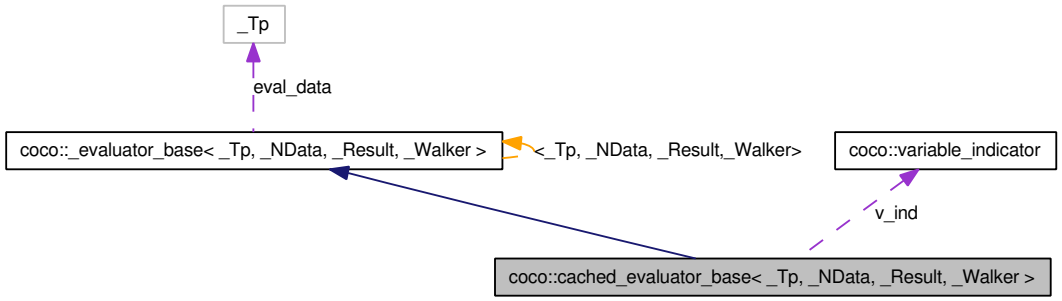
Base class of all caching evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >:



Collaboration diagram for coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >:



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

10.21.1 Detailed Description

template<class _Tp, class _NData, class _Result, class _Walker> class coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >

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.

Definition at line 417 of file evaluator.h.

10.21.2 Member Typedef Documentation

10.21.2.1 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::const_walker coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 435 of file evaluator.h.

10.21.2.2 typedef _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.21.2.3 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::node_data_type coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 431 of file evaluator.h.

10.21.2.4 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::return_value coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::return_value

This type is the result type of the evaluator.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >, coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 433 of file evaluator.h.

10.21.3 Constructor & Destructor Documentation

10.21.3.1 template<class _Tp, class _NData, class _Result, class _Walker> coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::cached_evaluator_base () [inline]

Standard Constructor

Definition at line 455 of file evaluator.h.

10.21.3.2 template<class _Tp, class _NData, class _Result, class _Walker> coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.21.3.3 template<class _Tp, class _NData, class _Result, class _Walker> coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::cached_evaluator_base (const _Self & *__x*) [inline]

Standard Copy Constructor

Definition at line 461 of file evaluator.h.

10.21.3.4 template<class _Tp, class _NData, class _Result, class _Walker> virtual coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::~cached_evaluator_base () [inline, virtual]

Standard Destructor

Definition at line 464 of file evaluator.h.

10.21.4 Member Function Documentation

10.21.4.1 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 359 of file evaluator.h.

10.21.4.2 virtual void coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 364 of file evaluator.h.

10.21.4.3 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >, coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< b_interval_eval_type, expression_node, b_interval, expression_const_walker >, coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >, coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >, coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >, coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >, coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >, coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >, coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,

expression_const_walker >, coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >, coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >, coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >, coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >, coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >, coco::cached_forward_evaluator_base< analyticd_eval_type, expression_node, analyticd, expression_const_walker >, coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >, coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >, and coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 446 of file evaluator.h.

10.21.4.4 template<class _Tp, class _NData, class _Result, class _Walker> virtual const_walker coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::func_d_eval, coco::der_eval, coco::func_d_eval, coco::der_eval, coco::dfunc_eval< _T, DN >, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::hessForwardEvaluator, coco::hessBackwardEvaluator, coco::func_id_eval, coco::ider_eval, coco::iderf_eval, coco::ihessForwardEvaluator, coco::ihessBackwardEvaluator, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::func_islp_eval, coco::islp_eval, and coco::model::model::lincoeff_visitor.

Definition at line 450 of file evaluator.h.

10.21.4.5 virtual return_value coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 336 of file evaluator.h.

10.21.4.6 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 348 of file evaluator.h.

10.21.4.7 virtual return_value coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 332 of file evaluator.h.

10.21.5 Member Data Documentation

10.21.5.1 _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.21.5.2 template<class _Tp, class _NData, class _Result, class _Walker> const variable_indicator* coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >::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:

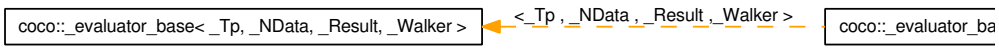
- evaluator.h

10.22 coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker > Class Template Reference

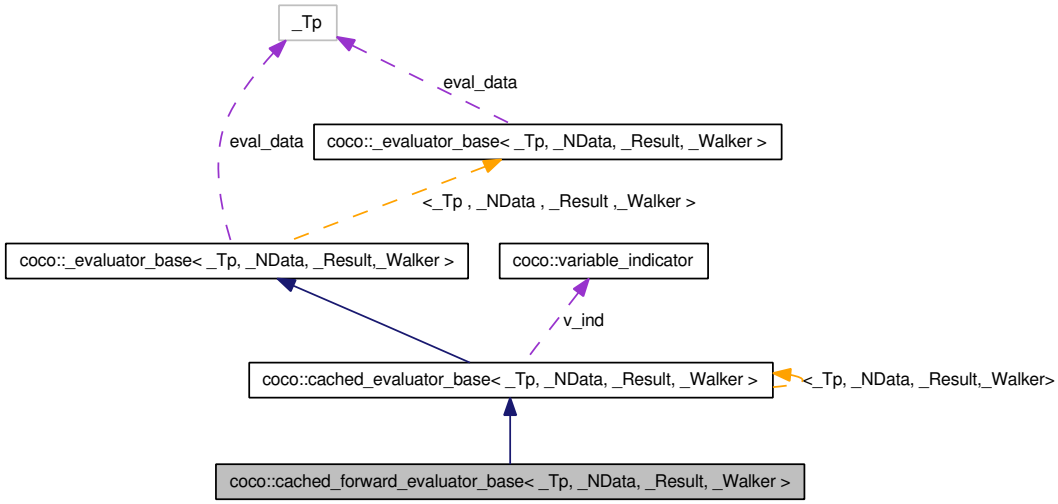
Base class of all (non-caching) forward evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >:



Collaboration diagram for coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >:



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

- 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)
- 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

10.22.1 Detailed Description

template<class _Tp, class _NData, class _Result, class _Walker> class coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >

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.

Definition at line 711 of file evaluator.h.

10.22.2 Member Typedef Documentation

10.22.2.1 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::const_walker coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 723 of file evaluator.h.

10.22.2.2 typedef _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.22.2.3 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::node_data_type coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 719 of file evaluator.h.

10.22.2.4 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::return_value coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::return_value

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 721 of file evaluator.h.

10.22.3 Member Function Documentation

10.22.3.1 template<class _Tp, class _NData, class _Result, class _Walker> virtual void coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::prep_d_eval, coco::func_d_eval, coco::prep_d_eval, coco::func_d_eval, coco::dfunc_eval< _T, DN >, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::hessPreparationEvaluator, coco::hessForwardEvaluator, coco::prep_id_eval, coco::func_id_eval, coco::iderf_eval, coco::ihessPreparationEvaluator, coco::ihessForwardEvaluator, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::prep_islp_eval, coco::func_islp_eval, and coco::model::model::detect_0chain_visitor.

Definition at line 796 of file evaluator.h.

10.22.3.2 template<class _Tp, class _NData, class _Result, class _Walker> virtual return_value coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::prep_d_eval, coco::func_d_eval, coco::prep_d_eval, coco::func_d_eval, coco::dfunc_eval< _T, DN >, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::hessPreparationEvaluator, coco::hessForwardEvaluator, coco::prep_id_eval, coco::func_id_eval, coco::iderf_eval, coco::ihessPreparationEvaluator, coco::ihessForwardEvaluator, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::prep_islp_eval, coco::func_islp_eval, and coco::model::model::detect_0chain_visitor.

Definition at line 830 of file evaluator.h.

10.22.3.3 template<class _Tp, class _NData, class _Result, class _Walker> virtual void coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.22.3.4 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 359 of file evaluator.h.

10.22.3.5 template<class _Tp, class _NData, class _Result, class _Walker> int coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.22.3.6 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::prep_d_eval, coco::func_d_eval, coco::prep_d_eval, coco::func_d_eval, coco::dfunc_eval< _T, DN >, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::hessPreparationEvaluator, coco::hessForwardEvaluator, coco::prep_id_eval, coco::func_id_eval, coco::iderf_eval, coco::ihessPreparationEvaluator, coco::ihessForwardEvaluator, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::prep_islp_eval, coco::func_islp_eval, and coco::model::model::detect_0chain_visitor.

Definition at line 792 of file evaluator.h.

10.22.3.7 template<class _Tp, class _NData, class _Result, class _Walker> virtual void coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented in coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::prep_d_eval, coco::func_d_eval, coco::prep_d_eval, coco::func_d_eval, coco::dfunc_eval< _T, DN >, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::hessPreparationEvaluator, coco::hessForwardEvaluator, coco::prep_id_eval, coco::func_id_eval, coco::iderf_eval, coco::ihessPreparationEvaluator, coco::ihessForwardEvaluator, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::prep_islp_eval, coco::func_islp_eval, and coco::model::model::detect_0chain_visitor.

Definition at line 781 of file evaluator.h.

10.22.3.8 template<class _Tp, class _NData, class _Result, class _Walker> virtual bool coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::prep_d_eval, coco::func_d_eval, coco::prep_d_eval, coco::func_d_eval, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::hessPreparationEvaluator, coco::hessForwardEvaluator, coco::prep_id_eval, coco::func_id_eval, coco::iderf_eval, coco::ihessPreparationEvaluator, coco::ihessForwardEvaluator, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::prep_islp_eval, coco::func_islp_eval, and coco::model::model::detect_0chain_visitor.

Definition at line 776 of file evaluator.h.

10.22.3.9 virtual void coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 364 of file evaluator.h.

10.22.3.10 template<class _Tp, class _NData, class _Result, class _Walker> void coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.22.3.11 virtual int coco::cached_evaluator_base< _Tp , _NData , _Result , _Walker >::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 446 of file evaluator.h.

10.22.3.12 template<class _Tp, class _NData, class _Result, class _Walker> int coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.22.3.13 template<class _Tp, class _NData, class _Result, class _Walker> virtual void coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::prep_d_eval, coco::func_d_eval, coco::prep_d_eval, coco::func_d_eval, coco::dfunc_eval< _T, DN >, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::hessPreparationEvaluator,

coco::hessForwardEvaluator, coco::prep_id_eval, coco::func_id_eval, coco::iderf_eval, coco::ihessPreparationEvaluator, coco::ihessForwardEvaluator, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::prep_islp_eval, coco::func_islp_eval, and coco::model::model::detect_0chain_visitor.

Definition at line 800 of file evaluator.h.

10.22.3.14 virtual const_walker coco::cached_evaluator_base< _Tp , _NData , _Result , _Walker >::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 450 of file evaluator.h.

10.22.3.15 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::func_d_eval, coco::func_d_eval, coco::dfunc_eval< _T, DN >, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::func_id_eval, coco::iderf_eval, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::func_islp_eval, and coco::model::model::detect_0chain_visitor.

Definition at line 826 of file evaluator.h.

10.22.3.16 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

Reimplemented in coco::analyticd_eval, coco::b_interval_eval, coco::cinterval_eval, coco::func_d_eval, coco::func_d_eval, coco::dfunc_eval< _T, DN >, coco::diffI_eval, coco::diffNumber_eval, coco::func_eval, coco::func_eval, coco::hessForwardEvaluator, coco::func_id_eval, coco::iderf_eval, coco::ihessForwardEvaluator, coco::infbound_eval, coco::interval_eval, coco::Islope_eval, coco::func_islp_eval, and coco::model::model::detect_0chain_visitor.

Definition at line 814 of file evaluator.h.

10.22.3.17 template<class _Tp, class _NData, class _Result, class _Walker> return_value coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result,_Walker >.

Definition at line 762 of file evaluator.h.

10.22.3.18 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 348 of file evaluator.h.

10.22.3.19 template<class _Tp, class _NData, class _Result, class _Walker> int coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.22.3.20 template<class _Tp, class _NData, class _Result, class _Walker> void coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.22.3.21 template<class _Tp, class _NData, class _Result, class _Walker> return_value coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result,_Walker >.

Definition at line 767 of file evaluator.h.

10.22.4 Member Data Documentation

10.22.4.1 _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::eval_data

`[protected, inherited]`

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.22.4.2 const variable_indicator* coco::cached_evaluator_base< _Tp , _NData , _Result , _Walker >::v_ind

`[protected, inherited]`

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:

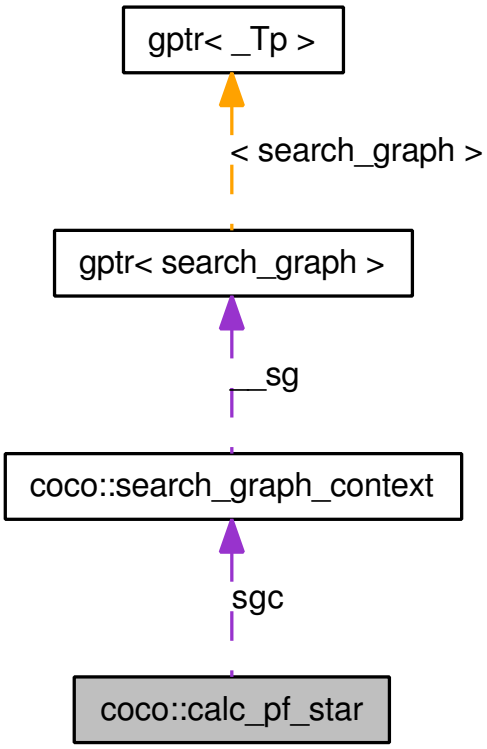
- evaluator.h

10.23 coco::calc_pf_star Class Reference

Stored procedure calculating the pf∗ value of a box.

`#include <pfstar.h>`

Collaboration diagram for coco::calc_pf_star:



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)

10.23.1 Detailed Description

This class is used as a stored procedure (vdbl::method) for calculating the pf∗ value of a subproblem.

Definition at line 43 of file pfstar.h.

10.23.2 Member Typedef Documentation

10.23.2.1 typedef search_graph_context coco::calc_pf_star::context

The evaluation context type

Definition at line 47 of file pfstar.h.

10.23.2.2 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.

10.23.3 Constructor & Destructor Documentation

10.23.3.1 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.

10.23.3.2 coco::calc_pf_star::calc_pf_star (const calc_pf_star & *_i*) [inline]

Standard Copy Constructor

Definition at line 69 of file pfstar.h.

10.23.3.3 virtual coco::calc_pf_star::~calc_pf_star () [inline, virtual]

Standard Destructor

Definition at line 73 of file pfstar.h.

10.23.4 Member Function Documentation

10.23.4.1 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.

10.23.4.2 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.

10.23.4.3 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:

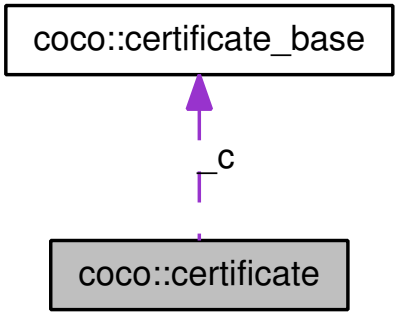
- pfstar.h
- pfstar.cc

10.24 coco::certificate Class Reference

The certificate class (certifies deltas for rigorous mode operation).

`#include <api_certbase.h>`

Collaboration diagram for coco::certificate:



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.

10.24.1 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 api_certbase.h.

10.24.2 Constructor & Destructor Documentation

10.24.2.1 coco::certificate::certificate () [inline]

Standard Constructor

Definition at line 62 of file api_certbase.h.

10.24.2.2 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.

10.24.2.3 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.

10.24.2.4 coco::certificate::~certificate () [inline]

Destructor, which frees the _c using delete

Definition at line 49 of file api_cert.h.

10.24.3 Member Function Documentation

10.24.3.1 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.

10.24.3.2 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.

10.24.3.3 bool coco::certificate::operator!= (const certificate & _c) const [inline]

Definition at line 74 of file api_cert.h.

10.24.3.4 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.

10.24.3.5 bool coco::certificate::operator== (const certificate & _c) const [inline]

Comparison operators

Definition at line 71 of file api_cert.h.

10.24.3.6 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.

10.24.4 Friends And Related Function Documentation

10.24.4.1 friend class certificate_base [friend]

Definition at line 102 of file api_certbase.h.

10.24.4.2 friend class ie_return_type [friend]

Definition at line 103 of file api_certbase.h.

10.24.4.3 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 files:

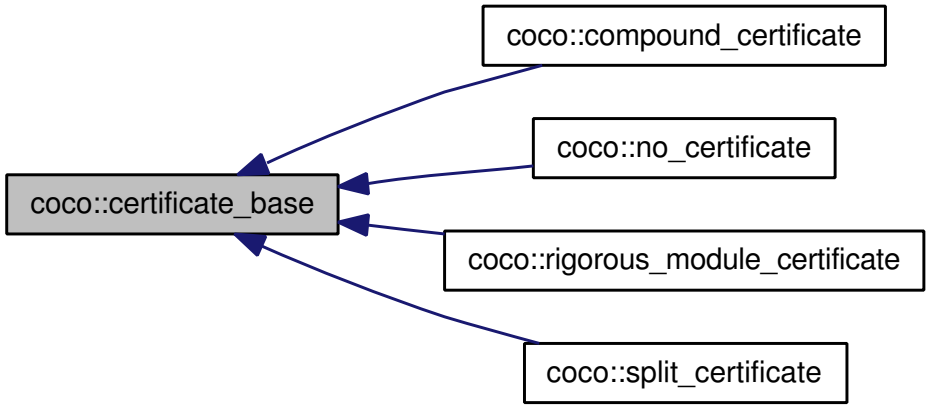
- api_certbase.h
- api_cert.h

10.25 coco::certificate_base Class Reference

Base class for the certificates.

`#include <api_certbase.h>`

Inheritance diagram for 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

Protected Attributes

- std::string _contents

10.25.1 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 api_certbase.h.

10.25.2 Constructor & Destructor Documentation

10.25.2.1 coco::certificate_base::certificate_base () [inline]

Standard Constructor

Definition at line 132 of file api_certbase.h.

10.25.2.2 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.

10.25.2.3 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.

10.25.2.4 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.

10.25.3 Member Function Documentation

10.25.3.1 virtual std::string coco::certificate_base::get_contents () const [inline, virtual]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::split_certificate, and coco::rigorous_module_certificate.

Definition at line 165 of file api_certbase.h.

10.25.3.2 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.

10.25.3.3 virtual certificate_base* coco::certificate_base::new_copy () const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::no_certificate, coco::split_certificate, coco::compound_certificate, and coco::rigorous_module_certificate.

Definition at line 146 of file api_certbase.h.

10.25.3.4 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.

Reimplemented in coco::no_certificate, coco::split_certificate, coco::compound_certificate, and coco::rigorous_module_certificate.

Definition at line 169 of file api_certbase.h.

10.25.4 Member Data Documentation

10.25.4.1 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

10.26 coco::checking_my< T > Struct Template 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)

10.26.1 Detailed Description

template<class T> struct coco::checking_my< T >

Definition at line 49 of file interval_boost.h.

10.26.2 Member Function Documentation

10.26.2.1 template<class T > static T coco::checking_my< T >::empty_inf () [inline, static]

Definition at line 68 of file interval_boost.h.

10.26.2.2 template<class T > static T coco::checking_my< T >::empty_lower () [inline, static]

Definition at line 78 of file interval_boost.h.

10.26.2.3 template<class T > static T coco::checking_my< T >::empty_upper () [inline, static]

Definition at line 73 of file interval_boost.h.

10.26.2.4 template<class T > static T coco::checking_my< T >::inf () [inline, static]

Definition at line 53 of file interval_boost.h.

10.26.2.5 template<class T > static bool coco::checking_my< T >::is_empty (const T & *l*, const T & *u*) [inline, static]

Definition at line 83 of file interval_boost.h.

10.26.2.6 template<class T > static bool coco::checking_my< T >::is_nan (const T & *x*) [inline, static]

Definition at line 64 of file interval_boost.h.

10.26.2.7 template<class T > static T coco::checking_my< T >::nan () [inline, static]

Definition at line 59 of file interval_boost.h.

10.26.2.8 template<class T > static T coco::checking_my< T >::neg_inf () [inline, static]

Definition at line 52 of file interval_boost.h.

10.26.2.9 template<class T > static T coco::checking_my< T >::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

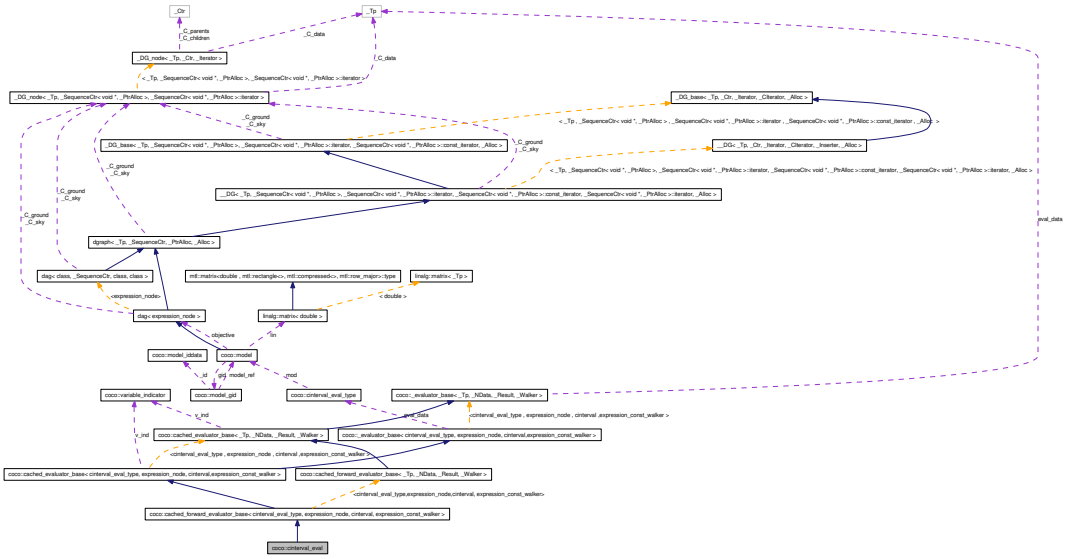
10.27 coco::cinterval_eval Class Reference

`#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
- typedef cinterval_eval_type data_type

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 void cleanup (const node_data_type &__data)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- cinterval_eval_type eval_data

10.27.1 Detailed Description

Definition at line 60 of file cint_evaluator.h.

10.27.2 Member Typedef Documentation

10.27.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.27.2.2 typedef cinterval_eval_type coco::_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.27.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.27.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.27.3 Constructor & Destructor Documentation

10.27.3.1 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 115 of file cint_evaluator.h.

10.27.3.2 coco::cinterval_eval::cinterval_eval (const cinterval_eval & *__v*) [inline]

Definition at line 127 of file cint_evaluator.h.

10.27.3.3 coco::cinterval_eval::~cinterval_eval () [inline]

Definition at line 129 of file cint_evaluator.h.

10.27.4 Member Function Documentation

10.27.4.1 void coco::cinterval_eval::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 192 of file cint_evaluator.h.

10.27.4.2 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::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 669 of file cint_evaluator.h.

10.27.4.3 virtual void coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::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 804 of file evaluator.h.

10.27.4.4 int coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.27.4.5 int coco::cinterval_eval::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 136 of file cint_evaluator.h.

10.27.4.6 void coco::cinterval_eval::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 134 of file cint_evaluator.h.

10.27.4.7 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::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 69 of file cint_evaluator.h.

10.27.4.8 void coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.27.4.9 int coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.27.4.10 void coco::cinterval_eval::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 202 of file cint_evaluator.h.

10.27.4.11 expression_const_walker coco::cinterval_eval::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 131 of file cint_evaluator.h.

10.27.4.12 int coco::cinterval_eval::update (const expression_node & *__data*, const cinterval & *__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.

Reimplemented from coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 227 of file cint_evaluator.h.

10.27.4.13 int coco::cinterval_eval::update (const cinterval & *__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.

Reimplemented from coco::cached_forward_evaluator_base< cinterval_eval_type, expression_node, cinterval, expression_const_walker >.

Definition at line 221 of file cint_evaluator.h.

10.27.4.14 return_value coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.27.4.15 int coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.27.4.16 void coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::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 771 of file evaluator.h.

10.27.4.17 return_value coco::cached_forward_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< cinterval_eval_type, expression_node, cinterval,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.27.5 Member Data Documentation

10.27.5.1 cinterval_eval_type coco::_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.27.5.2 const variable_indicator* coco::cached_evaluator_base< cinterval_eval_type , expression_node , cinterval , expression_const_walker >::v_ind [protected, inherited]

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:

- cint_evaluator.h

10.28 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

10.28.1 Detailed Description

Definition at line 48 of file cint_evaluator.h.

10.28.2 Member Data Documentation

10.28.2.1 std::vector<cinterval>* coco::cinterval_eval_type::cache

Definition at line 51 of file cint_evaluator.h.

10.28.2.2 cinterval coco::cinterval_eval_type::d

Definition at line 54 of file cint_evaluator.h.

10.28.2.3 int coco::cinterval_eval_type::info

Definition at line 55 of file cint_evaluator.h.

10.28.2.4 const model* coco::cinterval_eval_type::mod

Definition at line 52 of file cint_evaluator.h.

10.28.2.5 unsigned int coco::cinterval_eval_type::n

Definition at line 57 of file cint_evaluator.h.

10.28.2.6 void* coco::cinterval_eval_type::p

Definition at line 53 of file cint_evaluator.h.

10.28.2.7 cinterval coco::cinterval_eval_type::r

Definition at line 56 of file cint_evaluator.h.

10.28.2.8 const std::vector<cinterval>* coco::cinterval_eval_type::x

Definition at line 50 of file cint_evaluator.h.

The documentation for this struct was generated from the following file:

- cint_evaluator.h

10.29 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

10.29.1 Detailed Description

This is a class which can be used as random number object in STL like algorithms.

Definition at line 249 of file coconut_random.h.

10.29.2 Constructor & Destructor Documentation

10.29.2.1 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.

10.29.3 Member Function Documentation

10.29.3.1 size_t coco::coconut_random_f::operator() (size_t *n*) const [inline]

Definition at line 272 of file coconut_random.h.

10.29.3.2 int coco::coconut_random_f::operator() (long int *n*) const [inline]

Definition at line 269 of file coconut_random.h.

10.29.3.3 int coco::coconut_random_f::operator() (int *n*) const [inline]

Definition at line 266 of file coconut_random.h.

10.29.3.4 double coco::coconut_random_f::operator() () const [inline]

Definition at line 263 of file coconut_random.h.

10.29.3.5 double coco::coconut_random_f::operator() (double *u*) const [inline]

Definition at line 260 of file coconut_random.h.

10.29.3.6 double coco::coconut_random_f::operator() (double *l*, double *u*) const [inline]

Definition at line 257 of file coconut_random.h.

10.29.3.7 interval coco::coconut_random_f::operator() (const interval & *i*) const [inline]

Definition at line 254 of file coconut_random.h.

10.29.4 Member Data Documentation

10.29.4.1 double coco::coconut_random_f::alpha

Definition at line 252 of file coconut_random.h.

10.29.4.2 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:

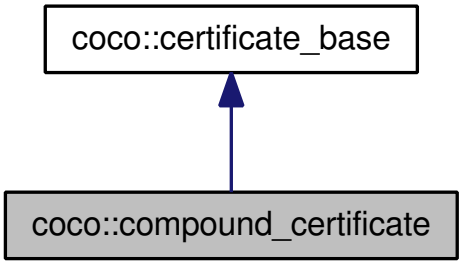
- coconut_random.h

10.30 coco::compound_certificate Class Reference

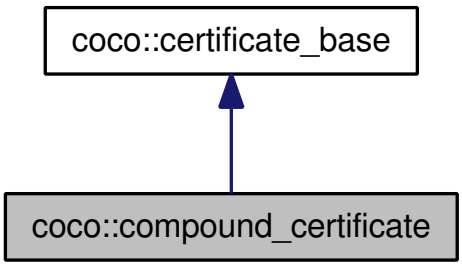
The certificate for deltas formed by compressing bound_delta entries.

`#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)
- virtual std::string get_contents () const

Protected Attributes

- std::string _contents

10.30.1 Detailed Description

The compound certificate for compound deltas generated by management modules

Definition at line 201 of file api_cert.h.

10.30.2 Constructor & Destructor Documentation

10.30.2.1 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.

10.30.2.2 coco::compound_certificate::compound_certificate (const compound_certificate & __c) [inline]

Standard Copy Constructor

Definition at line 213 of file api_cert.h.

10.30.2.3 virtual coco::compound_certificate::~compound_certificate () [inline, virtual]

Standard Destructor

Definition at line 222 of file api_cert.h.

10.30.3 Member Function Documentation

10.30.3.1 void coco::compound_certificate::destroy_copy (certificate_base * __c) const [inline]

Clone Destructor

Definition at line 233 of file api_cert.h.

10.30.3.2 virtual std::string coco::certificate_base::get_contents () const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::split_certificate, and coco::rigorous_module_certificate.

Definition at line 165 of file api_certbase.h.

10.30.3.3 certificate 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 api_certbase.h.

10.30.3.4 compound_certificate* coco::compound_certificate::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::certificate_base.

Definition at line 225 of file api_cert.h.

10.30.3.5 bool coco::compound_certificate::operator!= (const compound_certificate & _c) const [inline]

Definition at line 252 of file api_cert.h.

10.30.3.6 bool coco::compound_certificate::operator!= (const certificate_base & _c) const [inline]

Definition at line 245 of file api_cert.h.

10.30.3.7 bool coco::compound_certificate::operator== (const compound_certificate & _c) const [inline]

Comparison operators

Definition at line 250 of file api_cert.h.

10.30.3.8 bool coco::compound_certificate::operator== (const certificate_base & _c) const [inline]

Comparison operators

Definition at line 242 of file api_cert.h.

10.30.3.9 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::certificate_base.

Definition at line 239 of file api_cert.h.

10.30.4 Member Data Documentation

10.30.4.1 std::string coco::certificate_base::_contents [protected, inherited]

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:

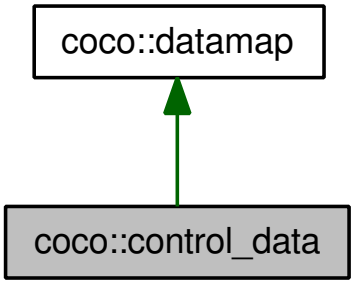
- api_cert.h

10.31 coco::control_data Class Reference

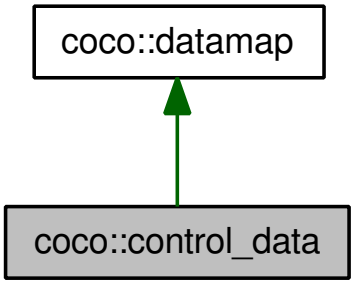
The class for communicating parameter information to COCONUT modules.

`#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 linalg::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 linalg::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 linalg::matrix< _TS > *&__b, const linalg::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 linalg::matrix< _TS > *&__b, const linalg::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 linalg::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 linalg::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 linalg::matrix< _TS > *&__b, const linalg::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 linalg::matrix< _TS > *&__b, const linalg::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, 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 linalg::matrix< double > *&__v) const

- bool retrieve (const std::string &__s, const linalg::matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const linalg::matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< int > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< interval > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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 &__is) 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 linalg::matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__def) const

10.31.1 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.

Definition at line 44 of file control_data.h.

10.31.2 Constructor & Destructor Documentation

10.31.2.1 coco::control_data::control_data () [inline]

Standard Constructor

Definition at line 55 of file control_data.h.

10.31.2.2 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.

10.31.2.3 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.

10.31.2.4 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.

10.31.2.5 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.

10.31.2.6 virtual coco::control_data::~control_data () [inline, virtual]

Standard Destructor

Definition at line 71 of file control_data.h.

10.31.3 Member Function Documentation

10.31.3.1 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 174 of file control_data.hpp.

10.31.3.2 template<class _TS > void coco::control_data::assign (const char * __s, const linalg::matrix< _TS > *& __b, const linalg::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 167 of file control_data.hpp.

10.31.3.3 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 160 of file control_data.hpp.

10.31.3.4 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 125 of file control_data.hpp.

10.31.3.5 template<class _TS > void coco::control_data::assign (const std::string & *__s*, const linalg::matrix< _TS > *& *__b*, const linalg::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 118 of file control_data.hpp.

10.31.3.6 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 110 of file control_data.hpp.

10.31.3.7 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 153 of file control_data.hpp.

10.31.3.8 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 103 of file control_data.hpp.

10.31.3.9 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 147 of file control_data.hpp.

10.31.3.10 template<class _TS > void coco::control_data::assign (const char * __s, const linalg::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 140 of file control_data.hpp.

10.31.3.11 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 132 of file control_data.hpp.

10.31.3.12 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 92 of file control_data.hpp.

10.31.3.13 template<class _TS > void coco::control_data::assign (const std::string & __s, const linalg::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 79 of file control_data.hpp.

10.31.3.14 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 66 of file control_data.hpp.

10.31.3.15 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 293 of file control_data.hpp.

10.31.3.16 template<class _TS > void coco::control_data::assign_i (const char * *__s*, int *i*, const linalg::matrix< _TS > *& *__b*, const linalg::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 285 of file control_data.hpp.

10.31.3.17 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 277 of file control_data.hpp.

10.31.3.18 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 241 of file control_data.hpp.

10.31.3.19 template<class _TS > void coco::control_data::assign_i (const std::string & *__s*, int *i*, const linalg::matrix< _TS > *& *__b*, const linalg::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 233 of file control_data.hpp.

10.31.3.20 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 225 of file control_data.hpp.

10.31.3.21 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 270 of file control_data.hpp.

10.31.3.22 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 218 of file control_data.hpp.

10.31.3.23 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 264 of file control_data.hpp.

10.31.3.24 template<class _TS > void coco::control_data::assign_i (const char * *__s*, int *i*, const linalg::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 256 of file control_data.hpp.

10.31.3.25 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 248 of file control_data.hpp.

10.31.3.26 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 207 of file control_data.hpp.

10.31.3.27 template<class _TS > void coco::control_data::assign_i (const std::string & *__s*, int *i*, const linalg::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 194 of file control_data.hpp.

10.31.3.28 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 181 of file control_data.hpp.

10.31.3.29 bool coco::control_data::check_service (const char * *__n*) const [inline]

This method checks whether the service is set to ___n.

Definition at line 55 of file control_data.hpp.

10.31.3.30 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 50 of file control_data.hpp.

10.31.3.31 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.

10.31.3.32 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.

10.31.3.33 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.

10.31.3.34 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.

10.31.3.35 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.

10.31.3.36 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.

10.31.3.37 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.

10.31.3.38 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.

10.31.3.39 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.

10.31.3.40 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.

10.31.3.41 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.

10.31.3.42 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.

10.31.3.43 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::datamap.

Definition at line 60 of file control_data.hpp.

10.31.3.44 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.

10.31.3.45 control_data& coco::control_data::operator= (const control_data & __c) [inline]

Standard Assignment Operator

Definition at line 74 of file control_data.h.

10.31.3.46 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.

10.31.3.47 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.

10.31.3.48 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.

10.31.3.49 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.

10.31.3.50 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< std::string > *& __v, const linalg::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 658 of file datamap.hpp.

10.31.3.51 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< interval > *& __v, const linalg::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 647 of file datamap.hpp.

10.31.3.52 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< int > *& __v, const linalg::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 636 of file datamap.hpp.

10.31.3.53 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< double > *& __v, const linalg::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 625 of file datamap.hpp.

10.31.3.54 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 615 of file datamap.hpp.

10.31.3.55 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 605 of file datamap.hpp.

10.31.3.56 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 595 of file datamap.hpp.

10.31.3.57 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 575 of file datamap.hpp.

10.31.3.58 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 585 of file datamap.hpp.

10.31.3.59 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 565 of file datamap.hpp.

10.31.3.60 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 555 of file datamap.hpp.

10.31.3.61 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 545 of file datamap.hpp.

10.31.3.62 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 535 of file datamap.hpp.

10.31.3.63 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 525 of file datamap.hpp.

10.31.3.64 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 515 of file datamap.hpp.

10.31.3.65 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 505 of file datamap.hpp.

10.31.3.66 bool coco::datamap::retrieve (const std::string & __s, const linalg::matrix< std::string > *& __v, const linalg::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 489 of file datamap.hpp.

10.31.3.67 bool coco::datamap::retrieve (const std::string & *__s*, const linalg::matrix< interval > *& *__v*, const linalg::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 467 of file datamap.hpp.

10.31.3.68 bool coco::datamap::retrieve (const std::string & *__s*, const linalg::matrix< int > *& *__v*, const linalg::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 445 of file datamap.hpp.

10.31.3.69 bool coco::datamap::retrieve (const std::string & *__s*, const linalg::matrix< double > *& *__v*, const linalg::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 423 of file datamap.hpp.

10.31.3.70 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 401 of file datamap.hpp.

10.31.3.71 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 379 of file datamap.hpp.

10.31.3.72 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 357 of file datamap.hpp.

10.31.3.73 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 335 of file datamap.hpp.

10.31.3.74 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 313 of file datamap.hpp.

10.31.3.75 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 291 of file datamap.hpp.

10.31.3.76 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.

10.31.3.77 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.

10.31.3.78 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.

10.31.3.79 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.

10.31.3.80 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.

10.31.3.81 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.

10.31.3.82 bool coco::datamap::retrieve (const char * *__s*, const linalg::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 653 of file datamap.hpp.

10.31.3.83 bool coco::datamap::retrieve (const char * *__s*, const linalg::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 642 of file datamap.hpp.

10.31.3.84 bool coco::datamap::retrieve (const char * __s, const linalg::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.

10.31.3.85 bool coco::datamap::retrieve (const char * __s, const linalg::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 620 of file datamap.hpp.

10.31.3.86 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 610 of file datamap.hpp.

10.31.3.87 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 600 of file datamap.hpp.

10.31.3.88 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 590 of file datamap.hpp.

10.31.3.89 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 570 of file datamap.hpp.

10.31.3.90 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 580 of file datamap.hpp.

10.31.3.91 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 560 of file datamap.hpp.

10.31.3.92 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 550 of file datamap.hpp.

10.31.3.93 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 540 of file datamap.hpp.

10.31.3.94 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 530 of file datamap.hpp.

10.31.3.95 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 520 of file datamap.hpp.

10.31.3.96 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 510 of file datamap.hpp.

10.31.3.97 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 500 of file datamap.hpp.

10.31.3.98 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 477 of file datamap.hpp.

10.31.3.99 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 455 of file datamap.hpp.

10.31.3.100 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 433 of file datamap.hpp.

10.31.3.101 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 411 of file datamap.hpp.

10.31.3.102 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 389 of file datamap.hpp.

10.31.3.103 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 367 of file datamap.hpp.

10.31.3.104 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 345 of file datamap.hpp.

10.31.3.105 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 323 of file datamap.hpp.

10.31.3.106 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 301 of file datamap.hpp.

10.31.3.107 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 279 of file datamap.hpp.

10.31.3.108 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.

10.31.3.109 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.

10.31.3.110 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.

10.31.3.111 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.

10.31.3.112 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.

10.31.3.113 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.

10.31.3.114 bool coco::datamap::retrieve_i (const char * __s, int *i*, const linalg::matrix< std::string > *& __v, const linalg::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 1143 of file datamap.hpp.

10.31.3.115 bool coco::datamap::retrieve_i (const char * __s, int *i*, const linalg::matrix< interval > *& __v, const linalg::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 1132 of file datamap.hpp.

10.31.3.116 bool coco::datamap::retrieve_i (const char * __s, int *i*, const linalg::matrix< int > *& __v, const linalg::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 1121 of file datamap.hpp.

10.31.3.117 bool coco::datamap::retrieve_i (const char * *__s*, int *i*, const linalg::matrix< double > *&* *__v*, const linalg::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 1110 of file datamap.hpp.

10.31.3.118 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 1100 of file datamap.hpp.

10.31.3.119 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 1090 of file datamap.hpp.

10.31.3.120 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 1080 of file datamap.hpp.

10.31.3.121 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 1070 of file datamap.hpp.

10.31.3.122 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 1060 of file datamap.hpp.

10.31.3.123 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 1050 of file datamap.hpp.

10.31.3.124 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 1040 of file datamap.hpp.

10.31.3.125 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 1030 of file datamap.hpp.

10.31.3.126 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 1020 of file datamap.hpp.

10.31.3.127 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 1010 of file datamap.hpp.

10.31.3.128 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 1000 of file datamap.hpp.

10.31.3.129 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 990 of file datamap.hpp.

10.31.3.130 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const linalg::matrix< std::string > *& *__v*, const linalg::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 975 of file datamap.hpp.

10.31.3.131 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const linalg::matrix< interval > *& *__v*, const linalg::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 955 of file datamap.hpp.

10.31.3.132 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const linalg::matrix< int > *& *__v*, const linalg::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 935 of file datamap.hpp.

10.31.3.133 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const linalg::matrix< double > *& *__v*, const linalg::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 915 of file datamap.hpp.

10.31.3.134 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 895 of file datamap.hpp.

10.31.3.135 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 875 of file datamap.hpp.

10.31.3.136 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 855 of file datamap.hpp.

10.31.3.137 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 835 of file datamap.hpp.

10.31.3.138 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 815 of file datamap.hpp.

10.31.3.139 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 795 of file datamap.hpp.

10.31.3.140 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 775 of file datamap.hpp.

10.31.3.141 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 755 of file datamap.hpp.

10.31.3.142 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 735 of file datamap.hpp.

10.31.3.143 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 715 of file datamap.hpp.

10.31.3.144 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 695 of file datamap.hpp.

10.31.3.145 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 675 of file datamap.hpp.

10.31.3.146 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1138 of file datamap.hpp.

10.31.3.147 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1127 of file datamap.hpp.

10.31.3.148 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1116 of file datamap.hpp.

10.31.3.149 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1105 of file datamap.hpp.

10.31.3.150 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 1095 of file datamap.hpp.

10.31.3.151 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 1085 of file datamap.hpp.

10.31.3.152 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 1075 of file datamap.hpp.

10.31.3.153 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 1065 of file datamap.hpp.

10.31.3.154 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 1055 of file datamap.hpp.

10.31.3.155 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 1045 of file datamap.hpp.

10.31.3.156 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 1035 of file datamap.hpp.

10.31.3.157 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 1025 of file datamap.hpp.

10.31.3.158 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 1015 of file datamap.hpp.

10.31.3.159 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 1005 of file datamap.hpp.

10.31.3.160 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 995 of file datamap.hpp.

10.31.3.161 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 985 of file datamap.hpp.

10.31.3.162 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 964 of file datamap.hpp.

10.31.3.163 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 944 of file datamap.hpp.

10.31.3.164 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 924 of file datamap.hpp.

10.31.3.165 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 904 of file datamap.hpp.

10.31.3.166 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 884 of file datamap.hpp.

10.31.3.167 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 864 of file datamap.hpp.

10.31.3.168 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 844 of file datamap.hpp.

10.31.3.169 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 824 of file datamap.hpp.

10.31.3.170 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 804 of file datamap.hpp.

10.31.3.171 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 784 of file datamap.hpp.

10.31.3.172 bool coco::datamap::retrieve_i (const std::string & *__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 764 of file datamap.hpp.

10.31.3.173 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 744 of file datamap.hpp.

10.31.3.174 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 724 of file datamap.hpp.

10.31.3.175 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 704 of file datamap.hpp.

10.31.3.176 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 684 of file datamap.hpp.

10.31.3.177 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 664 of file datamap.hpp.

10.31.3.178 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 41 of file control_data.hpp.

10.31.3.179 void coco::control_data::service (const char * __s) [inline]

This method sets the service to __s.

Definition at line 36 of file control_data.hpp.

10.31.3.180 void coco::control_data::service (const std::string & __s) [inline]

This method sets the service to __s.

Definition at line 34 of file control_data.hpp.

10.31.3.181 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.

10.31.3.182 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.

10.31.3.183 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.

10.31.3.184 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.

10.31.3.185 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.

10.31.3.186 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.

10.31.3.187 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.

10.31.3.188 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.

10.31.3.189 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.

10.31.3.190 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.

10.31.3.191 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.

10.31.3.192 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.

10.31.3.193 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 34 of file datamap.cc.

10.31.3.194 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.

10.31.3.195 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.

10.31.3.196 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.

10.31.3.197 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.

10.31.3.198 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.

10.31.3.199 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.

10.31.3.200 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.

10.31.3.201 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.

The documentation for this class was generated from the following files:

- control_data.h
- control_data.hpp

10.32 coco::convex_e Class Reference

Convexity information.

`#include <semantics.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)
  Equality comparison operator with a convex_info.
- bool operator== (const convex_info &__c, const convex_e &__d)
  Equality comparison operator with a convex_info.
- bool operator!= (const convex_e &__c, const convex_e &__d)
  Disequality comparison operator.
- bool operator!= (const convex_e &__c, const convex_info &__d)
  Disequality comparison operator with a convex_info.
- bool operator!= (const convex_info &__c, const convex_e &__d)
  Disequality comparison operator with a convex_info.
- std::ostream & operator<< (std::ostream &o, const convex_e &__s)
  C++ stream output operator for convex_e.

10.32.1 Detailed Description

This class stores the convexity information of a node. It is part of the semantics structure.

Definition at line 106 of file semantics.h.

10.32.2 Constructor & Destructor Documentation

10.32.2.1 coco::convex_e::convex_e () [inline]

Standard Constructor

Definition at line 116 of file semantics.h.

10.32.2.2 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 119 of file semantics.h.

10.32.2.3 coco::convex_e::convex_e (const convex_e & __e) [inline]

Standard Copy Constructor

Definition at line 121 of file semantics.h.

10.32.2.4 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 124 of file semantics.h.

10.32.2.5 coco::convex_e::~convex_e () [inline]

Standard Destructor

Definition at line 127 of file semantics.h.

10.32.3 Member Function Documentation

10.32.3.1 const convex_info& coco::convex_e::i () const [inline]

This method returns the convex_info stored.

Definition at line 130 of file semantics.h.

10.32.3.2 void coco::convex_e::merge (const convex_e & __c) [inline]

This method merges a convex_e with another convex_e ___c.

Definition at line 182 of file semantics.h.

10.32.3.3 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 137 of file semantics.h.

10.32.3.4 convex_e& coco::convex_e::operator= (uint16_t *__t*) [inline]

Assignment Operator, which only assigns the algorithm complexity part.

Definition at line 172 of file semantics.h.

10.32.3.5 convex_e& coco::convex_e::operator= (unsigned int *__t*) [inline]

Assignment Operator, which only assigns the algorithm complexity part.

Definition at line 165 of file semantics.h.

10.32.3.6 convex_e& coco::convex_e::operator= (const convex_info & *__i*) [inline]

Assignment Operator, which only assigns the convex_info part.

Definition at line 158 of file semantics.h.

10.32.3.7 convex_e& coco::convex_e::operator= (const convex_e & *__c*) [inline]

Standard Assignment Operator

Definition at line 150 of file semantics.h.

10.32.3.8 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 semantics.cc.

10.32.3.9 uint16_t coco::convex_e::t () const [inline]

This method returns the algorithm complexity stored.

Definition at line 132 of file semantics.h.

10.32.4 Friends And Related Function Documentation

10.32.4.1 bool operator!= (const convex_info & __c, const convex_e & __d) [friend]

Disequality comparison operator with a convex_info.

Definition at line 250 of file semantics.h.

10.32.4.2 bool operator!= (const convex_e & __c, const convex_info & __d) [friend]

Disequality comparison operator with a convex_info.

Definition at line 242 of file semantics.h.

10.32.4.3 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 234 of file semantics.h.

10.32.4.4 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 258 of file semantics.h.

10.32.4.5 bool operator== (const convex_info & __c, const convex_e & __d) [friend]

Equality comparison operator with a convex_info.

Definition at line 226 of file semantics.h.

10.32.4.6 bool operator== (const convex_e & __c, const convex_info & __d) [friend]

Equality comparison operator with a convex_info.

Definition at line 218 of file semantics.h.

10.32.4.7 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 210 of file semantics.h.

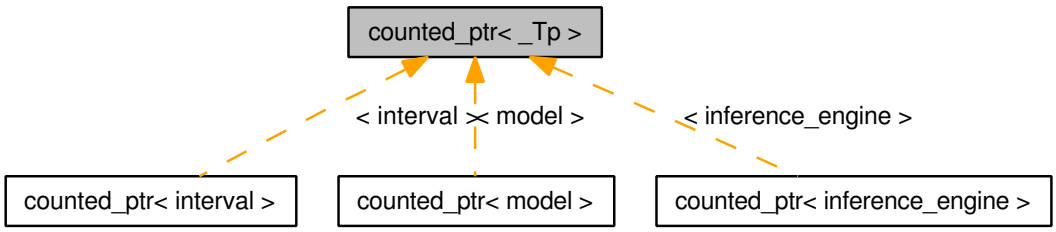
The documentation for this class was generated from the following files:

- semantics.h
- semantics.cc

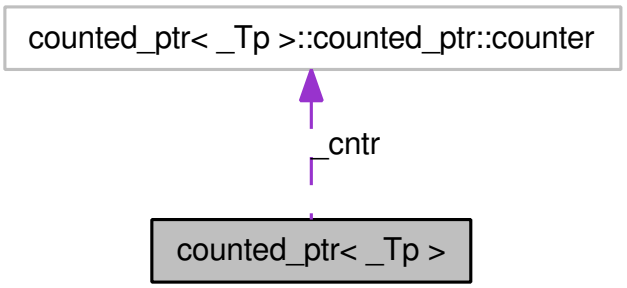
10.33 counted_ptr< _Tp > Class Template Reference

`#include <counted_ptr.h>`

Inheritance diagram for counted_ptr< _Tp >:



Collaboration diagram for counted_ptr< _Tp >:



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)
- _Tp & operator* () const throw ()
- _Tp * operator -> () const throw ()
- _Tp * get () const throw ()
- bool is_unique () const throw ()

10.33.1 Detailed Description

template<typename _Tp> class counted_ptr< _Tp >

This class provides a simple reference counted pointer. The implementation is non-intrusive; it allocates an additional int and pointer for every counted object.

Definition at line 39 of file counted_ptr.h.

10.33.2 Member Typedef Documentation

10.33.2.1 template<typename _Tp> typedef _Tp counted_ptr< _Tp >::element_type

The element type of the counted pointer

Definition at line 63 of file counted_ptr.h.

10.33.3 Constructor & Destructor Documentation

10.33.3.1 template<typename _Tp> counted_ptr< _Tp >::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.

10.33.3.2 template<typename _Tp> counted_ptr< _Tp >::~counted_ptr () [inline]

The Standard Destructor just deletes this reference to the counted object.

Definition at line 72 of file counted_ptr.h.

10.33.3.3 template<typename _Tp> counted_ptr< _Tp >::counted_ptr (const counted_ptr< _Tp > & *r*) throw () [inline]

The Copy Constructor gets a new reference to the counted object.

Definition at line 75 of file counted_ptr.h.

10.33.3.4 template<typename _Tp> template<typename _Tq > counted_ptr< _Tp >::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.

10.33.4 Member Function Documentation

10.33.4.1 template<typename _Tp> _Tp* counted_ptr< _Tp >::get () const throw () [inline]

This is a safeguarded pointer operator

Definition at line 135 of file counted_ptr.h.

10.33.4.2 template<typename _Tp> bool counted_ptr< _Tp >::is_unique () const throw () [inline]

This checks whether the object has only this (unique) reference.

Definition at line 138 of file counted_ptr.h.

10.33.4.3 template<typename _Tp> _Tp& counted_ptr< _Tp >::operator* () const throw () [inline]

This is the dereferentiation operator

Definition at line 129 of file counted_ptr.h.

10.33.4.4 template<typename _Tp> _Tp* counted_ptr< _Tp >::operator → () const throw () [inline]

This is the pointer operator

Definition at line 132 of file counted_ptr.h.

10.33.4.5 template<typename _Tp> template<typename _Tq > counted_ptr& counted_ptr< _Tp >::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.

10.33.4.6 template<typename _Tp> counted_ptr& counted_ptr< _Tp >::operator= (element_type * *r*) [inline]

This Assignment Operator explicitly sets a new element_type pointer.

Definition at line 92 of file counted_ptr.h.

10.33.4.7 template<typename _Tp> counted_ptr& counted_ptr< _Tp >::operator= (const counted_ptr< _Tp > & *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.

The documentation for this class was generated from the following file:

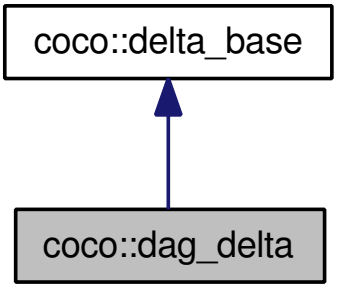
- counted_ptr.h

10.34 coco::dag_delta Class Reference

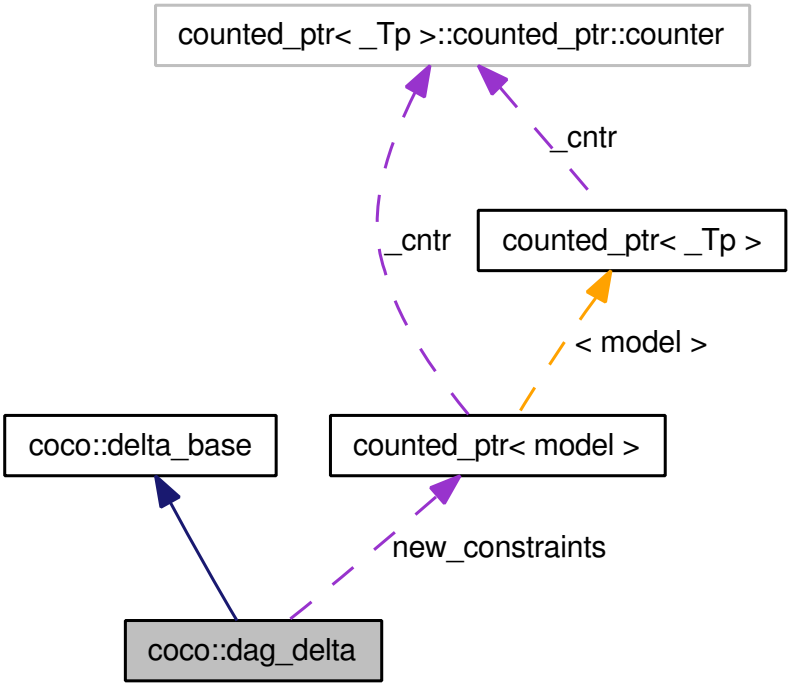
The DAG delta class for performing changes to the DAG of a model.

`#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) 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)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) 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

10.34.1 Detailed Description

This class is used to specify information for changes to the DAG of a model.

Definition at line 161 of file dag_delta.h.

10.34.2 Constructor & Destructor Documentation

10.34.2.1 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 185 of file dag_delta.h.

10.34.2.2 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 193 of file dag_delta.h.

10.34.2.3 coco::dag_delta::dag_delta (const dag_delta & *__d*) [inline]

Standard Copy Constructor

Definition at line 200 of file dag_delta.h.

10.34.2.4 coco::dag_delta::~dag_delta () [inline]

Standard Destructor

Definition at line 211 of file dag_delta.h.

10.34.3 Member Function Documentation

10.34.3.1 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.

10.34.3.2 void coco::dag_delta::add_new (model * *__m*) [inline]

This method initializes the new_constraints member.

Definition at line 228 of file dag_delta.h.

10.34.3.3 bool coco::dag_delta::apply (work_node & *x*, undelta_base *& *_u*, const delta_id & *_did*) const [virtual]

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

Reimplemented from coco::delta_base.

Definition at line 110 of file dag_delta.cc.

10.34.3.4 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *&& *_u*, const delta_id & *_d*) 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.

10.34.3.5 virtual void 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.

Reimplemented in coco::table_delta.

Definition at line 189 of file api_deltabase.h.

10.34.3.6 void coco::dag_delta::destroy_copy (delta_base * *__d*) const [inline]

Clone Destructor

Definition at line 225 of file dag_delta.h.

10.34.3.7 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.34.3.8 delta 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 api_deltabase.h.

10.34.3.9 dag_delta* coco::dag_delta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::delta_base.

Definition at line 223 of file dag_delta.h.

10.34.3.10 bool coco::dag_delta::operator!= (const dag_delta & _c) const [inline]

Definition at line 312 of file dag_delta.h.

10.34.3.11 bool coco::dag_delta::operator!= (const delta_base & _c) const [inline]

Definition at line 305 of file dag_delta.h.

10.34.3.12 bool coco::dag_delta::operator== (const dag_delta & _c) const [inline]

Comparison operators

Definition at line 310 of file dag_delta.h.

10.34.3.13 bool coco::dag_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Definition at line 302 of file dag_delta.h.

10.34.3.14 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 242 of file dag_delta.h.

10.34.3.15 void coco::dag_delta::remove (const walker & *_nn*) [inline]

This method adds another node _nn to be removed.

Definition at line 239 of file dag_delta.h.

10.34.3.16 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::delta_base.

Definition at line 247 of file dag_delta.h.

10.34.4 Friends And Related Function Documentation

10.34.4.1 friend class dag_undelta [friend]

Definition at line 315 of file dag_delta.h.

10.34.5 Member Data Documentation

10.34.5.1 std::string coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file api_deltabase.h.

10.34.5.2 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 181 of file dag_delta.h.

10.34.5.3 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 170 of file dag_delta.h.

10.34.5.4 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 176 of file dag_delta.h.

The documentation for this class was generated from the following files:

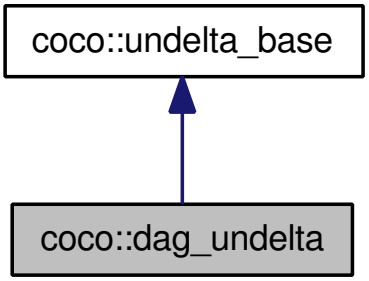
- dag_delta.h
- dag_delta.cc

10.35 coco::dag_undelta Class Reference

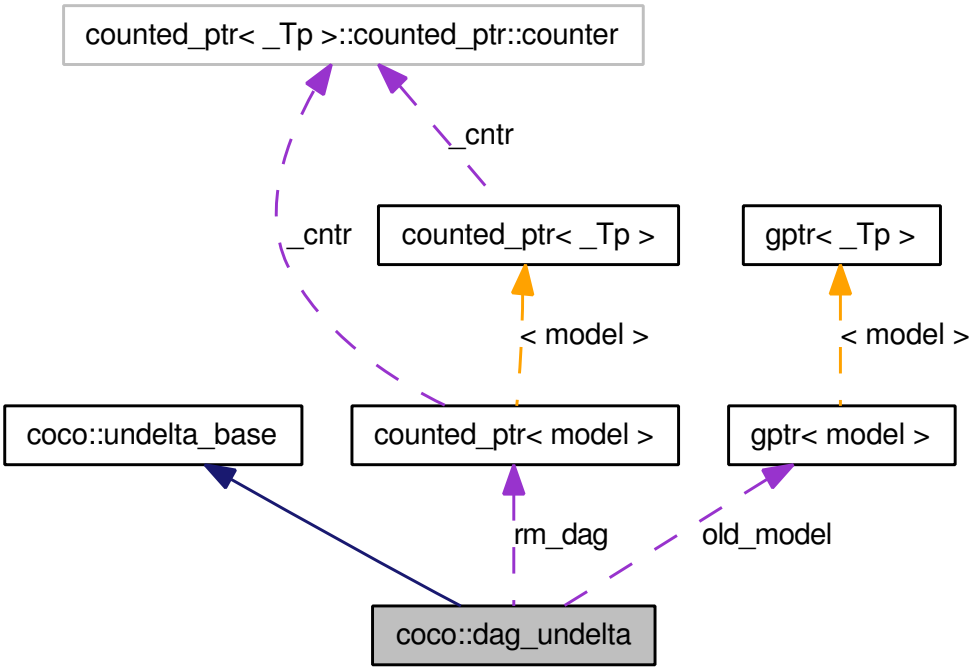
The DAG undelta class for undoing changes to the DAG of a model.

`#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 ()
- 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_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

10.35.1 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.

Definition at line 42 of file dag_delta.h.

10.35.2 Constructor & Destructor Documentation

10.35.2.1 coco::dag_undelta::dag_undelta (bool *full* = `false`) `[inline]`

Standard Constructor, setting the is_full_undelta member to `full`.

Definition at line 101 of file dag_delta.h.

10.35.2.2 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 111 of file dag_delta.h.

10.35.2.3 coco::dag_undelta::dag_undelta (const dag_undelta & *_du*) `[inline]`

Standard Copy Constructor

Definition at line 120 of file dag_delta.h.

10.35.2.4 coco::dag_undelta::~dag_undelta () `[inline]`

Standard Destructor

Definition at line 136 of file dag_delta.h.

10.35.3 Member Function Documentation

10.35.3.1 void coco::dag_undelta::destroy_copy (undelta_base * *__d*) const [inline]

Clone Destructor

Definition at line 147 of file dag_delta.h.

10.35.3.2 undelta coco::undelta_base::make_undelta () [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 283 of file api_deltabase.h.

10.35.3.3 dag_undelta* coco::dag_undelta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::undelta_base.

Definition at line 145 of file dag_delta.h.

10.35.3.4 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::undelta_base.

Definition at line 327 of file dag_delta.cc.

10.35.3.5 bool 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.

10.35.4 Friends And Related Function Documentation

10.35.4.1 friend class dag_delta [friend]

Definition at line 152 of file dag_delta.h.

10.35.5 Member Data Documentation

10.35.5.1 std::vector<walker> coco::dag_undelta::added_constraints

The vector of constraints added by the dag_delta

Definition at line 62 of file dag_delta.h.

10.35.5.2 std::vector<walker> coco::dag_undelta::added_ghosts

The vector of ghosts added by the dag_delta

Definition at line 64 of file dag_delta.h.

10.35.5.3 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.

10.35.5.4 std::vector<walker> coco::dag_undelta::added_vars

The vector of variables added by the dag_delta

Definition at line 66 of file dag_delta.h.

10.35.5.5 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 71 of file dag_delta.h.

10.35.5.6 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 75 of file dag_delta.h.

10.35.5.7 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.

10.35.5.8 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.

10.35.5.9 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:

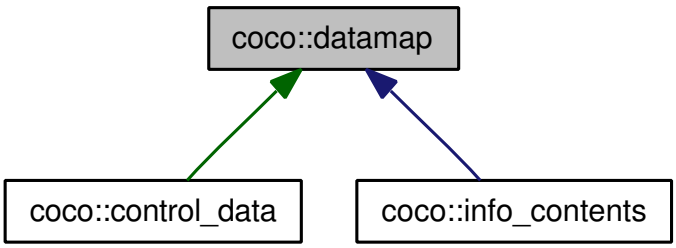
- dag_delta.h
- dag_delta.cc

10.36 coco::datamap Class Reference

The base class for communicating with COCONUT modules.

`#include <datamap.h>`

Inheritance diagram for 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, 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 linalg::matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const linalg::matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const linalg::matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< int > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< interval > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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 &__is) 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 linalg::matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< int > *&__v) const

- bool retrieve_i (const char *__s, int i, const linalg::matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__def) const

10.36.1 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).

Definition at line 44 of file datamap.h.

10.36.2 Constructor & Destructor Documentation

10.36.2.1 coco::datamap::datamap () [inline]

Standard Constructor

Definition at line 66 of file datamap.h.

10.36.2.2 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.

10.36.2.3 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.

10.36.2.4 coco::datamap::datamap (const datamap & __c) [inline]

Standard Copy Constructor

Definition at line 76 of file datamap.h.

10.36.2.5 virtual coco::datamap::~datamap () [inline, virtual]

Standard Destructor

Definition at line 78 of file datamap.h.

10.36.3 Member Function Documentation

10.36.3.1 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.

10.36.3.2 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.

10.36.3.3 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.

10.36.3.4 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.

10.36.3.5 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 99 of file datamap.cc.

10.36.3.6 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.

10.36.3.7 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.

10.36.3.8 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.

10.36.3.9 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.

10.36.3.10 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< std::string > *& __v, const linalg::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 658 of file datamap.hpp.

10.36.3.11 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< interval > *& __v, const linalg::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 647 of file datamap.hpp.

10.36.3.12 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< int > *& __v, const linalg::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 636 of file datamap.hpp.

10.36.3.13 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< double > *& __v, const linalg::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 625 of file datamap.hpp.

10.36.3.14 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 615 of file datamap.hpp.

10.36.3.15 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 605 of file datamap.hpp.

10.36.3.16 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 595 of file datamap.hpp.

10.36.3.17 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 575 of file datamap.hpp.

10.36.3.18 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 585 of file datamap.hpp.

10.36.3.19 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 565 of file datamap.hpp.

10.36.3.20 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 555 of file datamap.hpp.

10.36.3.21 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 545 of file datamap.hpp.

10.36.3.22 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 535 of file datamap.hpp.

10.36.3.23 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 525 of file datamap.hpp.

10.36.3.24 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 515 of file datamap.hpp.

10.36.3.25 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 505 of file datamap.hpp.

10.36.3.26 bool coco::datamap::retrieve (const std::string & __s, const linalg::matrix< std::string > *& __v, const linalg::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 489 of file datamap.hpp.

10.36.3.27 bool coco::datamap::retrieve (const std::string & __s, const linalg::matrix< interval > *& __v, const linalg::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 467 of file datamap.hpp.

10.36.3.28 bool coco::datamap::retrieve (const std::string & __s, const linalg::matrix< int > *& __v, const linalg::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 445 of file datamap.hpp.

10.36.3.29 bool coco::datamap::retrieve (const std::string & __s, const linalg::matrix< double > *& __v, const linalg::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 423 of file datamap.hpp.

10.36.3.30 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 401 of file datamap.hpp.

10.36.3.31 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 379 of file datamap.hpp.

10.36.3.32 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 357 of file datamap.hpp.

10.36.3.33 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 335 of file datamap.hpp.

10.36.3.34 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 313 of file datamap.hpp.

10.36.3.35 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 291 of file datamap.hpp.

10.36.3.36 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.

10.36.3.37 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.

10.36.3.38 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.

10.36.3.39 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.

10.36.3.40 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.

10.36.3.41 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.

10.36.3.42 bool coco::datamap::retrieve (const char * *__s*, const linalg::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 653 of file datamap.hpp.

10.36.3.43 bool coco::datamap::retrieve (const char * __s, const linalg::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 642 of file datamap.hpp.

10.36.3.44 bool coco::datamap::retrieve (const char * __s, const linalg::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.

10.36.3.45 bool coco::datamap::retrieve (const char * __s, const linalg::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 620 of file datamap.hpp.

10.36.3.46 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 610 of file datamap.hpp.

10.36.3.47 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 600 of file datamap.hpp.

10.36.3.48 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 590 of file datamap.hpp.

10.36.3.49 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 570 of file datamap.hpp.

10.36.3.50 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 580 of file datamap.hpp.

10.36.3.51 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 560 of file datamap.hpp.

10.36.3.52 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 550 of file datamap.hpp.

10.36.3.53 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 540 of file datamap.hpp.

10.36.3.54 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 530 of file datamap.hpp.

10.36.3.55 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 520 of file datamap.hpp.

10.36.3.56 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 510 of file datamap.hpp.

10.36.3.57 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 500 of file datamap.hpp.

10.36.3.58 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 477 of file datamap.hpp.

10.36.3.59 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 455 of file datamap.hpp.

10.36.3.60 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 433 of file datamap.hpp.

10.36.3.61 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 411 of file datamap.hpp.

10.36.3.62 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 389 of file datamap.hpp.

10.36.3.63 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 367 of file datamap.hpp.

10.36.3.64 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 345 of file datamap.hpp.

10.36.3.65 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 323 of file datamap.hpp.

10.36.3.66 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 301 of file datamap.hpp.

10.36.3.67 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 279 of file datamap.hpp.

10.36.3.68 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.

10.36.3.69 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.

10.36.3.70 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.

10.36.3.71 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.

10.36.3.72 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.

10.36.3.73 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.

10.36.3.74 bool coco::datamap::retrieve_i (const char * *__s*, int *i*, const linalg::matrix< std::string > *& *__v*, const linalg::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 1143 of file datamap.hpp.

10.36.3.75 bool coco::datamap::retrieve_i (const char * *__s*, int *i*, const linalg::matrix< interval > *& *__v*, const linalg::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 1132 of file datamap.hpp.

10.36.3.76 bool coco::datamap::retrieve_i (const char * *__s*, int *i*, const linalg::matrix< int > *&* *__v*, const linalg::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 1121 of file datamap.hpp.

10.36.3.77 bool coco::datamap::retrieve_i (const char * *__s*, int *i*, const linalg::matrix< double > *&* *__v*, const linalg::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 1110 of file datamap.hpp.

10.36.3.78 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 1100 of file datamap.hpp.

10.36.3.79 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 1090 of file datamap.hpp.

10.36.3.80 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 1080 of file datamap.hpp.

10.36.3.81 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 1070 of file datamap.hpp.

10.36.3.82 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 1060 of file datamap.hpp.

10.36.3.83 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 1050 of file datamap.hpp.

10.36.3.84 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 1040 of file datamap.hpp.

10.36.3.85 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 1030 of file datamap.hpp.

10.36.3.86 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 1020 of file datamap.hpp.

10.36.3.87 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 1010 of file datamap.hpp.

10.36.3.88 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 1000 of file datamap.hpp.

10.36.3.89 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 990 of file datamap.hpp.

10.36.3.90 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::matrix< std::string > *& __v, const linalg::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 975 of file datamap.hpp.

10.36.3.91 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::matrix< interval > *& __v, const linalg::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 955 of file datamap.hpp.

10.36.3.92 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::matrix< int > *& __v, const linalg::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 935 of file datamap.hpp.

10.36.3.93 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::matrix< double > *& __v, const linalg::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 915 of file datamap.hpp.

10.36.3.94 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 895 of file datamap.hpp.

10.36.3.95 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 875 of file datamap.hpp.

10.36.3.96 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 855 of file datamap.hpp.

10.36.3.97 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 835 of file datamap.hpp.

10.36.3.98 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 815 of file datamap.hpp.

10.36.3.99 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 795 of file datamap.hpp.

10.36.3.100 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 775 of file datamap.hpp.

10.36.3.101 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 755 of file datamap.hpp.

10.36.3.102 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 735 of file datamap.hpp.

10.36.3.103 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 715 of file datamap.hpp.

10.36.3.104 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 695 of file datamap.hpp.

10.36.3.105 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 675 of file datamap.hpp.

10.36.3.106 bool coco::datamap::retrieve_i (const char * *__s*, int *i*, const linalg::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 1138 of file datamap.hpp.

10.36.3.107 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1127 of file datamap.hpp.

10.36.3.108 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1116 of file datamap.hpp.

10.36.3.109 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1105 of file datamap.hpp.

10.36.3.110 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 1095 of file datamap.hpp.

10.36.3.111 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 1085 of file datamap.hpp.

10.36.3.112 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 1075 of file datamap.hpp.

10.36.3.113 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 1065 of file datamap.hpp.

10.36.3.114 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 1055 of file datamap.hpp.

10.36.3.115 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 1045 of file datamap.hpp.

10.36.3.116 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 1035 of file datamap.hpp.

10.36.3.117 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 1025 of file datamap.hpp.

10.36.3.118 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 1015 of file datamap.hpp.

10.36.3.119 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 1005 of file datamap.hpp.

10.36.3.120 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 995 of file datamap.hpp.

10.36.3.121 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 985 of file datamap.hpp.

10.36.3.122 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 964 of file datamap.hpp.

10.36.3.123 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 944 of file datamap.hpp.

10.36.3.124 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 924 of file datamap.hpp.

10.36.3.125 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 904 of file datamap.hpp.

10.36.3.126 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 884 of file datamap.hpp.

10.36.3.127 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 864 of file datamap.hpp.

10.36.3.128 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 844 of file datamap.hpp.

10.36.3.129 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 824 of file datamap.hpp.

10.36.3.130 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 804 of file datamap.hpp.

10.36.3.131 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 784 of file datamap.hpp.

10.36.3.132 bool coco::datamap::retrieve_i (const std::string & __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 764 of file datamap.hpp.

10.36.3.133 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 744 of file datamap.hpp.

10.36.3.134 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 724 of file datamap.hpp.

10.36.3.135 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 704 of file datamap.hpp.

10.36.3.136 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 684 of file datamap.hpp.

10.36.3.137 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 664 of file datamap.hpp.

10.36.3.138 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.

10.36.3.139 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.

10.36.3.140 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.

10.36.3.141 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.

10.36.3.142 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.

10.36.3.143 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.

10.36.3.144 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.

10.36.3.145 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 34 of file datamap.cc.

10.36.3.146 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.

10.36.3.147 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.

The documentation for this class was generated from the following files:

- datamap.h
- datamap.cc
- datamap.hpp

10.37 coco::dbccmp_false< _TR > Struct Template Reference

```
#include <dbcompare.hpp>
```

Public Member Functions

- dbccmp_false ()
- ~dbccmp_false ()
- bool operator() (const _TR &x, const _TR &y)

10.37.1 Detailed Description

template<typename _TR> struct coco::dbccmp_false< _TR >

Definition at line 229 of file dbcompare.hpp.

10.37.2 Constructor & Destructor Documentation

10.37.2.1 template<typename _TR > coco::dbccmp_false< _TR >::dbccmp_false () [inline]

Definition at line 231 of file dbcompare.hpp.

10.37.2.2 template<typename _TR > coco::dbccmp_false< _TR >::~dbccmp_false () [inline]

Definition at line 232 of file dbcompare.hpp.

10.37.3 Member Function Documentation

10.37.3.1 template<typename _TR > bool coco::dbccmp_false< _TR >::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

10.38 coco::dbccmp_true< _TR > Struct Template Reference

`#include <dbcompare.hpp>`

Public Member Functions

- dbccmp_true ()
- ~dbccmp_true ()
- bool operator() (const _TR &x, const _TR &y)

10.38.1 Detailed Description

template<typename _TR> struct coco::dbccmp_true< _TR >

Definition at line 220 of file dbcompare.hpp.

10.38.2 Constructor & Destructor Documentation

10.38.2.1 template<typename _TR > coco::dbccmp_true< _TR >::dbccmp_true () [inline]

Definition at line 222 of file dbcompare.hpp.

10.38.2.2 template<typename _TR > coco::dbccmp_true< _TR >::~dbccmp_true () [inline]

Definition at line 223 of file dbcompare.hpp.

10.38.3 Member Function Documentation

10.38.3.1 template<typename _TR > bool coco::dbccmp_true< _TR >::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

10.39 coco::dbccmps_absgt< _TR > Class Template 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)

10.39.1 Detailed Description

template<typename _TR> class coco::dbccmps_absgt< _TR >

Definition at line 139 of file dbcompare.hpp.

10.39.2 Constructor & Destructor Documentation

10.39.2.1 template<typename _TR> coco::dbccmps_absgt< _TR >::dbccmps_absgt (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 144 of file dbcompare.hpp.

10.39.2.2 template<typename _TR> coco::dbccmps_absgt< _TR >::~dbccmps_absgt () [inline]

Definition at line 146 of file dbcompare.hpp.

10.39.3 Member Function Documentation

10.39.3.1 template<typename _TR> bool coco::dbccmps_absgt< _TR >::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

10.40 coco::dbccmps_abslt< _TR > Class Template 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)

10.40.1 Detailed Description

template<typename _TR> class coco::dbccmps_abslt< _TR >

Definition at line 170 of file dbcompare.hpp.

10.40.2 Constructor & Destructor Documentation

10.40.2.1 template<typename _TR> coco::dbccmps_abslt< _TR >::dbccmps_abslt (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 175 of file dbcompare.hpp.

10.40.2.2 template<typename _TR> coco::dbccmps_abslt< _TR >::~dbccmps_abslt () [inline]

Definition at line 177 of file dbcompare.hpp.

10.40.3 Member Function Documentation

10.40.3.1 template<typename _TR> bool coco::dbccmps_abslt< _TR >::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

10.41 coco::dbccmps_false< _TR > Class Template 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)

10.41.1 Detailed Description

template<typename _TR> class coco::dbccmps_false< _TR >

Definition at line 203 of file dbcompare.hpp.

10.41.2 Constructor & Destructor Documentation

10.41.2.1 template<typename _TR> coco::dbccmps_false< _TR >::dbccmps_false (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 208 of file dbcompare.hpp.

10.41.2.2 template<typename _TR> coco::dbccmps_false< _TR >::~dbccmps_false () [inline]

Definition at line 210 of file dbcompare.hpp.

10.41.3 Member Function Documentation

10.41.3.1 template<typename _TR> bool coco::dbccmps_false< _TR >::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

10.42 coco::dbccmps_gt< _TR > Class Template 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)

10.42.1 Detailed Description

template<typename _TR> class coco::dbccmps_gt< _TR >

Definition at line 124 of file dbcompare.hpp.

10.42.2 Constructor & Destructor Documentation

10.42.2.1 template<typename _TR> coco::dbccmps_gt< _TR >::dbccmps_gt (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 129 of file dbcompare.hpp.

10.42.2.2 template<typename _TR> coco::dbccmps_gt< _TR >::~dbccmps_gt () [inline]

Definition at line 131 of file dbcompare.hpp.

10.42.3 Member Function Documentation

10.42.3.1 template<typename _TR> bool coco::dbccmps_gt< _TR >::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

10.43 coco::dbccmps_lt< _TR > Class Template 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)

10.43.1 Detailed Description

template<typename _TR> class coco::dbccmps_lt< _TR >

Definition at line 155 of file dbcompare.hpp.

10.43.2 Constructor & Destructor Documentation

10.43.2.1 template<typename _TR> coco::dbccmps_lt< _TR >::dbccmps_lt (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 160 of file dbcompare.hpp.

10.43.2.2 template<typename _TR> coco::dbccmps_lt< _TR >::~dbccmps_lt () [inline]

Definition at line 162 of file dbcompare.hpp.

10.43.3 Member Function Documentation

10.43.3.1 template<typename _TR> bool coco::dbccmps_lt< _TR >::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

10.44 coco::dbccmps_true< _TR > Class Template 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)

10.44.1 Detailed Description

template<typename _TR> class coco::dbccmps_true< _TR >

Definition at line 186 of file dbcompare.hpp.

10.44.2 Constructor & Destructor Documentation

10.44.2.1 template<typename _TR> coco::dbccmps_true< _TR >::dbccmps_true (double *_rel* = 1.0, double *_abs* = 0.0) [inline]

Definition at line 191 of file dbcompare.hpp.

10.44.2.2 template<typename _TR> coco::dbccmps_true< _TR >::~dbccmps_true () [inline]

Definition at line 193 of file dbcompare.hpp.

10.44.3 Member Function Documentation

10.44.3.1 template<typename _TR> bool coco::dbccmps_true< _TR >::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

10.45 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)

10.45.1 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 42 of file dbbasic.h.

10.45.2 Constructor & Destructor Documentation

10.45.2.1 coco::dbt_row::dbt_row () [inline]

Standard Constructor

Definition at line 51 of file dbbasic.h.

10.45.2.2 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.

10.45.2.3 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.

10.45.2.4 coco::dbt_row::dbt_row (const dbt_row & *dr*) [inline]

Standard Copy Constructor

Definition at line 62 of file dbbasic.h.

10.45.2.5 coco::dbt_row::~dbt_row () [inline]

Standard Destructor

Definition at line 65 of file dbbasic.h.

10.45.3 Member Function Documentation

10.45.3.1 template<class _TC > void 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.

10.45.3.2 template<class _TC > void 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.

10.45.3.3 _Base& coco::dbt_row::row () [inline]

Return the row contents of the dbt_row.

Definition at line 71 of file dbbasic.h.

10.45.3.4 const _Base& coco::dbt_row::row () const [inline]

Return the row contents of the dbt_row.

Definition at line 68 of file dbbasic.h.

The documentation for this class was generated from the following file:

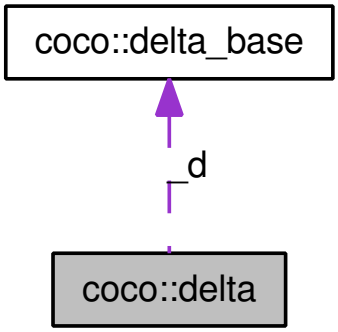
- dbbasic.h

10.46 coco::delta Class Reference

The delta class (updates to work nodes).

`#include <api_deltabase.h>`

Collaboration diagram for coco::delta:



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.

10.46.1 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 api_deltabase.h.

10.46.2 Constructor & Destructor Documentation

10.46.2.1 coco::delta::delta () [inline]

Standard Constructor

Definition at line 72 of file api_deltabase.h.

10.46.2.2 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.

10.46.2.3 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.

10.46.2.4 coco::delta::~delta () [inline]

Destructor, which frees the `_d` using `delete`

Definition at line 120 of file api_delta.h.

10.46.3 Member Function Documentation

10.46.3.1 bool 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.

10.46.3.2 bool 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 159 of file api_delta.h.

10.46.3.3 void 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 180 of file api_delta.h.

10.46.3.4 const std::string & 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.

10.46.3.5 const delta_base * coco::delta::get_base () const [inline]

Return the delta_base stored in this wrapper

Definition at line 137 of file api_delta.h.

10.46.3.6 bool coco::delta::operator!= (const delta & *_c*) const [inline]

Definition at line 193 of file api_delta.h.

10.46.3.7 delta & 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.

10.46.3.8 bool coco::delta::operator== (const delta & _c) const [inline]

Comparison operators

Definition at line 190 of file api_delta.h.

10.46.3.9 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.

Definition at line 36 of file delta.cc.

10.46.3.10 void 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 188 of file api_delta.h.

10.46.4 Friends And Related Function Documentation

10.46.4.1 friend class delta_base [friend]

Definition at line 127 of file api_deltabase.h.

10.46.4.2 friend class ie_return_type [friend]

Definition at line 128 of file api_deltabase.h.

10.46.4.3 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 files:

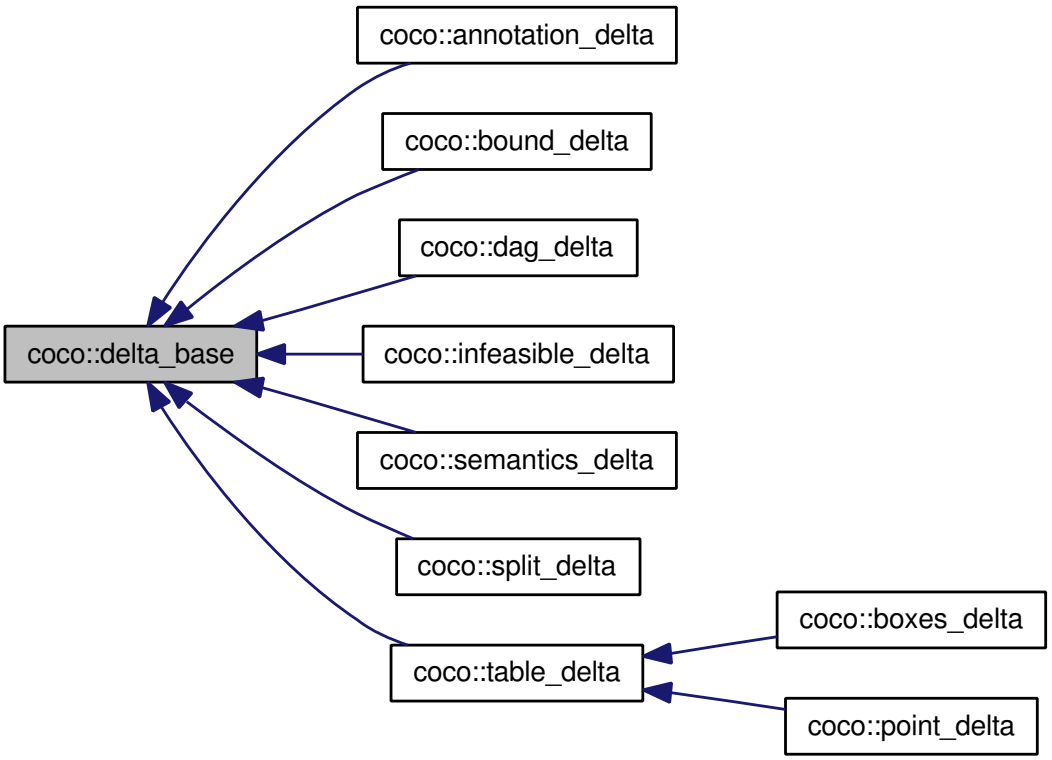
- api_deltabase.h
- api_delta.h
- delta.cc

10.47 coco::delta_base Class Reference

Base class for the deltas.

`#include <api_deltabase.h>`

Inheritance diagram for 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 &_d) const
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Protected Attributes

- std::string _action

10.47.1 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 api_deltabase.h.

10.47.2 Constructor & Destructor Documentation

10.47.2.1 coco::delta_base::delta_base () [inline]

Standard Constructor

Definition at line 158 of file api_deltabase.h.

10.47.2.2 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.

10.47.2.3 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.

10.47.2.4 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.

10.47.3 Member Function Documentation

10.47.3.1 virtual bool coco::delta_base::apply (work_node & _x, undelta_base *& _u, const delta_id & _d) 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.

Reimplemented in coco::annotation_delta, coco::bound_delta, coco::boxes_delta, coco::dag_delta, coco::infeasible_delta, coco::point_delta, coco::semantics_delta, coco::split_delta, and coco::table_delta.

Definition at line 198 of file api_deltabase.h.

10.47.3.2 bool coco::delta_base::apply3 (work_node & _x, const work_node & _y, undelta_base *& _u, const delta_id & _d) 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.

10.47.3.3 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.

Reimplemented in coco::table_delta.

Definition at line 189 of file api_deltabase.h.

10.47.3.4 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.

10.47.3.5 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.

10.47.3.6 virtual delta_base* coco::delta_base::new_copy () const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::annotation_delta, coco::bound_delta, coco::boxes_delta, coco::dag_delta, coco::infeasible_delta, coco::point_delta, coco::semantics_delta, coco::split_delta, and coco::table_delta.

Definition at line 167 of file api_deltabase.h.

10.47.3.7 virtual void 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 api_deltabase.h.

10.47.4 Member Data Documentation

10.47.4.1 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 files:

- api_deltabase.h
- api_delta.h

10.48 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)

10.48.1 Detailed Description

A database method returning the delta action specifier, used as a stored procedure.

Definition at line 54 of file api_delta.h.

10.48.2 Member Typedef Documentation

10.48.2.1 typedef vdbl::context coco::delta_get_action::context

The evaluation context class

Definition at line 66 of file api_delta.h.

10.48.2.2 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.

10.48.3 Constructor & Destructor Documentation

10.48.3.1 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.

10.48.3.2 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.

10.48.3.3 virtual coco::delta_get_action::~delta_get_action () [inline, virtual]

Standard Destructor

Definition at line 76 of file api_delta.h.

10.48.4 Member Function Documentation

10.48.4.1 const std::string& coco::delta_get_action::def () const [inline]

Default value of the procedure

Definition at line 81 of file api_delta.h.

10.48.4.2 const std::string & coco::delta_get_action::operator() () const [inline]

Result of the procedure

Definition at line 98 of file api_delta.h.

10.48.4.3 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:

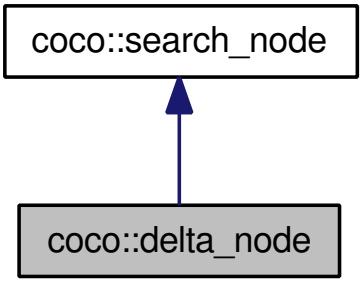
- api_delta.h

10.49 coco::delta_node Class Reference

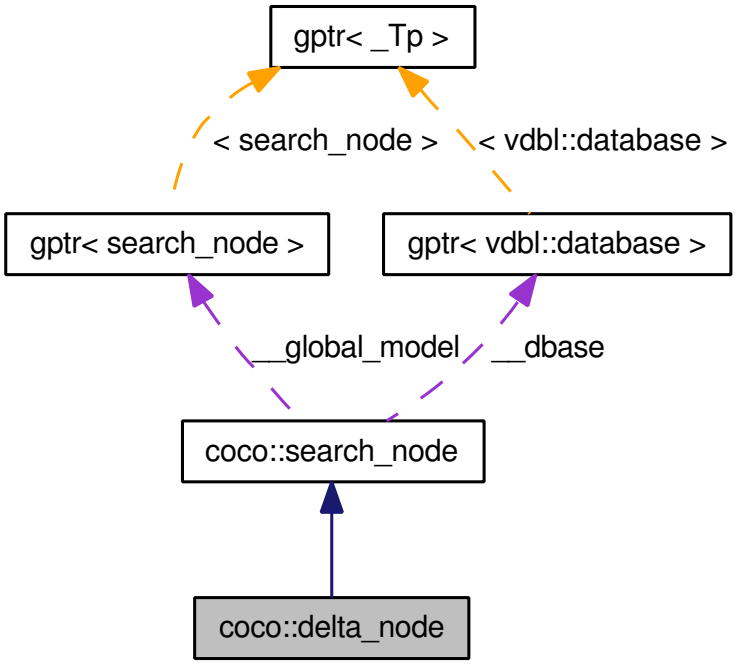
Class holding the delta nodes in the search graph.

`#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

10.49.1 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

Definition at line 210 of file search_node.h.

10.49.2 Constructor & Destructor Documentation

10.49.2.1 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 227 of file search_node.h.

10.49.2.2 virtual coco::delta_node::~delta_node () [inline, virtual]

Standard Destructor

Definition at line 235 of file search_node.h.

10.49.3 Member Function Documentation

10.49.3.1 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 261 of file search_node.h.

10.49.3.2 gptr<vdbl::database>* coco::search_node::database () const [inline, inherited]

This is the accessor method for the search database.

Definition at line 164 of file search_node.h.

10.49.3.3 vdbl::userid coco::search_node::get_dbuserid () const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 158 of file search_node.h.

10.49.3.4 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 408 of file search_node.hpp.

10.49.3.5 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 394 of file search_node.hpp.

10.49.3.6 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 249 of file search_node.h.

10.49.3.7 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 267 of file search_node.h.

10.49.3.8 search_node_id coco::search_node::get_id () const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 167 of file search_node.h.

10.49.3.9 vdbl::rowid coco::search_node::get_rowid () const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 173 of file search_node.h.

10.49.3.10 gptr<search_node>* coco::search_node::global_model () const [inline, inherited]

This is the accessor method for the global model.

Definition at line 161 of file search_node.h.

10.49.3.11 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::search_node.

Definition at line 243 of file search_node.h.

10.49.3.12 void 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 184 of file search_node.h.

10.49.3.13 void 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 180 of file search_node.h.

10.49.3.14 unsigned int coco::delta_node::n_deltas () const [inline]

This method returns the number of deltas stored in this delta node.

Definition at line 246 of file search_node.h.

10.49.3.15 delta_node & coco::delta_node::operator= (const delta_node & *__w*) [inline]

Standard Assignment Operator

Definition at line 197 of file search_node.hpp.

10.49.3.16 search_node_relation 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 117 of file search_node.h.

10.49.3.17 void 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 170 of file search_node.h.

10.49.3.18 void 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 176 of file search_node.h.

10.49.3.19 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.

10.49.3.20 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.

10.49.4 Friends And Related Function Documentation

10.49.4.1 friend class search_graph [friend, inherited]

Definition at line 195 of file search_node.h.

10.49.5 Member Data Documentation

10.49.5.1 gptr<vdbl::database>* 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 96 of file search_node.h.

10.49.5.2 gptr<search_node>* 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 93 of file search_node.h.

10.49.5.3 vdbl::userid 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 99 of file search_node.h.

10.49.5.4 search_node_id coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 104 of file search_node.h.

10.49.5.5 std::vector<annotation> 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 108 of file search_node.h.

10.49.5.6 vdbl::rowid 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 112 of file search_node.h.

10.49.5.7 search_node_relation 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 102 of file search_node.h.

The documentation for this class was generated from the following files:

- search_node.h
- search_node.hpp

10.50 coco::der_cache Struct Reference

Cache data for der_eval.

`#include <der_evaluator_n.h>`

Public Attributes

- std::vector< double > * d
- bool v

10.50.1 Detailed Description

This class is the cache data type for the der_eval evaluator.

Definition at line 982 of file der_evaluator_n.h.

10.50.2 Member Data Documentation

10.50.2.1 std::vector<double>* coco::der_cache::d

the derivative information

Definition at line 984 of file der_evaluator_n.h.

10.50.2.2 bool coco::der_cache::v

whether we have already visited

Definition at line 985 of file der_evaluator_n.h.

The documentation for this struct was generated from the following file:

- der_evaluator_n.h

10.51 coco::der_eval Class Reference

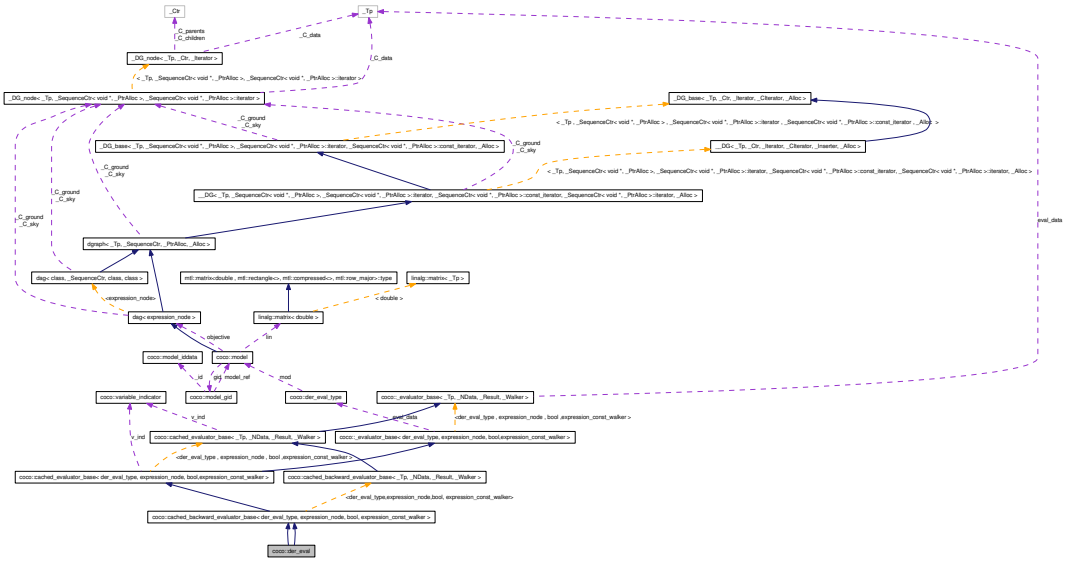
Backward gradient evaluation with prepared derivative data.

`#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
- typedef der_eval_type data_type

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)
- der_eval (std::vector< func_cache > &__func_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< func_cache > &__func_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 ()

- 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)

Protected Attributes

- const variable_indicator * v_ind
- der_eval_type eval_data

10.51.1 Detailed Description

Backward gradient evaluation including preparation of derivative data.

This class is a cached backward evaluator performing a gradient evaluation with available partial derivative information.

Definition at line 1002 of file der_evaluator.h.

10.51.2 Member Typedef Documentation

10.51.2.1 typedef _Base::const_walker coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 853 of file evaluator.h.

10.51.2.2 typedef der_eval_type coco::_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.51.2.3 typedef _Base::node_data_type coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 849 of file evaluator.h.

10.51.2.4 typedef _Base::return_value coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 851 of file evaluator.h.

10.51.3 Constructor & Destructor Documentation

10.51.3.1 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 1035 of file der_evaluator.h.

10.51.3.2 coco::der_eval::der_eval (const der_eval & *__d*) [inline]

Standard Copy Constructor

Definition at line 1049 of file der_evaluator.h.

10.51.3.3 coco::der_eval::~der_eval () [inline]

Standard Destructor

Definition at line 1052 of file der_evaluator.h.

10.51.3.4 coco::der_eval::der_eval (std::vector< func_cache > & *__func_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 1036 of file der_evaluator_n.h.

10.51.3.5 coco::der_eval::der_eval (const der_eval & *__d*) [inline]

Standard Copy Constructor

Definition at line 1050 of file der_evaluator_n.h.

10.51.3.6 coco::der_eval::~der_eval () [inline]

Standard Destructor

Definition at line 1053 of file der_evaluator_n.h.

10.51.4 Member Function Documentation

10.51.4.1 int coco::der_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1095 of file der_evaluator_n.h.

10.51.4.2 int coco::der_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1090 of file der_evaluator.h.

10.51.4.3 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::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1775 of file der_evaluator_n.h.

10.51.4.4 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::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1182 of file der_evaluator.h.

10.51.4.5 void coco::der_eval::cleanup (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1731 of file der_evaluator_n.h.

10.51.4.6 void coco::der_eval::cleanup (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1138 of file der_evaluator.h.

10.51.4.7 int coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 879 of file evaluator.h.

10.51.4.8 void coco::der_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1089 of file der_evaluator_n.h.

10.51.4.9 void coco::der_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1084 of file der_evaluator.h.

10.51.4.10 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::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1022 of file der_evaluator_n.h.

10.51.4.11 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::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1014 of file der_evaluator.h.

10.51.4.12 void coco::der_eval::new_point (std::vector< func_cache > & *__func_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 1058 of file der_evaluator_n.h.

10.51.4.13 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 1057 of file der_evaluator.h.

10.51.4.14 void coco::der_eval::new_result (std::vector< double > & *__grad*) [inline]

This method changes the result vector to __grad.

Definition at line 1069 of file der_evaluator_n.h.

10.51.4.15 void coco::der_eval::new_result (std::vector< double > & *__grad*) [inline]

This method changes the result vector to __grad.

Definition at line 1065 of file der_evaluator.h.

10.51.4.16 void coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::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. It is translated into a call to cleanup.

Reimplemented from coco::_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 874 of file evaluator.h.

10.51.4.17 int coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Reimplemented from coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 862 of file evaluator.h.

10.51.4.18 void coco::der_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1742 of file der_evaluator_n.h.

10.51.4.19 void coco::der_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1149 of file der_evaluator.h.

10.51.4.20 void coco::der_eval::set_mult (double *scal*) [inline]

This method causes the gradient to be multiplied by `scal`.

Definition at line 1075 of file der_evaluator_n.h.

10.51.4.21 void coco::der_eval::set_mult (double *scal*) [inline]

This method causes the gradient to be multiplied by `scal`.

Definition at line 1071 of file der_evaluator.h.

10.51.4.22 expression_const_walker coco::der_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 1083 of file der_evaluator_n.h.

10.51.4.23 expression_const_walker coco::der_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 1078 of file der_evaluator.h.

10.51.4.24 int coco::der_eval::update (const expression_node & *__data*, const bool & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1756 of file der_evaluator_n.h.

10.51.4.25 int coco::der_eval::update (const bool & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1749 of file der_evaluator_n.h.

10.51.4.26 int coco::der_eval::update (const expression_node & *__data*, const bool & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1163 of file der_evaluator.h.

10.51.4.27 int coco::der_eval::update (const bool & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< der_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1156 of file der_evaluator.h.

10.51.4.28 return_value coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 891 of file evaluator.h.

10.51.4.29 int coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 885 of file evaluator.h.

10.51.4.30 void coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::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 900 of file evaluator.h.

10.51.4.31 return_value coco::cached_backward_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< der_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 896 of file evaluator.h.

10.51.5 Member Data Documentation

10.51.5.1 der_eval_type coco::_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.51.5.2 const variable_indicator* coco::cached_evaluator_base< der_eval_type , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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 files:

- der_evaluator.h
- der_evaluator_n.h

10.52 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

10.52.1 Detailed Description

This class is the visitor data type for the der_eval evaluator.

Definition at line 985 of file der_evaluator.h.

10.52.2 Member Data Documentation

10.52.2.1 unsigned int coco::der_eval_type::child_n

children counter

Definition at line 993 of file der_evaluator.h.

10.52.2.2 std::vector<std::vector<double> >* coco::der_eval_type::d_cache

derivative cache

Definition at line 988 of file der_evaluator.h.

10.52.2.3 std::vector<std::vector<double> >* coco::der_eval_type::d_data

partial derivative data

Definition at line 987 of file der_evaluator.h.

10.52.2.4 std::vector<double>* coco::der_eval_type::grad_vec

the gradient vector

Definition at line 989 of file der_evaluator.h.

10.52.2.5 const model* coco::der_eval_type::mod

the DAG

Definition at line 990 of file der_evaluator.h.

10.52.2.6 double coco::der_eval_type::mult

the current value on the path

Definition at line 991 of file der_evaluator.h.

10.52.2.7 double coco::der_eval_type::mult_trans

transfer variable for mult

Definition at line 992 of file der_evaluator.h.

The documentation for this struct was generated from the following file:

- der_evaluator.h

10.53 coco::dfunc_eval< _T, DN > Class Template Reference

`#include <dfunc_evaluator.h>`

Inheritance diagram for coco::dfunc_eval< _T, DN >:



Collaboration diagram for coco::dfunc_eval< _T, DN >:



Public Types

- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker
- typedef dfunc_eval_type< _T, DN > data_type

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 bool is_cached (const node_data_type &__data)
- virtual void cleanup (const node_data_type &__data)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- dfunc_eval_type< _T, DN > eval_data

10.53.1 Detailed Description

template<typename _T, int DN> class coco::dfunc_eval< _T, DN >

Definition at line 68 of file dfunc_evaluator.h.

10.53.2 Member Typedef Documentation

10.53.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.53.2.2 typedef dfunc_eval_type< _T, DN > coco::_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.53.2.3 typedef _Base::return_value coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.53.3 Constructor & Destructor Documentation

10.53.3.1 template<typename _T , int DN> coco::dfunc_eval< _T, DN >::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.

10.53.3.2 template<typename _T , int DN> coco::dfunc_eval< _T, DN >::dfunc_eval (const dfunc_eval< _T, DN > & *__v*) [inline]

Definition at line 146 of file dfunc_evaluator.h.

10.53.3.3 template<typename _T , int DN> coco::dfunc_eval< _T, DN >::~dfunc_eval () [inline]

Definition at line 148 of file dfunc_evaluator.h.

10.53.4 Member Function Documentation

10.53.4.1 template<typename _T , int DN> void coco::dfunc_eval< _T, DN >::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 238 of file dfunc_evaluator.h.

10.53.4.2 template<typename _T , int DN> retval coco::dfunc_eval< _T, DN >::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::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 649 of file dfunc_evaluator.h.

10.53.4.3 virtual void coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::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 804 of file evaluator.h.

10.53.4.4 int coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.53.4.5 template<typename _T , int DN> int coco::dfunc_eval< _T, DN >::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 162 of file dfunc_evaluator.h.

10.53.4.6 template<typename _T , int DN> void coco::dfunc_eval< _T, DN >::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from 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.

10.53.4.7 virtual bool coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::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 776 of file evaluator.h.

10.53.4.8 template<typename _T , int DN> bool coco::dfunc_eval< _T, DN >::is_cached (const node_data_type & *__data*) [inline, protected]

Definition at line 80 of file dfunc_evaluator.h.

10.53.4.9 template<typename _T , int DN> void coco::dfunc_eval< _T, DN >::new_point (const _T & *__x*, const variable_indicator & *__v*) [inline]

Definition at line 154 of file dfunc_evaluator.h.

10.53.4.10 void coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.53.4.11 int coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.53.4.12 template<typename _T , int DN> void coco::dfunc_eval< _T, DN >::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 249 of file dfunc_evaluator.h.

10.53.4.13 template<typename _T , int DN> expression_const_walker coco::dfunc_eval< _T, DN >::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 150 of file dfunc_evaluator.h.

10.53.4.14 template<typename _T , int DN> int coco::dfunc_eval< _T, DN >::update (const expression_node & *__data*, const retval & *__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.

Reimplemented from coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 260 of file dfunc_evaluator.h.

10.53.4.15 template<typename _T , int DN> int coco::dfunc_eval< _T, DN >::update (const retval & *__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.

Reimplemented from coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >, expression_const_walker >.

Definition at line 254 of file dfunc_evaluator.h.

10.53.4.16 return_value coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.53.4.17 int coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.53.4.18 void coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::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 771 of file evaluator.h.

10.53.4.19 return_value coco::cached_forward_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< dfunc_eval_type< _T, DN >, expression_node, dfunc_eval_rettype< _T, DN >,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.53.5 Member Data Documentation

10.53.5.1 dfunc_eval_type< _T, DN > coco::_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.53.5.2 const variable_indicator* coco::cached_evaluator_base< dfunc_eval_type< _T, DN > , expression_node , dfunc_eval_rettype< _T, DN > , expression_const_walker >::v_ind [protected, inherited]

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:

- dfunc_evaluator.h

10.54 coco::dfunc_eval_rettype< _T, DN > Struct Template Reference

`#include <dfunc_evaluator.h>`

Public Attributes

- _T f [DN+1]

10.54.1 Detailed Description

template<typename _T, int DN> struct coco::dfunc_eval_rettype< _T, DN >

Definition at line 49 of file dfunc_evaluator.h.

10.54.2 Member Data Documentation

10.54.2.1 template<typename _T , int DN> _T coco::dfunc_eval_rettype< _T, DN >::f[DN+1]

Definition at line 51 of file dfunc_evaluator.h.

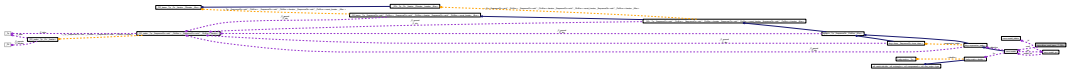
The documentation for this struct was generated from the following file:

- dfunc_evaluator.h

10.55 coco::dfunc_eval_type< _T, DN > Struct Template Reference

`#include <dfunc_evaluator.h>`

Collaboration diagram for coco::dfunc_eval_type< _T, DN >:



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

10.55.1 Detailed Description

template<typename _T, int DN> struct coco::dfunc_eval_type< _T, DN >

Definition at line 55 of file dfunc_evaluator.h.

10.55.2 Member Data Documentation

10.55.2.1 template<typename _T , int DN> unsigned int coco::dfunc_eval_type< _T, DN >::_0chnbase

Definition at line 57 of file dfunc_evaluator.h.

10.55.2.2 template<typename _T , int DN> std::vector<dfunc_eval_rettype<_T, DN> >* coco::dfunc_eval_type< _T, DN >::cache

Definition at line 59 of file dfunc_evaluator.h.

10.55.2.3 template<typename _T , int DN> _T coco::dfunc_eval_type< _T, DN >::d

Definition at line 62 of file dfunc_evaluator.h.

10.55.2.4 template<typename _T , int DN> const model* coco::dfunc_eval_type< _T, DN >::mod

Definition at line 60 of file dfunc_evaluator.h.

10.55.2.5 template<typename _T , int DN> unsigned int coco::dfunc_eval_type< _T, DN >::n

Definition at line 64 of file dfunc_evaluator.h.

10.55.2.6 template<typename _T , int DN> void* coco::dfunc_eval_type< _T, DN >::p

Definition at line 61 of file dfunc_evaluator.h.

10.55.2.7 template<typename _T , int DN> dfunc_eval_rettype<_T, DN> coco::dfunc_eval_type< _T, DN >::r

Definition at line 63 of file dfunc_evaluator.h.

10.55.2.8 template<typename _T , int DN> _T coco::dfunc_eval_type< _T, DN >::x

Definition at line 58 of file dfunc_evaluator.h.

The documentation for this struct was generated from the following file:

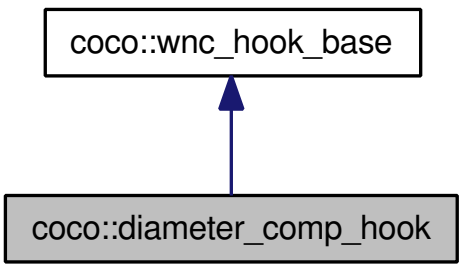
- dfunc_evaluator.h

10.56 coco::diameter_comp_hook Class Reference

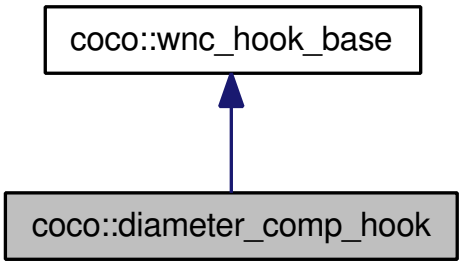
The log-volume computation hook (work node computation hook).

`#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

10.56.1 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”.

Definition at line 42 of file diameter_hook.h.

10.56.2 Constructor & Destructor Documentation

10.56.2.1 coco::diameter_comp_hook::diameter_comp_hook () [inline]

Standard Constructor

Definition at line 46 of file diameter_hook.h.

10.56.2.2 virtual coco::diameter_comp_hook::~diameter_comp_hook () [inline, virtual]

Standard Destructor

Definition at line 49 of file diameter_hook.h.

10.56.3 Member Function Documentation

10.56.3.1 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.

10.56.3.2 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.

10.56.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *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 238 of file comp_hook.h.

10.56.3.4 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.

10.56.3.5 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.

10.56.3.6 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.

10.56.3.7 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.

10.56.3.8 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.

10.56.3.9 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.

10.56.3.10 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.

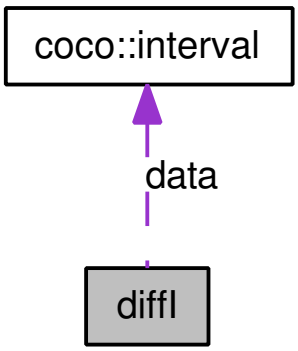
The documentation for this class was generated from the following file:

- diameter_hook.h

10.57 diffI Class Reference

`#include <diffI.h>`

Collaboration diagram for diffI:



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)
- I & operator[ ] (int pos) const
- void setDeg (int n)

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

10.57.1 Detailed Description

Definition at line 19 of file diffI.h.

10.57.2 Constructor & Destructor Documentation

10.57.2.1 diffI::diffI (void) [inline]

Definition at line 89 of file diffI.h.

10.57.2.2 diffI::diffI (double *a*) [inline]

Definition at line 103 of file diffI.h.

10.57.2.3 diffI::diffI (double *a*, double *b*) [inline]

Definition at line 111 of file diffI.h.

10.57.2.4 diffI::diffI (int *n*) [inline]

Definition at line 96 of file diffI.h.

10.57.2.5 diffI::diffI (double *a*, int *n*) [inline]

Definition at line 119 of file diffI.h.

10.57.2.6 diffI::diffI (double *a*, double *b*, int *n*) [inline]

Definition at line 127 of file diffI.h.

10.57.2.7 diffI::diffI (const I & *a*, int *n*) [inline]

Definition at line 135 of file diffI.h.

10.57.2.8 diffI::diffI (const diffI & *a*) [inline]

Definition at line 143 of file diffI.h.

10.57.2.9 diffI::~diffI () [inline, virtual]

Definition at line 650 of file diffI.h.

10.57.3 Member Function Documentation

10.57.3.1 diffI diffI::getVar (const I & *x*, int *n*) [inline, static]

Definition at line 208 of file diffI.h.

10.57.3.2 diffI diffI::getVar (const double & *x*, const double & *y*, int *n*) [inline, static]

Definition at line 200 of file diffI.h.

10.57.3.3 diffI diffI::getVar (const double & *x*, int *n*) [inline, static]

Definition at line 192 of file diffI.h.

10.57.3.4 diffI diffI::getVar (const double & *x*, const double & *y*) [inline, static]

Definition at line 184 of file diffI.h.

10.57.3.5 diffI diffI::getVar (const double & *x*) [inline, static]

Definition at line 176 of file diffI.h.

10.57.3.6 diffI & diffI::operator= (const diffI & *a*) [inline]

Definition at line 150 of file diffI.h.

10.57.3.7 interval & diffI::operator[ ] (int *pos*) const [inline]

Definition at line 167 of file diffI.h.

10.57.3.8 void diffI::setDeg (int *n*) [inline]

Definition at line 216 of file diffI.h.

10.57.4 Member Data Documentation

10.57.4.1 int diffI::MAXDEG

Definition at line 27 of file diffI.h.

The documentation for this class was generated from the following file:

- diffI.h

10.58 coco::diffI_eval Class Reference

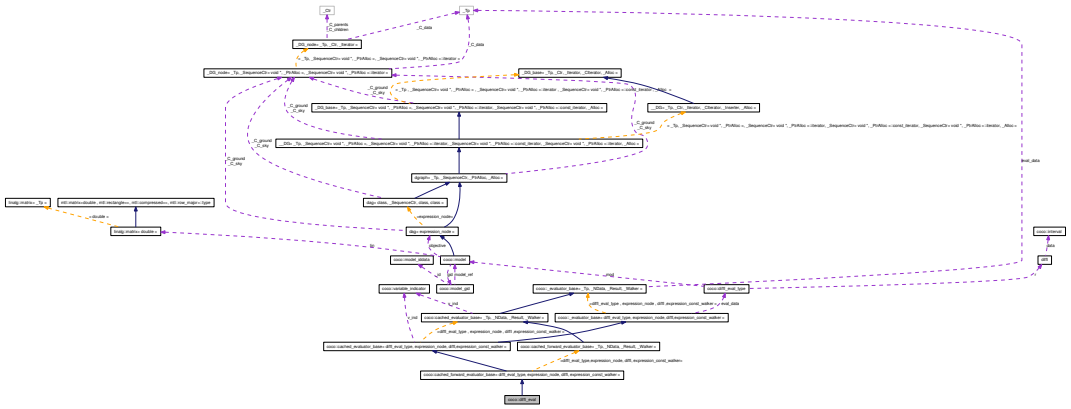
Forward function range evaluation.

`#include <diffI_evaluator.h>`

Inheritance diagram for coco::diffI_eval:



Collaboration diagram for coco::diffI_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker
- typedef diffI_eval_type data_type

Public Member Functions

- diffI_eval (const std::vector< diffI > &__x, const variable_indicator &__v, const model &__m, std::vector< diffI > *__c, int __maxd)
- diffI_eval (const diffI_eval &__v)
- ~diffI_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_interval (const std::vector< diffI > &__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 void cleanup (const node_data_type &__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 diffI &__rval)
  - int update (const expression_node &__data, const diffI &__rval)
  - diffI calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- diffI_eval_type eval_data

10.58.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural diffI extension.

Definition at line 75 of file diffI_evaluator.h.

10.58.2 Member Typedef Documentation

10.58.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_-const_walker >.

Definition at line 723 of file evaluator.h.

10.58.2.2 typedef diffI_eval_type coco::_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.58.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_-const_walker >.

Definition at line 719 of file evaluator.h.

10.58.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_-const_walker >.

Definition at line 721 of file evaluator.h.

10.58.3 Constructor & Destructor Documentation

10.58.3.1 coco::diffI_eval::diffI_eval (const std::vector< diffI > & *__x*, const variable_indicator & *__v*, const model & *__m*, std::vector< diffI > * *__c*, int *__maxd*) [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 diffI extension will use known node ranges for reducing the ranges.

Definition at line 143 of file diffI_evaluator.h.

10.58.3.2 coco::diffI_eval::diffI_eval (const diffI_eval & *__v*) [inline]

Standard Copy Constructor

Definition at line 158 of file diffI_evaluator.h.

10.58.3.3 coco::diffI_eval::~diffI_eval () [inline]

Standard Destructor

Definition at line 161 of file diffI_evaluator.h.

10.58.4 Member Function Documentation

10.58.4.1 void coco::diffI_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >.

Definition at line 244 of file diffI_evaluator.h.

10.58.4.2 diffI coco::diffI_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >.

Definition at line 697 of file diffI_evaluator.h.

10.58.4.3 virtual void coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::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 804 of file evaluator.h.

10.58.4.4 int coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.58.4.5 int coco::diffI_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >.

Definition at line 182 of file diffI_evaluator.h.

10.58.4.6 void coco::diffI_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >.

Definition at line 180 of file diffI_evaluator.h.

10.58.4.7 bool coco::diffI_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::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >.

Definition at line 88 of file diffI_evaluator.h.

10.58.4.8 void coco::diffI_eval::new_interval (const std::vector< diffI > & *__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 diffI_evaluator.h.

10.58.4.9 void coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_-walker >.

Definition at line 744 of file evaluator.h.

10.58.4.10 int coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_-const_walker >.

Definition at line 732 of file evaluator.h.

10.58.4.11 void coco::diffI_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >.

Definition at line 251 of file diffI_evaluator.h.

10.58.4.12 expression_const_walker coco::diffI_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< diffI_eval_type, expression_node, diffI,expression_-const_walker >.

Definition at line 164 of file diffI_evaluator.h.

10.58.4.13 int coco::diffI_eval::update (const expression_node & *__data*, const diffI & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >.

Definition at line 263 of file diffI_evaluator.h.

10.58.4.14 int coco::diffI_eval::update (const diffI & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffI_eval_type, expression_node, diffI, expression_const_walker >.

Definition at line 256 of file diffI_evaluator.h.

10.58.4.15 return_value coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.58.4.16 int coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.58.4.17 void coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::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 771 of file evaluator.h.

10.58.4.18 return_value coco::cached_forward_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< diffI_eval_type, expression_node, diffI,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.58.5 Member Data Documentation

10.58.5.1 diffI_eval_type coco::_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.58.5.2 const variable_indicator* coco::cached_evaluator_base< diffI_eval_type , expression_node , diffI , expression_const_walker >::v_ind [protected, inherited]

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:

- diffI_evaluator.h

10.59 coco::diffI_eval_type Struct Reference

Visitor data for diffI_eval.

`#include <diffI_evaluator.h>`

Collaboration diagram for coco::diffI_eval_type:



Public Attributes

- const std::vector< diffI > * x
- std::vector< diffI > * cache
- const model * mod
- void * p
- diffI r
- unsigned int n
- int maxdeg

10.59.1 Detailed Description

This class is the visitor data type for the diffI_eval evaluator.

Definition at line 57 of file diffI_evaluator.h.

10.59.2 Member Data Documentation

10.59.2.1 std::vector<diffI>* coco::diffI_eval_type::cache

function range the function range cache

Definition at line 60 of file diffI_evaluator.h.

10.59.2.2 int coco::diffI_eval_type::maxdeg

maximal degree of derivative

Definition at line 65 of file diffI_evaluator.h.

10.59.2.3 const model* coco::diffI_eval_type::mod

the DAG

Definition at line 61 of file diffI_evaluator.h.

10.59.2.4 unsigned int coco::diffI_eval_type::n

children counter

Definition at line 64 of file diffI_evaluator.h.

10.59.2.5 void* coco::diffI_eval_type::p

additional data for

Definition at line 62 of file diffI_evaluator.h.

10.59.2.6 diffI coco::diffI_eval_type::r

return value

Definition at line 63 of file diffI_evaluator.h.

10.59.2.7 const std::vector<diffI>* coco::diffI_eval_type::x

Definition at line 59 of file diffI_evaluator.h.

The documentation for this struct was generated from the following file:

- diffI_evaluator.h

10.60 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[ ] (int pos) const
- void setDeg (int n)

Static Public Member Functions

- static diffNumber getVar (const double &x)
- static diffNumber getVar (const double &x, int n)

Public Attributes

- int MAXDEG

10.60.1 Detailed Description

Definition at line 17 of file diffNumber.h.

10.60.2 Constructor & Destructor Documentation

10.60.2.1 diffNumber::diffNumber (void) [inline]

Definition at line 78 of file diffNumber.h.

10.60.2.2 diffNumber::diffNumber (double *a*) [inline]

Definition at line 95 of file diffNumber.h.

10.60.2.3 diffNumber::diffNumber (int *n*) [inline]

Definition at line 87 of file diffNumber.h.

10.60.2.4 diffNumber::diffNumber (double *a*, int *n*) [inline]

Definition at line 104 of file diffNumber.h.

10.60.2.5 diffNumber::diffNumber (const diffNumber & *a*) [inline]

Definition at line 113 of file diffNumber.h.

10.60.2.6 diffNumber::~diffNumber () [inline, virtual]

Definition at line 677 of file diffNumber.h.

10.60.3 Member Function Documentation

10.60.3.1 diffNumber diffNumber::getVar (const double & *x*, int *n*) [inline, static]

Definition at line 158 of file diffNumber.h.

10.60.3.2 diffNumber diffNumber::getVar (const double & *x*) [inline, static]

Definition at line 150 of file diffNumber.h.

10.60.3.3 diffNumber & diffNumber::operator= (const diffNumber & *a*) [inline]

Definition at line 121 of file diffNumber.h.

10.60.3.4 double & diffNumber::operator[ ] (int *pos*) const [inline]

Definition at line 140 of file diffNumber.h.

10.60.3.5 void diffNumber::setDeg (int *n*) [inline]

Definition at line 166 of file diffNumber.h.

10.60.4 Member Data Documentation

10.60.4.1 int diffNumber::MAXDEG

Definition at line 22 of file diffNumber.h.

The documentation for this class was generated from the following file:

- diffNumber.h

10.61 coco::diffNumber_eval Class Reference

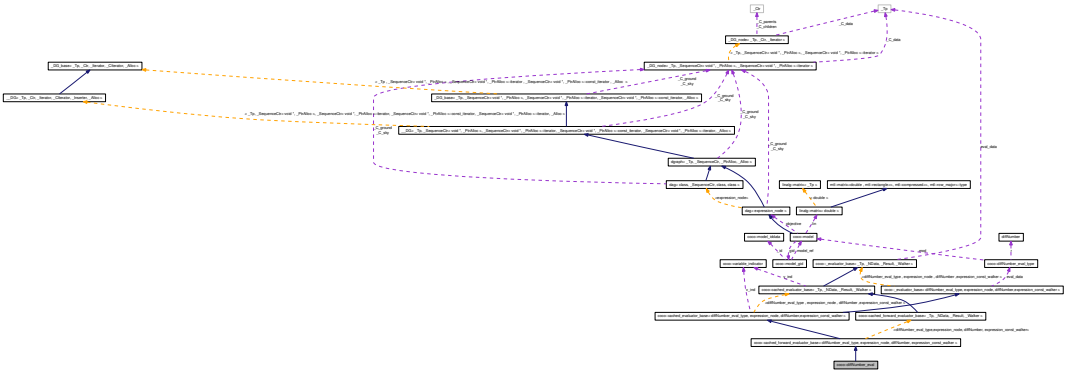
Forward function range evaluation.

`#include <diffNumber_evaluator.h>`

Inheritance diagram for coco::diffNumber_eval:



Collaboration diagram for coco::diffNumber_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker
- typedef diffNumber_eval_type data_type

Public Member Functions

- diffNumber_eval (const std::vector< diffNumber > &__x, const variable_indicator &__v, const model &__m, std::vector< diffNumber > *__c, int __maxd)
- diffNumber_eval (const diffNumber_eval &__v)
- ~diffNumber_eval ()
- expression_const_walker short_cut_to (const expression_node &__data)
- void new_point (const std::vector< diffNumber > &__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 void cleanup (const node_data_type &__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 diffNumber &__rval)
  - int update (const expression_node &__data, const diffNumber &__rval)
  - diffNumber calculate_value (bool eval_all)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- diffNumber_eval_type eval_data

10.61.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural diffNumber extension.

Definition at line 75 of file diffNumber_evaluator.h.

10.61.2 Member Typedef Documentation

10.61.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.61.2.2 typedef diffNumber_eval_type coco::_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.61.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.61.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.61.3 Constructor & Destructor Documentation

10.61.3.1 coco::diffNumber_eval::diffNumber_eval (const std::vector< diffNumber > & *__x*, const variable_indicator & *__v*, const model & *__m*, std::vector< diffNumber > * *__c*, int *__maxd*) [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 diffNumber extension will use known node ranges for reducing the ranges.

Definition at line 146 of file diffNumber_evaluator.h.

10.61.3.2 coco::diffNumber_eval::diffNumber_eval (const diffNumber_eval & *__v*) [inline]

Standard Copy Constructor

Definition at line 163 of file diffNumber_evaluator.h.

10.61.3.3 coco::diffNumber_eval::~diffNumber_eval () [inline]

Standard Destructor

Definition at line 166 of file diffNumber_evaluator.h.

10.61.4 Member Function Documentation

10.61.4.1 void coco::diffNumber_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >.

Definition at line 255 of file diffNumber_evaluator.h.

10.61.4.2 diffNumber coco::diffNumber_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >.

Definition at line 728 of file diffNumber_evaluator.h.

10.61.4.3 virtual void coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::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 804 of file evaluator.h.

10.61.4.4 int coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.61.4.5 int coco::diffNumber_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >.

Definition at line 189 of file diffNumber_evaluator.h.

10.61.4.6 void coco::diffNumber_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >.

Definition at line 187 of file diffNumber_evaluator.h.

10.61.4.7 bool coco::diffNumber_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::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >.

Definition at line 88 of file diffNumber_evaluator.h.

10.61.4.8 void coco::diffNumber_eval::new_point (const std::vector< diffNumber > & *__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 179 of file diffNumber_evaluator.h.

10.61.4.9 void coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.61.4.10 int coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.61.4.11 void coco::diffNumber_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >.

Definition at line 264 of file diffNumber_evaluator.h.

10.61.4.12 expression_const_walker coco::diffNumber_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 169 of file diffNumber_evaluator.h.

10.61.4.13 int coco::diffNumber_eval::update (const expression_node & *__data*, const diffNumber & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >.

Definition at line 280 of file diffNumber_evaluator.h.

10.61.4.14 int coco::diffNumber_eval::update (const diffNumber & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< diffNumber_eval_type, expression_node, diffNumber, expression_const_walker >.

Definition at line 271 of file diffNumber_evaluator.h.

10.61.4.15 return_value coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.61.4.16 int coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.61.4.17 void coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::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 771 of file evaluator.h.

10.61.4.18 return_value coco::cached_forward_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< diffNumber_eval_type, expression_node, diffNumber,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.61.5 Member Data Documentation

10.61.5.1 diffNumber_eval_type coco::_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.61.5.2 const variable_indicator* coco::cached_evaluator_base< diffNumber_eval_type , expression_node , diffNumber , expression_const_walker >::v_ind [protected, inherited]

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:

- diffNumber_evaluator.h

10.62 coco::diffNumber_eval_type Struct Reference

Visitor data for diffNumber_eval.

`#include <diffNumber_evaluator.h>`

Collaboration diagram for coco::diffNumber_eval_type:



Public Attributes

- const std::vector< diffNumber > * x
- std::vector< diffNumber > * cache
- const model * mod
- void * p
- diffNumber r
- unsigned int n
- int maxdeg

10.62.1 Detailed Description

This class is the visitor data type for the diffNumber_eval evaluator.

Definition at line 57 of file diffNumber_evaluator.h.

10.62.2 Member Data Documentation

10.62.2.1 std::vector<diffNumber>* coco::diffNumber_eval_type::cache

function range the function range cache

Definition at line 60 of file diffNumber_evaluator.h.

10.62.2.2 int coco::diffNumber_eval_type::maxdeg

maximal degree of derivative

Definition at line 65 of file diffNumber_evaluator.h.

10.62.2.3 const model* coco::diffNumber_eval_type::mod

the DAG

Definition at line 61 of file diffNumber_evaluator.h.

10.62.2.4 unsigned int coco::diffNumber_eval_type::n

children counter

Definition at line 64 of file diffNumber_evaluator.h.

10.62.2.5 void* coco::diffNumber_eval_type::p

additional data for

Definition at line 62 of file diffNumber_evaluator.h.

10.62.2.6 diffNumber coco::diffNumber_eval_type::r

return value

Definition at line 63 of file diffNumber_evaluator.h.

10.62.2.7 const std::vector<diffNumber>* coco::diffNumber_eval_type::x

Definition at line 59 of file diffNumber_evaluator.h.

The documentation for this struct was generated from the following file:

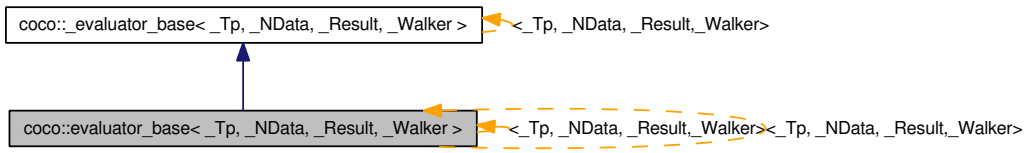
- diffNumber_evaluator.h

10.63 coco::evaluator_base< _Tp, _NData, _Result, _Walker > Class Template Reference

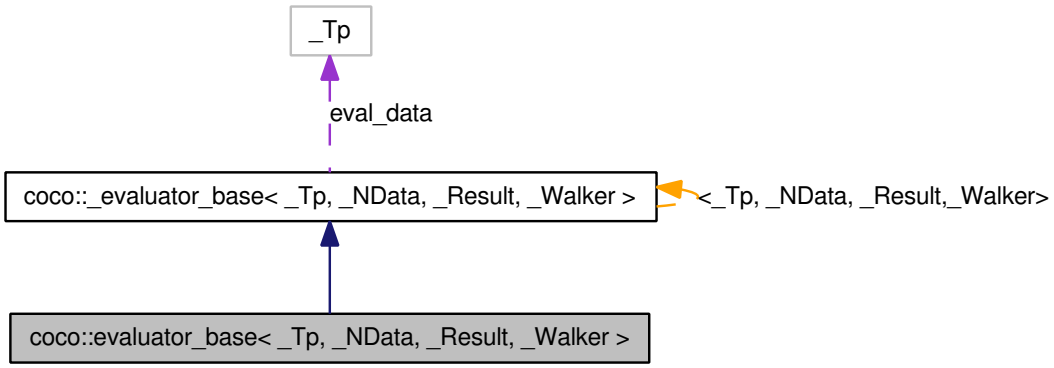
Base class of all (non-caching) evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::evaluator_base< _Tp, _NData, _Result, _Walker >:



Collaboration diagram for coco::evaluator_base< _Tp, _NData, _Result, _Walker >:



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

10.63.1 Detailed Description

template<class _Tp, class _NData, class _Result, class _Walker> class coco::evaluator_base< _Tp, _NData, _Result, _Walker >

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.

Definition at line 379 of file evaluator.h.

10.63.2 Member Typedef Documentation

10.63.2.1 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::const_walker coco::evaluator_base< _Tp, _NData, _Result, _Walker >::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >, and coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 390 of file evaluator.h.

10.63.2.2 typedef _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.63.2.3 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::node_data_type coco::evaluator_base< _Tp, _NData, _Result, _Walker >::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >, and coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 386 of file evaluator.h.

10.63.2.4 template<class _Tp, class _NData, class _Result, class _Walker> typedef _Base::return_value coco::evaluator_base< _Tp, _NData, _Result, _Walker >::return_value

This type is the result type of the evaluator.

Reimplemented from coco::_evaluator_base< _Tp, _NData, _Result, _Walker >.

Reimplemented in coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >, and coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 388 of file evaluator.h.

10.63.3 Member Function Documentation

10.63.3.1 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 359 of file evaluator.h.

10.63.3.2 virtual void coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 364 of file evaluator.h.

10.63.3.3 template<class _Tp, class _NData, class _Result, class _Walker> virtual int coco::evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.63.3.4 template<class _Tp, class _NData, class _Result, class _Walker> virtual const_walker coco::evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.63.3.5 virtual return_value coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 336 of file evaluator.h.

10.63.3.6 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 348 of file evaluator.h.

10.63.3.7 virtual return_value coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 332 of file evaluator.h.

10.63.4 Member Data Documentation

10.63.4.1 _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::eval_data [protected, inherited]

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:

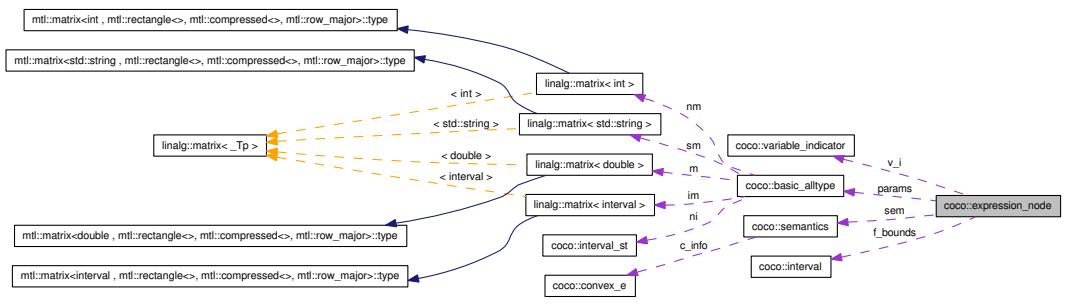
- evaluator.h

10.64 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)

10.64.1 Detailed Description

This class is the base class for all nodes in the expression DAGs used to specify the optimization problems to be solved.

Definition at line 1112 of file expression.h.

10.64.2 Constructor & Destructor Documentation

10.64.2.1 coco::expression_node::expression_node () [inline]

The Standard Constructor

Definition at line 1159 of file expression.h.

10.64.2.2 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 1166 of file expression.h.

10.64.2.3 coco::expression_node::expression_node (const expression_node & *__x*) [inline]

Standard Copy Constructor

Definition at line 1173 of file expression.h.

10.64.2.4 virtual coco::expression_node::~expression_node () [inline, virtual]

Standard Destructor

Definition at line 1181 of file expression.h.

10.64.3 Member Function Documentation

10.64.3.1 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 1260 of file expression.h.

10.64.3.2 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 1303 of file expression.h.

10.64.3.3 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 1295 of file expression.h.

10.64.3.4 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 1299 of file expression.h.

10.64.3.5 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 38 of file expression.cc.

10.64.3.6 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 1220 of file expression.h.

10.64.3.7 bool 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 214 of file expression.hpp.

10.64.3.8 expression_node& coco::expression_node::operator= (const expression_node & *__x*) [inline]

Standard Assignment Operator

Definition at line 1189 of file expression.h.

10.64.3.9 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 1268 of file expression.h.

10.64.3.10 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 1253 of file expression.h.

10.64.3.11 void coco::expression_node::set_bounds (int __i) [inline]

This method sets the node's bounds to the restriction "in __i".

Definition at line 1246 of file expression.h.

10.64.3.12 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 1240 of file expression.h.

10.64.3.13 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 1233 of file expression.h.

10.64.3.14 const variable_indicator& coco::expression_node::var_indicator () const [inline]

This method returns the variable indicator of this node.

Definition at line 1290 of file expression.h.

10.64.4 Friends And Related Function Documentation

10.64.4.1 std::ostream& operator<< (std::ostream & o, const expression_node & __x) [friend]

Definition at line 207 of file expression.cc.

10.64.5 Member Data Documentation

10.64.5.1 std::vector<double> coco::expression_node::coeffs

Definition at line 1130 of file expression.h.

10.64.5.2 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 1145 of file expression.h.

10.64.5.3 rhs_t coco::expression_node::f_bounds

Definition at line 1134 of file expression.h.

10.64.5.4 unsigned short coco::expression_node::is_var

Definition at line 1136 of file expression.h.

10.64.5.5 unsigned int coco::expression_node::n_children

Definition at line 1126 of file expression.h.

10.64.5.6 unsigned int coco::expression_node::n_parents

Definition at line 1126 of file expression.h.

10.64.5.7 unsigned int coco::expression_node::node_num

Definition at line 1115 of file expression.h.

10.64.5.8 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 1117 of file expression.h.

10.64.5.9 basic_alltype coco::expression_node::params

Definition at line 1132 of file expression.h.

10.64.5.10 semantics coco::expression_node::sem

additional semantics information like integer (y/n), stochastic (y/n), karush-john (y/n) convexity,...

Definition at line 1139 of file expression.h.

10.64.5.11 variable_indicator coco::expression_node::v_i

Definition at line 1143 of file expression.h.

10.64.5.12 std::vector<unsigned int> coco::expression_node::var_idx

Definition at line 1137 of file expression.h.

The documentation for this class was generated from the following files:

- expression.h
- expression.cc
- expression.hpp

10.65 coco::expression_node::expression_node::children_compare Class Reference

`#include <expression.hpp>`

Public Member Functions

- bool operator() (const expression_node &__x, const expression_node &__y) const

10.65.1 Detailed Description

Definition at line 78 of file expression.hpp.

10.65.2 Member Function Documentation

10.65.2.1 bool coco::expression_node::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 83 of file expression.hpp.

The documentation for this class was generated from the following file:

- expression.hpp

10.66 coco::expression_node::expression_node::parents_compare Class Reference

`#include <expression.hpp>`

Public Member Functions

- bool operator() (const expression_node &__x, const expression_node &__y) const

10.66.1 Detailed Description

Definition at line 35 of file expression.hpp.

10.66.2 Member Function Documentation

10.66.2.1 bool coco::expression_node::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 40 of file expression.hpp.

The documentation for this class was generated from the following file:

- expression.hpp

10.67 coco::expression_node::expression_node::parents_compare_eq Class Reference

`#include <expression.hpp>`

Public Member Functions

- bool operator() (const expression_node &__x, const expression_node &__y) const

10.67.1 Detailed Description

Definition at line 125 of file expression.hpp.

10.67.2 Member Function Documentation

10.67.2.1 bool coco::expression_node::expression_node::parents_compare_eq::operator() (const expression_node & *__x*, const expression_node & *__y*) const [inline]

This is the evaluation operator of this function class.

Definition at line 130 of file expression.hpp.

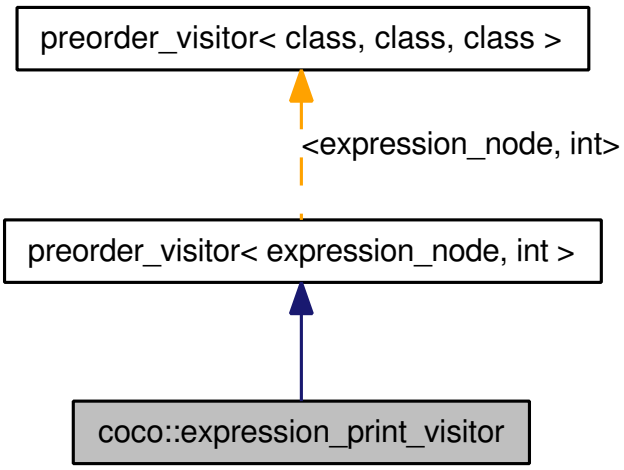
The documentation for this class was generated from the following file:

- expression.hpp

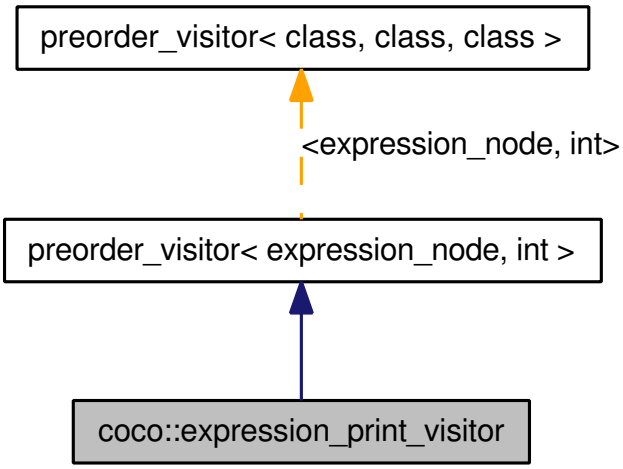
10.68 coco::expression_print_visitor Class Reference

`#include <expression.hpp>`

Inheritance diagram for coco::expression_print_visitor:



Collaboration diagram for coco::expression_print_visitor:



Public Types

- typedef _Ret **return_value**

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 ()
- virtual void **vinit** ()
- virtual bool **preorder** (const _Node &__n)
- virtual void **collect** (const _Node &__n, collect_value __r)

- bool preorder (const expression_node &r)
- int vvalue ()
- int value ()
- void collect (const expression_node &r, int __r)

10.68.1 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.

Definition at line 224 of file expression.hpp.

10.68.2 Member Typedef Documentation

10.68.2.1 typedef _Ret preorder_visitor< class, class, class >::return_value [inherited]

10.68.3 Constructor & Destructor Documentation

10.68.3.1 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 237 of file expression.hpp.

10.68.3.2 coco::expression_print_visitor::expression_print_visitor (const expression_print_visitor & __p) [inline]

Standard Copy Constructor

Definition at line 241 of file expression.hpp.

10.68.3.3 coco::expression_print_visitor::~expression_print_visitor () [inline]

Standard Destructor

Definition at line 244 of file expression.hpp.

10.68.4 Member Function Documentation

10.68.4.1 virtual void preorder_visitor< class, class, class >::collect (const _Node & __n, collect_value __r) [virtual, inherited]

10.68.4.2 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 265 of file expression.hpp.

10.68.4.3 virtual bool preorder_visitor< class, class, class >::preorder (const _Node & __n) [virtual, inherited]

10.68.4.4 bool coco::expression_print_visitor::preorder (const expression_node & *r*) [inline]

This function is needed by a VGTL preorder visitor.

Definition at line 248 of file expression.hpp.

10.68.4.5 int coco::expression_print_visitor::value () [inline]

This function is needed by a VGTL preorder visitor.

Definition at line 264 of file expression.hpp.

10.68.4.6 virtual void preorder_visitor< class, class, class >::vinit () [virtual, inherited]

10.68.4.7 int coco::expression_print_visitor::vvalue () [inline, virtual]

This function is needed by a VGTL preorder visitor.

Reimplemented from **preorder_visitor< expression_node, int >**.

Definition at line 263 of file expression.hpp.

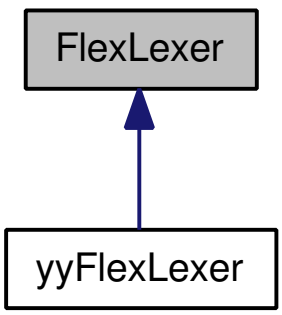
The documentation for this class was generated from the following file:

- expression.hpp

10.69 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

10.69.1 Detailed Description

Definition at line 61 of file FlexLexer.h.

10.69.2 Constructor & Destructor Documentation

10.69.2.1 virtual FlexLexer::~FlexLexer () [inline, virtual]

Definition at line 63 of file FlexLexer.h.

10.69.3 Member Function Documentation

10.69.3.1 int FlexLexer::debug () const [inline]

Definition at line 91 of file FlexLexer.h.

10.69.3.2 int FlexLexer::lineno () const [inline]

Definition at line 89 of file FlexLexer.h.

10.69.3.3 void FlexLexer::set_debug (int *flag*) [inline]

Definition at line 92 of file FlexLexer.h.

10.69.3.4 virtual void FlexLexer::switch_streams (FLEX_STD istream * *new_in* = 0, FLEX_STD ostream * *new_out* = 0) [pure virtual]

Implemented in yyFlexLexer.

10.69.3.5 virtual struct yy_buffer_state* FlexLexer::yy_create_buffer (FLEX_STD istream * *s*, int *size*) [read, pure virtual]

Implemented in yyFlexLexer.

10.69.3.6 virtual void FlexLexer::yy_delete_buffer (struct yy_buffer_state * *b*) [pure virtual]

Implemented in yyFlexLexer.

10.69.3.7 virtual void FlexLexer::yy_switch_to_buffer (struct yy_buffer_state * *new_buffer*) [pure virtual]

Implemented in yyFlexLexer.

10.69.3.8 int FlexLexer::YYLeng () [inline]

Definition at line 66 of file FlexLexer.h.

10.69.3.9 int FlexLexer::yylex (FLEX_STD istream * *new_in*, FLEX_STD ostream * *new_out* = 0) [inline]

Definition at line 78 of file FlexLexer.h.

10.69.3.10 virtual int FlexLexer::yylex () [pure virtual]

Implemented in yyFlexLexer.

10.69.3.11 virtual void FlexLexer::yyrestart (FLEX_STD istream * *s*) [pure virtual]

Implemented in yyFlexLexer.

10.69.3.12 const char* FlexLexer::YYText () [inline]

Definition at line 65 of file FlexLexer.h.

10.69.4 Member Data Documentation

10.69.4.1 int FlexLexer::yy_flex_debug [protected]

Definition at line 98 of file FlexLexer.h.

10.69.4.2 int FlexLexer::yyleng [protected]

Definition at line 96 of file FlexLexer.h.

10.69.4.3 int FlexLexer::yylineno [protected]

Definition at line 97 of file FlexLexer.h.

10.69.4.4 char* FlexLexer::yytext [protected]

Definition at line 95 of file FlexLexer.h.

The documentation for this class was generated from the following file:

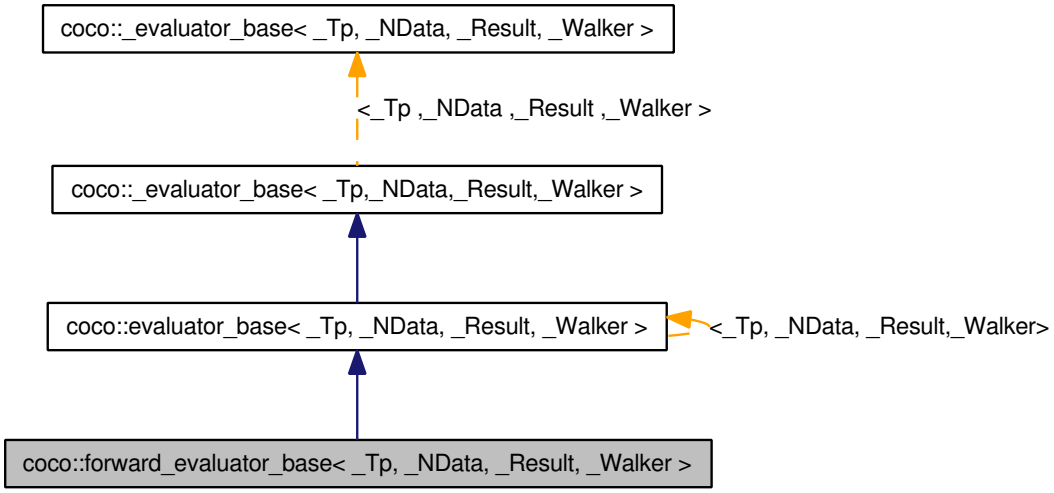
- FlexLexer.h

10.70 coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker > Class Template Reference

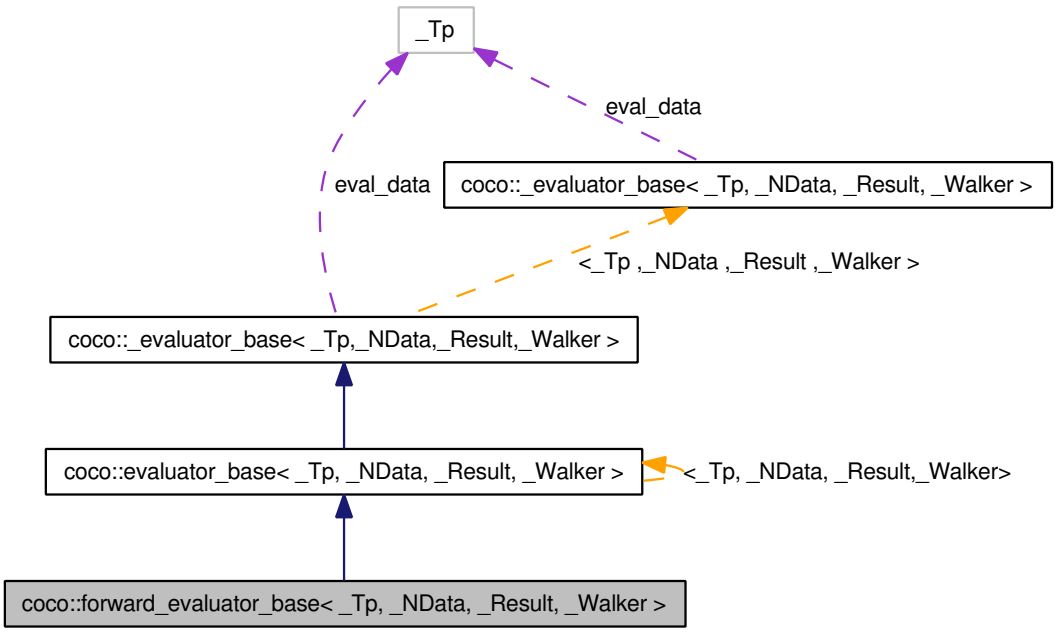
Base class of all (non-caching) forward evaluators.

`#include <evaluator.h>`

Inheritance diagram for coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >:



Collaboration diagram for coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >:



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

- 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)
- 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

10.70.1 Detailed Description

template<class _Tp, class _NData, class _Result, class _Walker> class coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >

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.

Definition at line 475 of file evaluator.h.

10.70.2 Member Typedef Documentation

10.70.2.1 template<class _Tp , class _NData , class _Result , class _Walker > typedef _Base::const_walker coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::const_walker

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 489 of file evaluator.h.

10.70.2.2 typedef _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.70.2.3 template<class _Tp , class _NData , class _Result , class _Walker > typedef _Base::node_data_type coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::node_data_type

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 485 of file evaluator.h.

10.70.2.4 template<class _Tp , class _NData , class _Result , class _Walker > typedef _Base::return_value coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::return_value

This type is the result type of the evaluator.

Reimplemented from coco::evaluator_base< _Tp, _NData, _Result, _Walker >.

Definition at line 487 of file evaluator.h.

10.70.3 Member Function Documentation

10.70.3.1 template<class _Tp , class _NData , class _Result , class _Walker > virtual void coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.2 template<class _Tp , class _NData , class _Result , class _Walker > virtual return_value coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.3 template<class _Tp , class _NData , class _Result , class _Walker > virtual void coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.4 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 359 of file evaluator.h.

10.70.3.5 template<class _Tp , class _NData , class _Result , class _Walker > int coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.6 template<class _Tp , class _NData , class _Result , class _Walker > virtual int coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.7 template<class _Tp , class _NData , class _Result , class _Walker > virtual void coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Definition at line 532 of file evaluator.h.

10.70.3.8 virtual void coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 364 of file evaluator.h.

10.70.3.9 template<class _Tp , class _NData , class _Result , class _Walker > void coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.10 virtual int coco::evaluator_base< _Tp , _NData , _Result , _Walker >::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 401 of file evaluator.h.

10.70.3.11 template<class _Tp , class _NData , class _Result , class _Walker > int coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.12 virtual const_walker coco::evaluator_base< _Tp , _NData , _Result , _Walker >::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 405 of file evaluator.h.

10.70.3.13 template<class _Tp , class _NData , class _Result , class _Walker > virtual int coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.14 template<class _Tp , class _NData , class _Result , class _Walker > virtual int coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.15 template<class _Tp , class _NData , class _Result , class _Walker > return_value coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< _Tp,_NData,_Result,_Walker >.

Definition at line 518 of file evaluator.h.

10.70.3.16 virtual int coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::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 348 of file evaluator.h.

10.70.3.17 template<class _Tp , class _NData , class _Result , class _Walker > int coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.18 template<class _Tp , class _NData , class _Result , class _Walker > void coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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.

10.70.3.19 template<class _Tp , class _NData , class _Result , class _Walker > return_value coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< _Tp,_NData,_Result,_Walker >.

Definition at line 523 of file evaluator.h.

10.70.4 Member Data Documentation

10.70.4.1 _Tp coco::_evaluator_base< _Tp , _NData , _Result , _Walker >::eval_data [protected, inherited]

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:

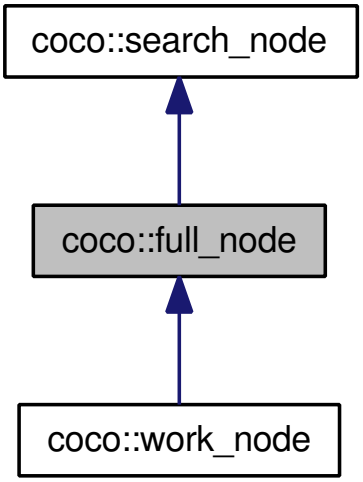
- evaluator.h

10.71 coco::full_node Class Reference

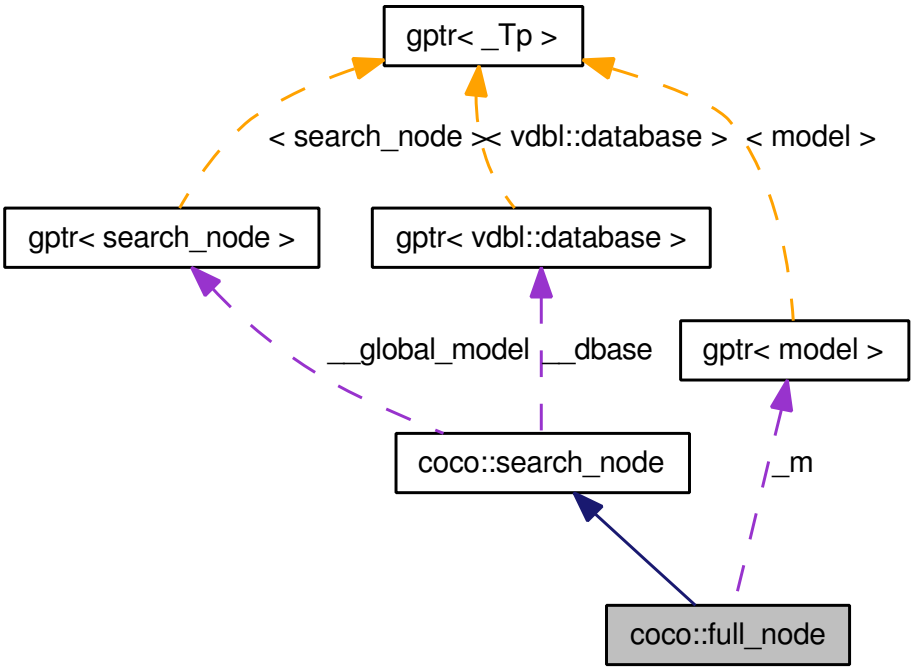
Class holding the full nodes in the search graph.

`#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

10.71.1 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.

Definition at line 278 of file search_node.h.

10.71.2 Constructor & Destructor Documentation

10.71.2.1 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 299 of file search_node.h.

10.71.2.2 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 312 of file search_node.h.

10.71.2.3 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 327 of file search_node.h.

10.71.2.4 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 340 of file search_node.h.

10.71.2.5 coco::full_node::full_node (const full_node & *_i*) [inline]

Standard copy constructor

Definition at line 348 of file search_node.h.

10.71.2.6 virtual coco::full_node::~full_node () [inline, virtual]

Standard Destructor

Definition at line 353 of file search_node.h.

10.71.3 Member Function Documentation

10.71.3.1 gptr<vdbl::database>* coco::search_node::database () const [inline, inherited]

This is the accessor method for the search database.

Definition at line 164 of file search_node.h.

10.71.3.2 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 369 of file search_node.h.

10.71.3.3 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 374 of file search_node.h.

10.71.3.4 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 385 of file search_node.h.

10.71.3.5 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 396 of file search_node.h.

10.71.3.6 vdbl::userid coco::search_node::get_dbuserid () const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 158 of file search_node.h.

10.71.3.7 search_node_id coco::search_node::get_id () const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 167 of file search_node.h.

10.71.3.8 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.

Reimplemented in coco::work_node.

Definition at line 379 of file search_node.h.

10.71.3.9 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 390 of file search_node.h.

10.71.3.10 vdbl::rowid coco::search_node::get_rowid () const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 173 of file search_node.h.

10.71.3.11 gptr<search_node>* coco::search_node::global_model () const [inline, inherited]

This is the accessor method for the global model.

Definition at line 161 of file search_node.h.

10.71.3.12 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::search_node.

Definition at line 361 of file search_node.h.

10.71.3.13 void 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 184 of file search_node.h.

10.71.3.14 void 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 180 of file search_node.h.

10.71.3.15 unsigned int coco::full_node::n_annotations () const [inline]

This method returns the number of annotations stored in this full node.

Definition at line 365 of file search_node.h.

10.71.3.16 full_node & coco::full_node::operator= (const full_node & *__w*) [inline]

Standard Assignment Operator

Reimplemented in coco::work_node.

Definition at line 207 of file search_node.hpp.

10.71.3.17 search_node_relation 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 117 of file search_node.h.

10.71.3.18 void 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 170 of file search_node.h.

10.71.3.19 void 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 176 of file search_node.h.

10.71.3.20 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.

10.71.3.21 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.

10.71.4 Friends And Related Function Documentation

10.71.4.1 friend class certificate_base [friend]

Definition at line 400 of file search_node.h.

10.71.4.2 friend class dag_delta [friend]

Definition at line 401 of file search_node.h.

10.71.4.3 friend class dag_undelta [friend]

Definition at line 402 of file search_node.h.

10.71.4.4 friend class delta_base [friend]

Definition at line 399 of file search_node.h.

10.71.4.5 friend class search_graph [friend, inherited]

Definition at line 195 of file search_node.h.

10.71.5 Member Data Documentation

10.71.5.1 gptr<vdbl::database>* 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 96 of file search_node.h.

10.71.5.2 gptr<search_node>* 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 93 of file search_node.h.

10.71.5.3 std::vector<annotation> coco::full_node::_ann

This is the vector of annotations active at this full node.

Definition at line 290 of file search_node.h.

10.71.5.4 vdbl::userid 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 99 of file search_node.h.

10.71.5.5 search_node_id coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 104 of file search_node.h.

10.71.5.6 std::vector<annotation> 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 108 of file search_node.h.

10.71.5.7 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 286 of file search_node.h.

10.71.5.8 vdbl::rowid 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 112 of file search_node.h.

10.71.5.9 search_node_relation 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 102 of file search_node.h.

The documentation for this class was generated from the following files:

- search_node.h
- search_node.hpp

10.72 coco::func_cache Struct Reference

Cache data for func_eval.

`#include <func_evaluator_n.h>`

Public Attributes

- double f
- bool v

10.72.1 Detailed Description

This class is the cache data type for the func_eval evaluator.

Definition at line 53 of file func_evaluator_n.h.

10.72.2 Member Data Documentation

10.72.2.1 double coco::func_cache::f

the node function value

Definition at line 54 of file func_evaluator_n.h.

10.72.2.2 bool coco::func_cache::v

has this node been visited

Definition at line 55 of file func_evaluator_n.h.

The documentation for this struct was generated from the following file:

- func_evaluator_n.h

10.73 coco::func_d_eval Class Reference

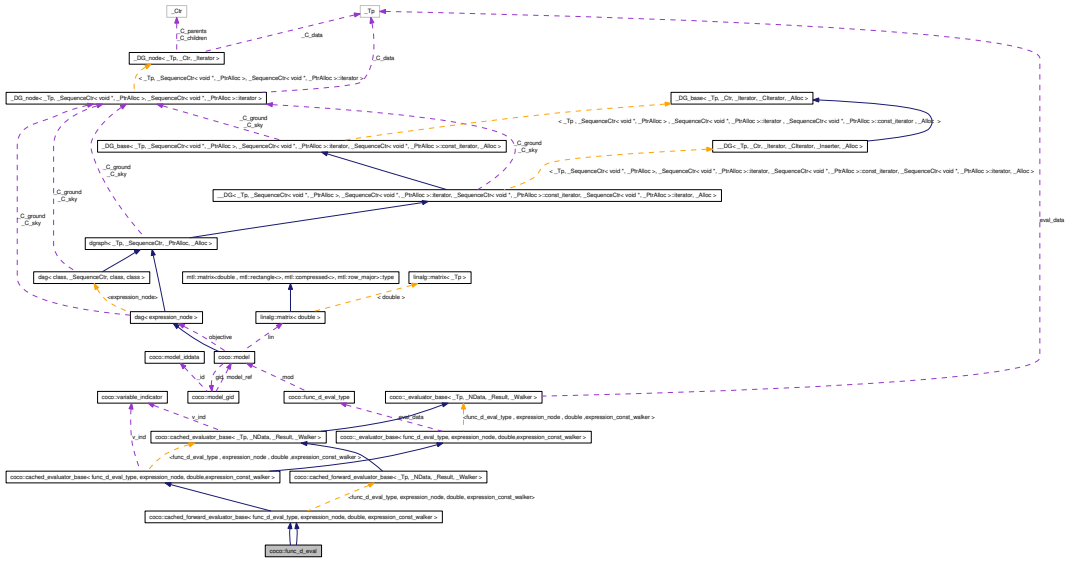
Forward function evaluation with preparation of derivative data.

`#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
- typedef func_d_eval_type data_type

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)
- 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 void cleanup (const node_data_type &__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 double &__rval)
- int update (const expression_node &__data, const double &__rval)
- double calculate_value (bool eval_all)

- 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)

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- func_d_eval_type eval_data

10.73.1 Detailed Description

This class is a cached forward evaluator performing a function evaluation and the preparation of partial derivative information for backwards differentiation.

Definition at line 148 of file der_evaluator.h.

10.73.2 Member Typedef Documentation

10.73.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.73.2.2 typedef func_d_eval_type coco::_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.73.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.73.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.73.3 Constructor & Destructor Documentation

10.73.3.1 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 241 of file der_evaluator.h.

10.73.3.2 coco::func_d_eval::func_d_eval (const func_d_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 256 of file der_evaluator.h.

10.73.3.3 coco::func_d_eval::~func_d_eval () [inline]

Standard Destructor

Definition at line 259 of file der_evaluator.h.

10.73.3.4 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 238 of file der_evaluator_n.h.

10.73.3.5 coco::func_d_eval::func_d_eval (const func_d_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 253 of file der_evaluator_n.h.

10.73.3.6 coco::func_d_eval::~func_d_eval () [inline]

Standard Destructor

Definition at line 256 of file der_evaluator_n.h.

10.73.4 Member Function Documentation

10.73.4.1 void coco::func_d_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 335 of file der_evaluator_n.h.

10.73.4.2 void coco::func_d_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 338 of file der_evaluator.h.

10.73.4.3 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::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 970 of file der_evaluator_n.h.

10.73.4.4 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::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 973 of file der_evaluator.h.

10.73.4.5 virtual void coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::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 804 of file evaluator.h.

10.73.4.6 int coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.73.4.7 int coco::func_d_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 275 of file der_evaluator_n.h.

10.73.4.8 void coco::func_d_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 273 of file der_evaluator_n.h.

10.73.4.9 int coco::func_d_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 278 of file der_evaluator.h.

10.73.4.10 void coco::func_d_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 276 of file der_evaluator.h.

10.73.4.11 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::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 157 of file der_evaluator_n.h.

10.73.4.12 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::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 160 of file der_evaluator.h.

10.73.4.13 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 265 of file der_evaluator_n.h.

10.73.4.14 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 268 of file der_evaluator.h.

10.73.4.15 void coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< func_d_eval_type, expression_node, double,expression_-const_walker >.

Definition at line 744 of file evaluator.h.

10.73.4.16 int coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.73.4.17 void coco::func_d_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 345 of file der_evaluator_n.h.

10.73.4.18 void coco::func_d_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 348 of file der_evaluator.h.

10.73.4.19 expression_const_walker coco::func_d_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 259 of file der_evaluator_n.h.

10.73.4.20 expression_const_walker coco::func_d_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 262 of file der_evaluator.h.

10.73.4.21 int coco::func_d_eval::update (const expression_node & *__data*, const double & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 378 of file der_evaluator_n.h.

10.73.4.22 int coco::func_d_eval::update (const double & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 372 of file der_evaluator_n.h.

10.73.4.23 int coco::func_d_eval::update (const expression_node & *__data*, const double & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 381 of file der_evaluator.h.

10.73.4.24 int coco::func_d_eval::update (const double & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_d_eval_type, expression_node, double, expression_const_walker >.

Definition at line 375 of file der_evaluator.h.

10.73.4.25 return_value coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.73.4.26 int coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< func_d_eval_type, expression_node, double,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.73.4.27 void coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::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 771 of file evaluator.h.

10.73.4.28 return_value coco::cached_forward_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< func_d_eval_type, expression_node, double,expression_-const_walker >.

Definition at line 767 of file evaluator.h.

10.73.5 Member Data Documentation

10.73.5.1 func_d_eval_type coco::_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.73.5.2 const variable_indicator* coco::cached_evaluator_base< func_d_eval_type , expression_node , double , expression_const_walker >::v_ind [protected, inherited]

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 files:

- der_evaluator.h
- der_evaluator_n.h

10.74 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
- union {
  void * p
  double d
  } u

10.74.1 Detailed Description

This class is the visitor data type for the func_d_eval evaluator.

Definition at line 129 of file der_evaluator.h.

10.74.2 Member Data Documentation

10.74.2.1 double coco::func_d_eval_type::d

Definition at line 135 of file der_evaluator.h.

10.74.2.2 std::vector< std::vector< double > > * coco::func_d_eval_type::d_data

the derivative data

Definition at line 133 of file der_evaluator.h.

10.74.2.3 std::vector< double > * coco::func_d_eval_type::f_cache

the function value cache

Definition at line 132 of file der_evaluator.h.

10.74.2.4 unsigned int coco::func_d_eval_type::info

needed for derivative of max, min

Definition at line 137 of file der_evaluator.h.

10.74.2.5 const model * coco::func_d_eval_type::mod

the DAG

Definition at line 134 of file der_evaluator.h.

10.74.2.6 unsigned int coco::func_d_eval_type::n

children counter

Definition at line 137 of file der_evaluator.h.

10.74.2.7 void* coco::func_d_eval_type::p

Definition at line 135 of file der_evaluator.h.

10.74.2.8 double coco::func_d_eval_type::r

return value

Definition at line 136 of file der_evaluator.h.

10.74.2.9 union { ... } coco::func_d_eval_type::u

additional data for complex nodes

10.74.2.10 union { ... } coco::func_d_eval_type::u

additional data for complex nodes

10.74.2.11 const std::vector< double > * coco::func_d_eval_type::x

the point

Definition at line 131 of file der_evaluator.h.

The documentation for this struct was generated from the following files:

- der_evaluator.h
- der_evaluator_n.h

10.75 coco::func_eval Class Reference

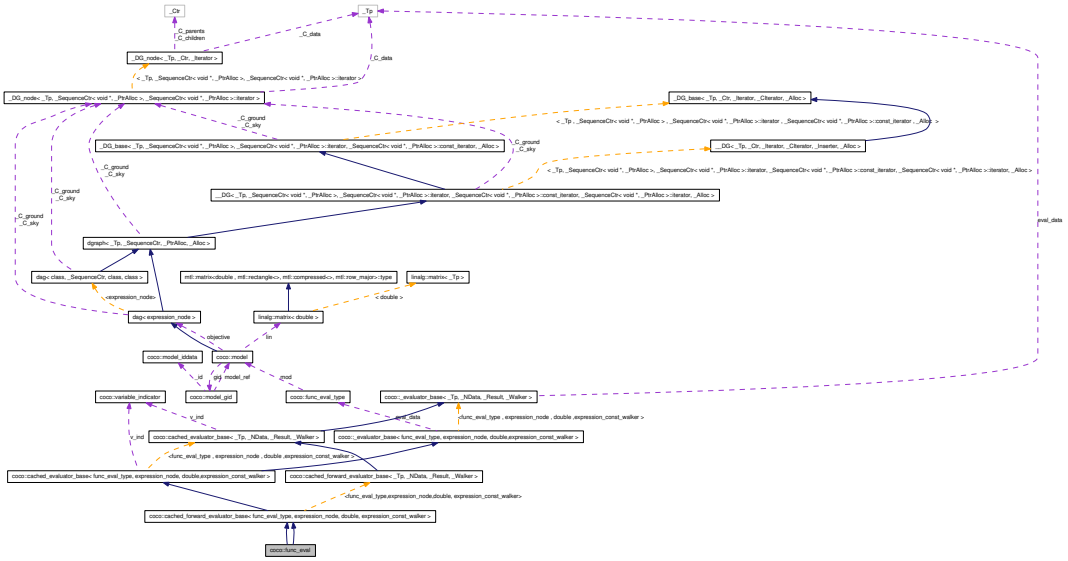
Forward function evaluation.

`#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
- typedef func_eval_type data_type

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)
- func_eval (const std::vector< double > &__x, const variable_indicator &__v, const model &__m, std::vector< func_cache > *__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 void cleanup (const node_data_type &__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 double &__rval)
- int update (const expression_node &__data, const double &__rval)
- double calculate_value (bool eval_all)

- 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)
- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- func_eval_type eval_data

10.75.1 Detailed Description

This class is a cached forward evaluator performing a function evaluation.

Definition at line 69 of file func_evaluator.h.

10.75.2 Member Typedef Documentation

10.75.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.75.2.2 typedef func_eval_type coco::_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.75.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.75.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.75.3 Constructor & Destructor Documentation

10.75.3.1 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 140 of file func_evaluator.h.

10.75.3.2 coco::func_eval::func_eval (const func_eval & __v) [inline]

Standard Copy Constructor

Definition at line 152 of file func_evaluator.h.

10.75.3.3 coco::func_eval::~func_eval () [inline]

Standard Destructor

Definition at line 155 of file func_evaluator.h.

10.75.3.4 coco::func_eval::func_eval (const std::vector< double > & __x, const variable_indicator & __v, const model & __m, std::vector< func_cache > * __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 150 of file func_evaluator_n.h.

10.75.3.5 coco::func_eval::func_eval (const func_eval & __v) [inline]

Standard Copy Constructor

Definition at line 162 of file func_evaluator_n.h.

10.75.3.6 coco::func_eval::~func_eval () [inline]

Standard Destructor

Definition at line 165 of file func_evaluator_n.h.

10.75.4 Member Function Documentation

10.75.4.1 void coco::func_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 260 of file func_evaluator_n.h.

10.75.4.2 void coco::func_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 247 of file func_evaluator.h.

10.75.4.3 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::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 715 of file func_evaluator_n.h.

10.75.4.4 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::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 699 of file func_evaluator.h.

10.75.4.5 virtual void coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::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 804 of file evaluator.h.

10.75.4.6 int coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< func_eval_type, expression_node, double,expression_-const_walker >.

Definition at line 750 of file evaluator.h.

10.75.4.7 int coco::func_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 187 of file func_evaluator_n.h.

10.75.4.8 void coco::func_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 185 of file func_evaluator_n.h.

10.75.4.9 int coco::func_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 174 of file func_evaluator.h.

10.75.4.10 void coco::func_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 172 of file func_evaluator.h.

10.75.4.11 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::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 91 of file func_evaluator_n.h.

10.75.4.12 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::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 81 of file func_evaluator.h.

10.75.4.13 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 174 of file func_evaluator_n.h.

10.75.4.14 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 164 of file func_evaluator.h.

10.75.4.15 void coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.75.4.16 int coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.75.4.17 void coco::func_eval::retrieve_from_cache (const expression_node & __data) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 269 of file func_evaluator_n.h.

10.75.4.18 void coco::func_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 256 of file func_evaluator.h.

10.75.4.19 expression_const_walker coco::func_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 168 of file func_evaluator_n.h.

10.75.4.20 expression_const_walker coco::func_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 158 of file func_evaluator.h.

10.75.4.21 int coco::func_eval::update (const expression_node & *__data*, const double & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 293 of file func_evaluator_n.h.

10.75.4.22 int coco::func_eval::update (const double & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 287 of file func_evaluator_n.h.

10.75.4.23 int coco::func_eval::update (const expression_node & *__data*, const double & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 280 of file func_evaluator.h.

10.75.4.24 int coco::func_eval::update (const double & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_eval_type, expression_node, double, expression_const_walker >.

Definition at line 274 of file func_evaluator.h.

10.75.4.25 return_value coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::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. It is translated to a call to calculate_value with parameter false.

Reimplemented from coco::_evaluator_base< func_eval_type, expression_node, double,expression_-const_walker >.

Definition at line 762 of file evaluator.h.

10.75.4.26 int coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.75.4.27 void coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::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 771 of file evaluator.h.

10.75.4.28 return_value coco::cached_forward_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< func_eval_type, expression_node, double,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.75.5 Member Data Documentation

10.75.5.1 func_eval_type coco::_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.75.5.2 const variable_indicator* coco::cached_evaluator_base< func_eval_type , expression_node , double , expression_const_walker >::v_ind [protected, inherited]

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 files:

- func_evaluator.h
- func_evaluator_n.h

10.76 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
- std::vector< func_cache > * cache
- union {
  void * p
  double d
  } u

10.76.1 Detailed Description

This class is the visitor data type for the func_eval evaluator.

Definition at line 53 of file func_evaluator.h.

10.76.2 Member Data Documentation

10.76.2.1 std::vector<func_cache>* coco::func_eval_type::cache

the function value cache

Definition at line 66 of file func_evaluator_n.h.

10.76.2.2 std::vector<double>* coco::func_eval_type::cache

the function value cache

Definition at line 56 of file func_evaluator.h.

10.76.2.3 double coco::func_eval_type::d

Definition at line 58 of file func_evaluator.h.

10.76.2.4 const model * coco::func_eval_type::mod

the DAG

Definition at line 57 of file func_evaluator.h.

10.76.2.5 unsigned int coco::func_eval_type::n

children counter

Definition at line 60 of file func_evaluator.h.

10.76.2.6 void* coco::func_eval_type::p

Definition at line 58 of file func_evaluator.h.

10.76.2.7 double coco::func_eval_type::r

return value

Definition at line 59 of file func_evaluator.h.

10.76.2.8 union { ... } coco::func_eval_type::u

additional data for complex nodes

10.76.2.9 union { ... } coco::func_eval_type::u

additional data for complex nodes

10.76.2.10 const std::vector< double > * coco::func_eval_type::x

the point

Definition at line 55 of file func_evaluator.h.

The documentation for this struct was generated from the following files:

- func_evaluator.h
- func_evaluator_n.h

10.77 coco::func_id_eval Class Reference

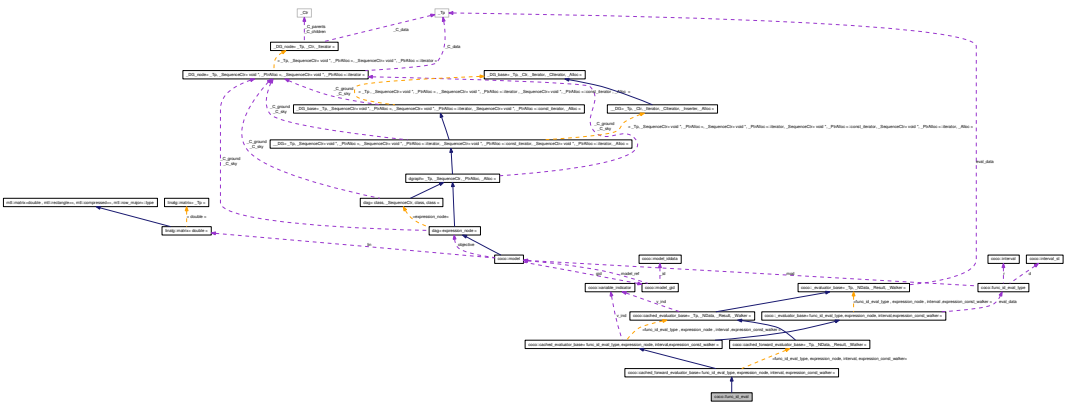
Forward function range evaluation with preparation of interval derivative data.

`#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
- typedef func_id_eval_type data_type

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)
- 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)
- 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 cleanup (const node_data_type &__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 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)

Protected Attributes

- const variable_indicator * v_ind
- func_id_eval_type eval_data

10.77.1 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.

Definition at line 151 of file ider_evaluator.h.

10.77.2 Member Typedef Documentation

10.77.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.77.2.2 typedef func_id_eval_type coco::_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.77.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.77.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.77.3 Constructor & Destructor Documentation

10.77.3.1 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*) [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 243 of file ider_evaluator.h.

10.77.3.2 coco::func_id_eval::func_id_eval (const func_id_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 262 of file ider_evaluator.h.

10.77.3.3 coco::func_id_eval::~func_id_eval () [inline]

Standard Destructor

Definition at line 265 of file ider_evaluator.h.

10.77.4 Member Function Documentation

10.77.4.1 void coco::func_id_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 358 of file ider_evaluator.h.

10.77.4.2 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::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 1263 of file ider_evaluator.h.

10.77.4.3 virtual void coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::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 804 of file evaluator.h.

10.77.4.4 int coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.77.4.5 int coco::func_id_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 293 of file ider_evaluator.h.

10.77.4.6 void coco::func_id_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 291 of file ider_evaluator.h.

10.77.4.7 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::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 163 of file ider_evaluator.h.

10.77.4.8 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 274 of file ider_evaluator.h.

10.77.4.9 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 283 of file ider_evaluator.h.

10.77.4.10 void coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.77.4.11 int coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.77.4.12 void coco::func_id_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 371 of file ider_evaluator.h.

10.77.4.13 expression_const_walker coco::func_id_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< func_id_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 268 of file ider_evaluator.h.

10.77.4.14 int coco::func_id_eval::update (const expression_node & *__data*, const interval & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 405 of file ider_evaluator.h.

10.77.4.15 int coco::func_id_eval::update (const interval & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_id_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 399 of file ider_evaluator.h.

10.77.4.16 return_value coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< func_id_eval_type, expression_node, interval,expression_-const_walker >.

Definition at line 762 of file evaluator.h.

10.77.4.17 int coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< func_id_eval_type, expression_node, interval,expression_-const_walker >.

Definition at line 756 of file evaluator.h.

10.77.4.18 void coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::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 771 of file evaluator.h.

10.77.4.19 return_value coco::cached_forward_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< func_id_eval_type, expression_node, interval,expression_-const_walker >.

Definition at line 767 of file evaluator.h.

10.77.5 Member Data Documentation

10.77.5.1 func_id_eval_type coco::_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.77.5.2 const variable_indicator* coco::cached_evaluator_base< func_id_eval_type , expression_node , interval , expression_const_walker >::v_ind [protected, inherited]

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:

- ider_evaluator.h

10.78 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

10.78.1 Detailed Description

This class is the visitor data type for the func_id_eval evaluator.

Definition at line 131 of file ider_evaluator.h.

10.78.2 Member Data Documentation

10.78.2.1 interval_st coco::func_id_eval_type::d

Definition at line 138 of file ider_evaluator.h.

10.78.2.2 std::vector<interval>* coco::func_id_eval_type::f

the function range cache

Definition at line 135 of file ider_evaluator.h.

10.78.2.3 std::vector<std::vector<interval> >* coco::func_id_eval_type::id_data

the interval derivative data

Definition at line 136 of file ider_evaluator.h.

10.78.2.4 unsigned int coco::func_id_eval_type::info

Definition at line 138 of file ider_evaluator.h.

10.78.2.5 const model* coco::func_id_eval_type::mod

the DAG

Definition at line 137 of file ider_evaluator.h.

10.78.2.6 unsigned int coco::func_id_eval_type::n

children counter

Definition at line 140 of file ider_evaluator.h.

10.78.2.7 void* coco::func_id_eval_type::p

Definition at line 138 of file ider_evaluator.h.

10.78.2.8 interval coco::func_id_eval_type::r

return value

Definition at line 139 of file ider_evaluator.h.

10.78.2.9 const std::vector<interval>* coco::func_id_eval_type::range

the ranges of all nodes

Definition at line 134 of file ider_evaluator.h.

10.78.2.10 union { ... } coco::func_id_eval_type::u

additional data for complex nodes

10.78.2.11 const std::vector<interval>* coco::func_id_eval_type::x

the box

Definition at line 133 of file ider_evaluator.h.

The documentation for this struct was generated from the following file:

- ider_evaluator.h

10.79 coco::func_islp_eval Class Reference

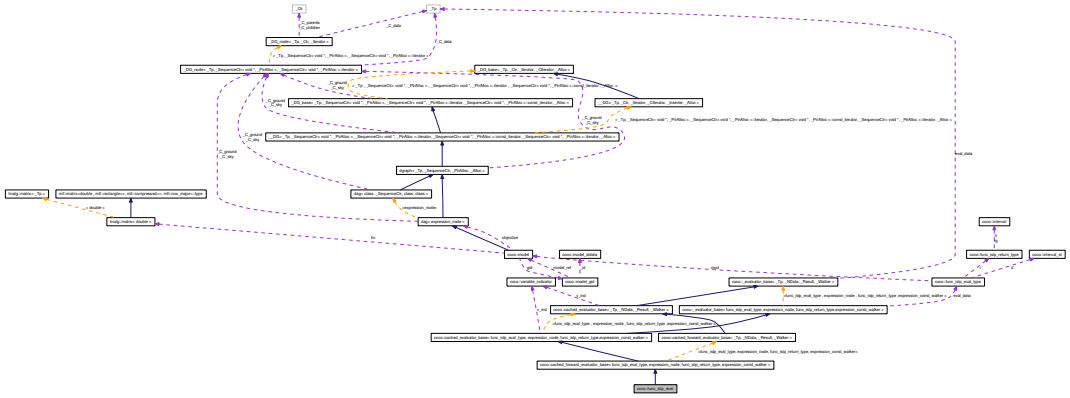
Forward function range evaluation with preparation of first order slope data.

`#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
- typedef func_islp_eval_type data_type

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< double > &__f)
- 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)
- 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 cleanup (const node_data_type &__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 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 node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- func_islp_eval_type eval_data

10.79.1 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.

Definition at line 162 of file islp_evaluator.h.

10.79.2 Member Typedef Documentation

10.79.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.79.2.2 typedef func_islp_eval_type coco::_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.79.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.79.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.79.3 Constructor & Destructor Documentation

10.79.3.1 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< double > & *__f*) [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 196 of file islp_evaluator.h.

10.79.3.2 coco::func_islp_eval::func_islp_eval (const func_islp_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 215 of file islp_evaluator.h.

10.79.3.3 coco::func_islp_eval::~func_islp_eval () [inline]

Standard Destructor

Definition at line 218 of file islp_evaluator.h.

10.79.4 Member Function Documentation

10.79.4.1 void coco::func_islp_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 313 of file islp_evaluator.h.

10.79.4.2 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::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 1419 of file islp_evaluator.h.

10.79.4.3 virtual void coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::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 804 of file evaluator.h.

10.79.4.4 int coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.79.4.5 int coco::func_islp_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 247 of file islp_evaluator.h.

10.79.4.6 void coco::func_islp_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 245 of file islp_evaluator.h.

10.79.4.7 bool coco::func_islp_eval::is_cached (const node_data_type & *__data*) [inline, protected, virtual]

This function determines, whether the function value and first order slope information for this node are already available.

Reimplemented from coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 174 of file islp_evaluator.h.

10.79.4.8 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 227 of file islp_evaluator.h.

10.79.4.9 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 236 of file islp_evaluator.h.

10.79.4.10 void coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.79.4.11 int coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.79.4.12 void coco::func_islp_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 326 of file islp_evaluator.h.

10.79.4.13 expression_const_walker coco::func_islp_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 221 of file islp_evaluator.h.

10.79.4.14 int coco::func_islp_eval::update (const expression_node & *__data*, const func_islp_return_type & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 354 of file islp_evaluator.h.

10.79.4.15 int coco::func_islp_eval::update (const func_islp_return_type & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type, expression_const_walker >.

Definition at line 348 of file islp_evaluator.h.

10.79.4.16 return_value coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::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. It is translated to a call to calculate_value with parameter false.

Reimplemented from coco::_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.79.4.17 int coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.79.4.18 void coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::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 771 of file evaluator.h.

10.79.4.19 return_value coco::cached_forward_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< func_islp_eval_type, expression_node, func_islp_return_type,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.79.5 Member Data Documentation

10.79.5.1 func_islp_eval_type coco::_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.79.5.2 const variable_indicator* coco::cached_evaluator_base< func_islp_eval_type , expression_node , func_islp_return_type , expression_const_walker >::v_ind [protected, inherited]

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:

- islp_evaluator.h

10.80 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< double > * 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

10.80.1 Detailed Description

This class is the visitor data type for the func_id_eval evaluator.

Definition at line 142 of file islp_evaluator.h.

10.80.2 Member Data Documentation

10.80.2.1 interval_st coco::func_islp_eval_type::d

Definition at line 149 of file islp_evaluator.h.

10.80.2.2 std::vector<double>* coco::func_islp_eval_type::f

the central function value cache

Definition at line 146 of file islp_evaluator.h.

10.80.2.3 unsigned int coco::func_islp_eval_type::info

Definition at line 149 of file islp_evaluator.h.

10.80.2.4 std::vector<std::vector<interval> >* coco::func_islp_eval_type::islp_data

the first order slope data

Definition at line 147 of file islp_evaluator.h.

10.80.2.5 const model* coco::func_islp_eval_type::mod

the DAG

Definition at line 148 of file islp_evaluator.h.

10.80.2.6 unsigned int coco::func_islp_eval_type::n

children counter

Definition at line 152 of file islp_evaluator.h.

10.80.2.7 void* coco::func_islp_eval_type::p

Definition at line 149 of file islp_evaluator.h.

10.80.2.8 func_islp_return_type coco::func_islp_eval_type::r

return value

Definition at line 151 of file islp_evaluator.h.

10.80.2.9 const std::vector<interval>* coco::func_islp_eval_type::range

the ranges of all nodes

Definition at line 145 of file islp_evaluator.h.

10.80.2.10 union { ... } coco::func_islp_eval_type::u

additional data for complex nodes

10.80.2.11 const std::vector<double>* coco::func_islp_eval_type::z

the center

Definition at line 144 of file islp_evaluator.h.

The documentation for this struct was generated from the following file:

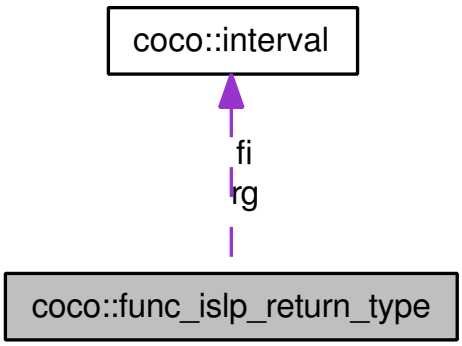
- islp_evaluator.h

10.81 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

- double f
- interval rg
- interval fi

10.81.1 Detailed Description

This is the return type of the func_islp_eval evaluator.

Definition at line 48 of file islp_evaluator.h.

10.81.2 Member Data Documentation

10.81.2.1 double coco::func_islp_return_type::f

Function value at the center

Definition at line 50 of file islp_evaluator.h.

10.81.2.2 interval coco::func_islp_return_type::fi

Enclosure of the function value at the center

Definition at line 52 of file islp_evaluator.h.

10.81.2.3 interval coco::func_islp_return_type::rg

Function range

Definition at line 51 of file islp_evaluator.h.

The documentation for this struct was generated from the following file:

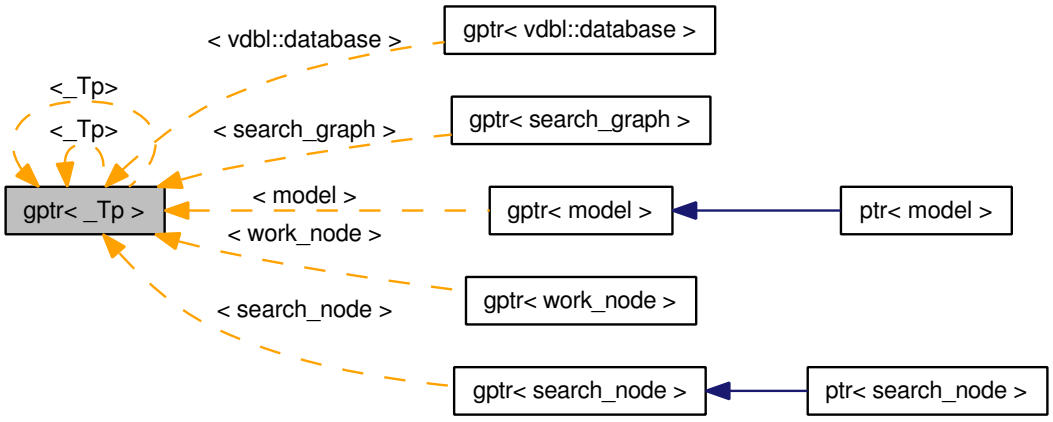
- islp_evaluator.h

10.82 gptr< _Tp > Class Template Reference

Global pointer class.

`#include <gptr.h>`

Inheritance diagram for gptr< _Tp >:



Public Member Functions

- virtual ~gptr ()

10.82.1 Detailed Description

template<class _Tp> class gptr< _Tp >

This class is a generalization of a pointer which should be adaptable to various distributed and parallel computing paradigms.

Definition at line 40 of file gptr.h.

10.82.2 Constructor & Destructor Documentation

10.82.2.1 template<class _Tp> virtual gptr< _Tp >::~gptr () [inline, virtual]

Standard Destructor

Definition at line 53 of file gptr.h.

The documentation for this class was generated from the following file:

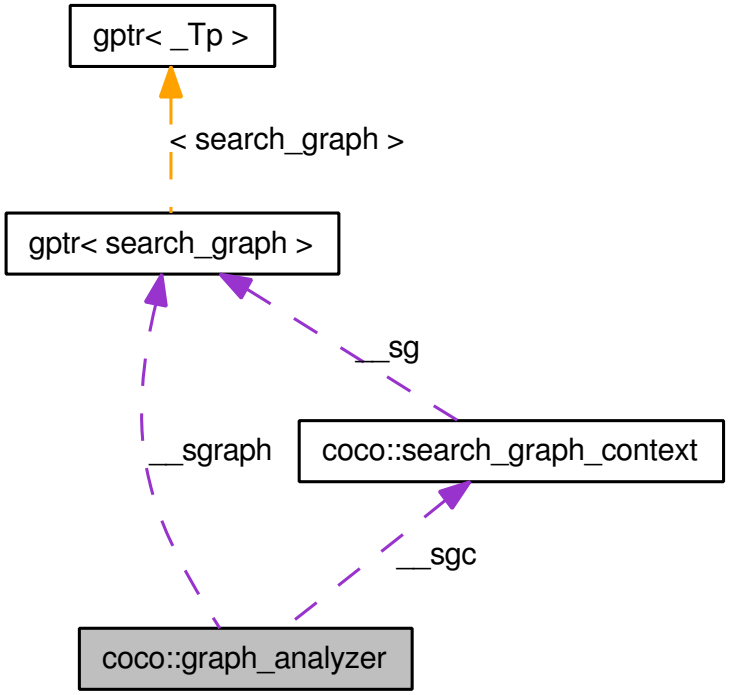
- gptr.h

10.83 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

10.83.1 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.

Definition at line 83 of file graph_analyzer.h.

10.83.2 Constructor & Destructor Documentation

10.83.2.1 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.

10.83.2.2 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.

10.83.2.3 virtual coco::graph_analyzer::~graph_analyzer () [inline, virtual]

Standard Destructor

Definition at line 121 of file graph_analyzer.h.

10.83.3 Member Function Documentation

10.83.3.1 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.

10.83.3.2 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.

10.83.3.3 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.

10.83.3.4 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.

10.83.4 Member Data Documentation

10.83.4.1 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.

10.83.4.2 const search_focus* coco::graph_analyzer::__sfoc [protected]

This is the current search focus.

Definition at line 91 of file graph_analyzer.h.

10.83.4.3 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.

10.83.4.4 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.

10.83.4.5 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:

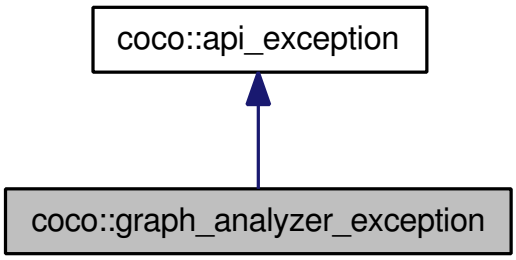
- graph_analyzer.h

10.84 coco::graph_analyzer_exception Class Reference

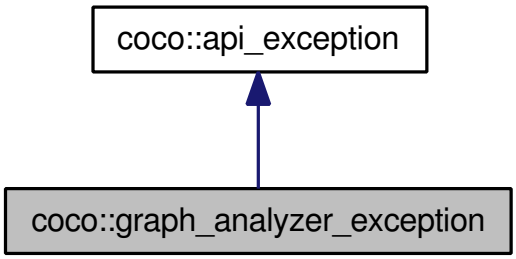
Graph analyzer exception class.

`#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 api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.84.1 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.

Definition at line 56 of file graph_analyzer.h.

10.84.2 Constructor & Destructor Documentation

10.84.2.1 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.

10.84.2.2 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.

10.84.2.3 virtual coco::graph_analyzer_exception::~graph_analyzer_exception () throw () [inline, virtual]

Standard Destructor

Definition at line 67 of file graph_analyzer.h.

10.84.3 Member Function Documentation

10.84.3.1 virtual std::string coco::api_exception::message () const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file api_exception.h.

10.84.3.2 virtual api_exception_type coco::api_exception::type () const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file api_exception.h.

10.84.3.3 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.

10.84.3.4 virtual char const* coco::api_exception::what () const throw () [inline, virtual, inherited]

This method returns the message as C-string.

Definition at line 89 of file api_exception.h.

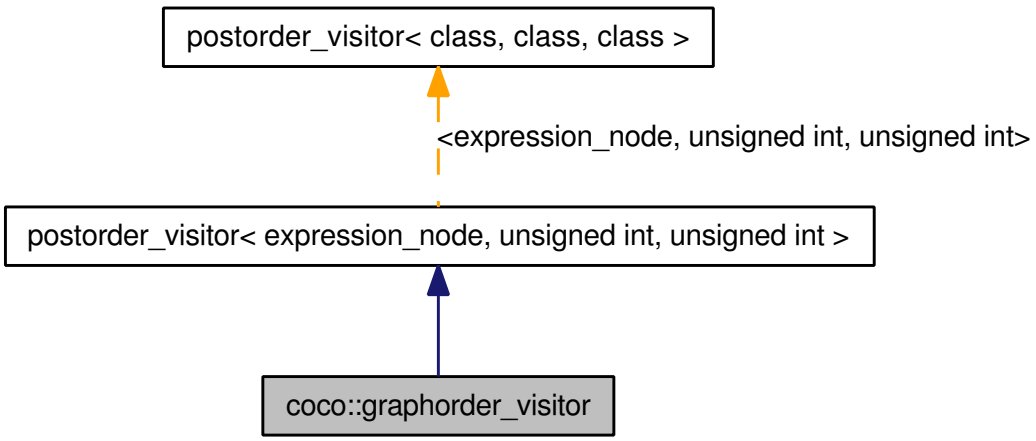
The documentation for this class was generated from the following file:

- graph_analyzer.h

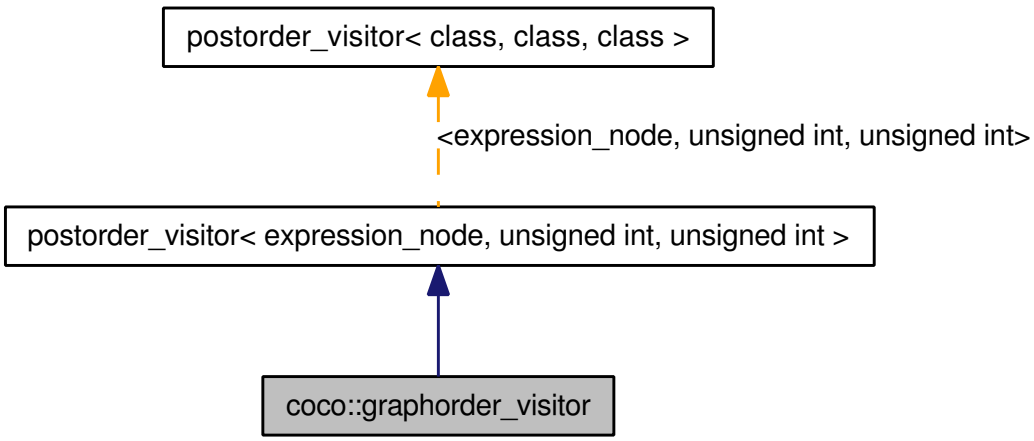
10.85 coco::graphorder_visitor Class Reference

This visitor class is used for computing a graph order.

Inheritance diagram for coco::graphorder_visitor:



Collaboration diagram for coco::graphorder_visitor:



Public Member Functions

- graphorder_visitor (std::vector< unsigned int > &__v, std::vector< bool > &__b, unsigned int __k=0)
- graphorder_visitor (const graphorder_visitor &__c)
- virtual void **vinit** ()
- virtual void **init** ()
- virtual bool **postorder** (const _Node &__n)
- virtual void **collect** (const _Node &__n, collect_value __r)

- 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)

10.85.1 Detailed Description

This postorder visitor computes an order of the nodes compatible with the DAG structure.

Definition at line 41 of file graphorder.cc.

10.85.2 Constructor & Destructor Documentation

10.85.2.1 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 55 of file graphorder.cc.

10.85.2.2 coco::graphorder_visitor::graphorder_visitor (const graphorder_visitor & __c) [inline]

Standard Copy Constructor

Definition at line 62 of file graphorder.cc.

10.85.3 Member Function Documentation

10.85.3.1 virtual void postorder_visitor< class, class, class >::collect (const _Node & __n, collect_value __r) [virtual, inherited]

10.85.3.2 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 82 of file graphorder.cc.

10.85.3.3 virtual void postorder_visitor< class, class, class >::init () [virtual, inherited]

10.85.3.4 virtual bool postorder_visitor< class, class, class >::postorder (const _Node & __n) [virtual, inherited]

10.85.3.5 bool coco::graphorder_visitor::postorder (const expression_node & r) [inline]

This is a method required for a postorder visitor.

Definition at line 71 of file graphorder.cc.

10.85.3.6 unsigned int coco::graphorder_visitor::value () [inline]

This is a method required for a postorder visitor.

Definition at line 69 of file graphorder.cc.

10.85.3.7 void coco::graphorder_visitor::vcollect (unsigned int __v) [inline]

This is a method required for a postorder visitor.

Definition at line 87 of file graphorder.cc.

10.85.3.8 virtual void postorder_visitor< class, class, class >::vinit () [virtual, inherited]

10.85.3.9 unsigned int coco::graphorder_visitor::vvalue () [inline, virtual]

This is a method required for a postorder visitor.

Reimplemented from **postorder_visitor< expression_node, unsigned int, unsigned int >**.

Definition at line 68 of file graphorder.cc.

The documentation for this class was generated from the following file:

- graphorder.cc

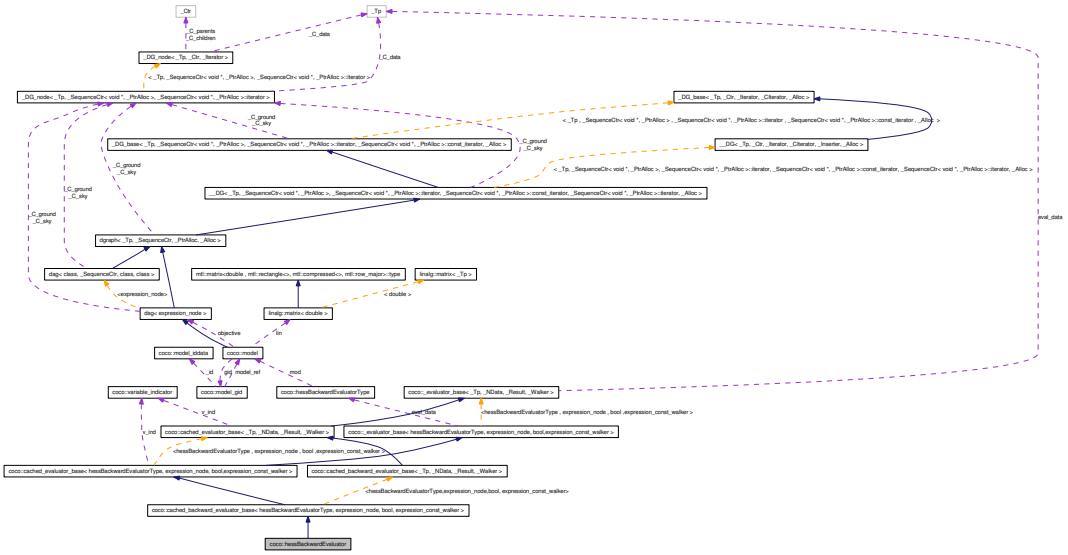
10.86 coco::hessBackwardEvaluator Class Reference

`#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
- typedef hessBackwardEvaluatorType data_type

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 void retrieve_from_cache (const node_data_type &__data)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- hessBackwardEvaluatorType eval_data

10.86.1 Detailed Description

Definition at line 1167 of file hess_evaluator.h.

10.86.2 Member Typedef Documentation

10.86.2.1 typedef _Base::const_walker coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 853 of file evaluator.h.

10.86.2.2 typedef hessBackwardEvaluatorType coco::_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.86.2.3 typedef _Base::node_data_type coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 849 of file evaluator.h.

10.86.2.4 typedef _Base::return_value coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 851 of file evaluator.h.

10.86.3 Constructor & Destructor Documentation

10.86.3.1 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 1189 of file hess_evaluator.h.

10.86.3.2 coco::hessBackwardEvaluator::hessBackwardEvaluator (const hessBackwardEvaluator & *__he*) [inline]

Definition at line 1219 of file hess_evaluator.h.

10.86.3.3 coco::hessBackwardEvaluator::~hessBackwardEvaluator () [inline]

Definition at line 1221 of file hess_evaluator.h.

10.86.4 Member Function Documentation

10.86.4.1 int coco::hessBackwardEvaluator::calculate (const expression_node & *__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.

Reimplemented from coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1261 of file hess_evaluator.h.

10.86.4.2 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::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1500 of file hess_evaluator.h.

10.86.4.3 void coco::hessBackwardEvaluator::cleanup (const expression_node & *__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.

Reimplemented from coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1398 of file hess_evaluator.h.

10.86.4.4 int coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 879 of file evaluator.h.

10.86.4.5 void coco::hessBackwardEvaluator::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1250 of file hess_evaluator.h.

10.86.4.6 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::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1176 of file hess_evaluator.h.

10.86.4.7 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 1223 of file hess_evaluator.h.

10.86.4.8 void coco::hessBackwardEvaluator::newResult (std::vector< double > * *__grad*, std::vector< double > * *__hess*) [inline]

Definition at line 1233 of file hess_evaluator.h.

10.86.4.9 void coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated into a call to cleanup.

Reimplemented from coco::_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 874 of file evaluator.h.

10.86.4.10 int coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Reimplemented from coco::cached_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 862 of file evaluator.h.

10.86.4.11 virtual void coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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 929 of file evaluator.h.

10.86.4.12 void coco::hessBackwardEvaluator::set_mult (double *scal*) [inline]

Definition at line 1239 of file hess_evaluator.h.

10.86.4.13 expression_const_walker coco::hessBackwardEvaluator::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 1246 of file hess_evaluator.h.

10.86.4.14 int coco::hessBackwardEvaluator::update (const expression_node & *__data*, const bool & *__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.

Reimplemented from coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1446 of file hess_evaluator.h.

10.86.4.15 int coco::hessBackwardEvaluator::update (const bool & *__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.

Reimplemented from coco::cached_backward_evaluator_base< hessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1436 of file hess_evaluator.h.

10.86.4.16 return_value coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter false.

Reimplemented from coco::_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 891 of file evaluator.h.

10.86.4.17 int coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 885 of file evaluator.h.

10.86.4.18 void coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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 900 of file evaluator.h.

10.86.4.19 return_value coco::cached_backward_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< hessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 896 of file evaluator.h.

10.86.5 Member Data Documentation

10.86.5.1 hessBackwardEvaluatorType coco::_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.86.5.2 const variable_indicator* coco::cached_evaluator_base< hessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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:

- hess_evaluator.h

10.87 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

10.87.1 Detailed Description

Definition at line 1145 of file hess_evaluator.h.

10.87.2 Member Data Documentation

10.87.2.1 bool coco::hessBackwardEvaluatorType::calc_grad

Definition at line 1153 of file hess_evaluator.h.

10.87.2.2 unsigned int coco::hessBackwardEvaluatorType::child_n

Definition at line 1162 of file hess_evaluator.h.

10.87.2.3 std::vector<std::vector<double> >* coco::hessBackwardEvaluatorType::d_cache

Definition at line 1149 of file hess_evaluator.h.

10.87.2.4 std::vector<std::vector<double> >* coco::hessBackwardEvaluatorType::d_data

Definition at line 1147 of file hess_evaluator.h.

10.87.2.5 double coco::hessBackwardEvaluatorType::d_mult

Definition at line 1155 of file hess_evaluator.h.

10.87.2.6 double coco::hessBackwardEvaluatorType::d_mult_trans

Definition at line 1158 of file hess_evaluator.h.

10.87.2.7 std::vector<double>* coco::hessBackwardEvaluatorType::grad_vec

Definition at line 1151 of file hess_evaluator.h.

10.87.2.8 std::vector<std::vector<double> >* coco::hessBackwardEvaluatorType::h_cache

Definition at line 1150 of file hess_evaluator.h.

10.87.2.9 std::vector<std::vector<double> >* coco::hessBackwardEvaluatorType::h_data

Definition at line 1148 of file hess_evaluator.h.

10.87.2.10 double coco::hessBackwardEvaluatorType::h_mult

Definition at line 1159 of file hess_evaluator.h.

10.87.2.11 double coco::hessBackwardEvaluatorType::h_mult_trans

Definition at line 1160 of file hess_evaluator.h.

10.87.2.12 std::vector<double>* coco::hessBackwardEvaluatorType::hess_vec

Definition at line 1152 of file hess_evaluator.h.

10.87.2.13 bool coco::hessBackwardEvaluatorType::hess_zero

Definition at line 1157 of file hess_evaluator.h.

10.87.2.14 double coco::hessBackwardEvaluatorType::hessian_of_current_node

Definition at line 1156 of file hess_evaluator.h.

10.87.2.15 bool coco::hessBackwardEvaluatorType::is_linear

Definition at line 1161 of file hess_evaluator.h.

10.87.2.16 const model* coco::hessBackwardEvaluatorType::mod

Definition at line 1154 of file hess_evaluator.h.

The documentation for this struct was generated from the following file:

- hess_evaluator.h

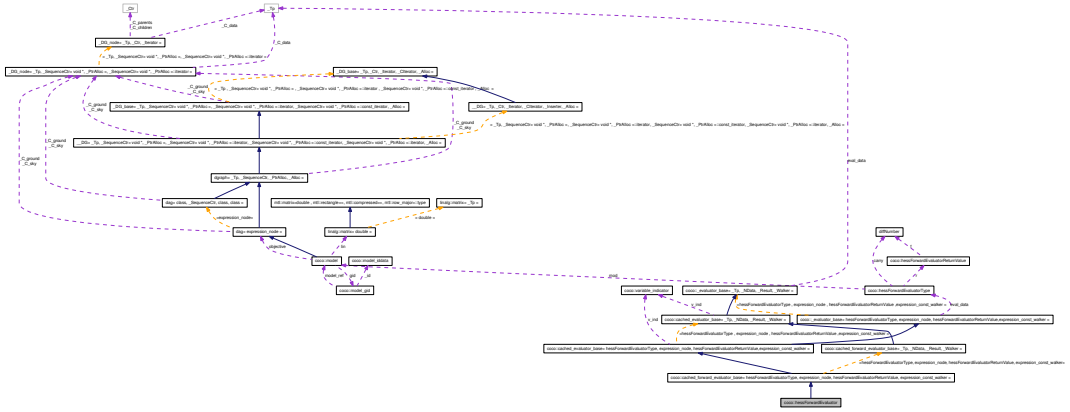
10.88 coco::hessForwardEvaluator Class Reference

`#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
- typedef hessForwardEvaluatorType data_type

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 void cleanup (const node_data_type &__data)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- hessForwardEvaluatorType eval_data

10.88.1 Detailed Description

Definition at line 168 of file hess_evaluator.h.

10.88.2 Member Typedef Documentation

10.88.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.88.2.2 typedef hessForwardEvaluatorType coco::_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.88.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.88.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.88.3 Constructor & Destructor Documentation

10.88.3.1 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.

10.88.3.2 coco::hessForwardEvaluator::hessForwardEvaluator (const hessForwardEvaluator & *__x*) [inline]

Definition at line 287 of file hess_evaluator.h.

10.88.3.3 coco::hessForwardEvaluator::~hessForwardEvaluator () [inline]

Definition at line 291 of file hess_evaluator.h.

10.88.4 Member Function Documentation

10.88.4.1 void coco::hessForwardEvaluator::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 406 of file hess_evaluator.h.

10.88.4.2 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::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 1136 of file hess_evaluator.h.

10.88.4.3 virtual void coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::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 804 of file evaluator.h.

10.88.4.4 int coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.88.4.5 int coco::hessForwardEvaluator::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 330 of file hess_evaluator.h.

10.88.4.6 void coco::hessForwardEvaluator::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 321 of file hess_evaluator.h.

10.88.4.7 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::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 177 of file hess_evaluator.h.

10.88.4.8 void coco::hessForwardEvaluator::newPoint (const std::vector< diffNumber > & *__x*, const variable_indicator & *__v*) [inline]

Definition at line 296 of file hess_evaluator.h.

10.88.4.9 void coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.88.4.10 int coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.88.4.11 void coco::hessForwardEvaluator::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 417 of file hess_evaluator.h.

10.88.4.12 expression_const_walker coco::hessForwardEvaluator::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 293 of file hess_evaluator.h.

10.88.4.13 virtual int coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::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 826 of file evaluator.h.

10.88.4.14 int coco::hessForwardEvaluator::update (const expression_node & *__data*, const hessForwardEvaluatorReturnValue & *__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.

Reimplemented from coco::cached_forward_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 434 of file hess_evaluator.h.

10.88.4.15 int coco::hessForwardEvaluator::update (const diffNumber & *__rval*) [inline]

Definition at line 424 of file hess_evaluator.h.

10.88.4.16 return_value coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.88.4.17 int coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.88.4.18 void coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::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 771 of file evaluator.h.

10.88.4.19 return_value coco::cached_forward_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< hessForwardEvaluatorType, expression_node, hessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.88.5 Member Data Documentation

10.88.5.1 hessForwardEvaluatorType coco::_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.88.5.2 const variable_indicator* coco::cached_evaluator_base< hessForwardEvaluatorType , expression_node , hessForwardEvaluatorReturnValue , expression_const_walker >::v_ind [protected, inherited]

For caching a variable indicator is needed to check which nodes need not be re-evaluated.

Definition at line 427 of file evaluator.h.

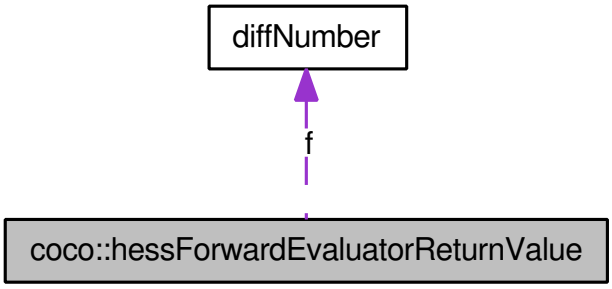
The documentation for this class was generated from the following file:

- hess_evaluator.h

10.89 coco::hessForwardEvaluatorReturnValue Struct Reference

`#include <hess_evaluator.h>`

Collaboration diagram for coco::hessForwardEvaluatorReturnValue:



Public Attributes

- diffNumber f
- unsigned int nn
- tristate in_chn

10.89.1 Detailed Description

Definition at line 144 of file hess_evaluator.h.

10.89.2 Member Data Documentation

10.89.2.1 diffNumber coco::hessForwardEvaluatorReturnValue::f

Definition at line 146 of file hess_evaluator.h.

10.89.2.2 tristate coco::hessForwardEvaluatorReturnValue::in_chn

Definition at line 149 of file hess_evaluator.h.

10.89.2.3 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

10.90 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

10.90.1 Detailed Description

Definition at line 152 of file hess_evaluator.h.

10.90.2 Member Data Documentation

10.90.2.1 std::vector<bool>* coco::hessForwardEvaluatorType::b

Definition at line 159 of file hess_evaluator.h.

10.90.2.2 diffNumber coco::hessForwardEvaluatorType::carry

Definition at line 161 of file hess_evaluator.h.

10.90.2.3 double coco::hessForwardEvaluatorType::d

Definition at line 163 of file hess_evaluator.h.

10.90.2.4 std::vector<std::vector<double> >* coco::hessForwardEvaluatorType::d_data

Definition at line 157 of file hess_evaluator.h.

10.90.2.5 std::vector<diffNumber>* coco::hessForwardEvaluatorType::f_cache

Definition at line 155 of file hess_evaluator.h.

10.90.2.6 std::vector<std::vector<double> >* coco::hessForwardEvaluatorType::h_data

Definition at line 156 of file hess_evaluator.h.

10.90.2.7 unsigned int coco::hessForwardEvaluatorType::info

Definition at line 164 of file hess_evaluator.h.

10.90.2.8 const model* coco::hessForwardEvaluatorType::mod

Definition at line 162 of file hess_evaluator.h.

10.90.2.9 unsigned int coco::hessForwardEvaluatorType::n

Definition at line 164 of file hess_evaluator.h.

10.90.2.10 void* coco::hessForwardEvaluatorType::p

Definition at line 163 of file hess_evaluator.h.

10.90.2.11 hessForwardEvaluatorReturnValue coco::hessForwardEvaluatorType::r

Definition at line 160 of file hess_evaluator.h.

10.90.2.12 std::vector<unsigned int>* coco::hessForwardEvaluatorType::t

Definition at line 158 of file hess_evaluator.h.

10.90.2.13 union { ... } coco::hessForwardEvaluatorType::u

10.90.2.14 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

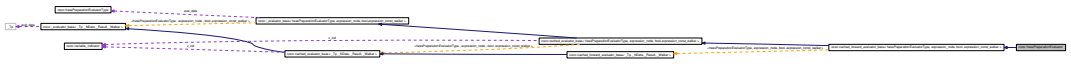
10.91 coco::hessPreparationEvaluator Class Reference

`#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
- typedef hessPreparationEvaluatorType data_type

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 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 const_walker short_cut_to (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- hessPreparationEvaluatorType eval_data

10.91.1 Detailed Description

Definition at line 78 of file hess_evaluator.h.

10.91.2 Member Typedef Documentation

10.91.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.91.2.2 typedef hessPreparationEvaluatorType coco::_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.91.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.91.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.91.3 Constructor & Destructor Documentation

10.91.3.1 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.

10.91.3.2 coco::hessPreparationEvaluator::hessPreparationEvaluator (const hessPreparationEvaluator & *__x*) [inline]

Definition at line 102 of file hess_evaluator.h.

10.91.3.3 coco::hessPreparationEvaluator::~hessPreparationEvaluator () [inline]

Definition at line 105 of file hess_evaluator.h.

10.91.4 Member Function Documentation

10.91.4.1 void coco::hessPreparationEvaluator::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 126 of file hess_evaluator.h.

10.91.4.2 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::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 138 of file hess_evaluator.h.

10.91.4.3 virtual void coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 804 of file evaluator.h.

10.91.4.4 int coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.91.4.5 int coco::hessPreparationEvaluator::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 116 of file hess_evaluator.h.

10.91.4.6 void coco::hessPreparationEvaluator::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 107 of file hess_evaluator.h.

10.91.4.7 bool coco::hessPreparationEvaluator::is_cached (const expression_node & *__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 from coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 109 of file hess_evaluator.h.

10.91.4.8 void coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.91.4.9 int coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.91.4.10 void coco::hessPreparationEvaluator::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< hessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 114 of file hess_evaluator.h.

10.91.4.11 virtual const_walker coco::cached_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 450 of file evaluator.h.

10.91.4.12 virtual int coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 826 of file evaluator.h.

10.91.4.13 virtual int coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 814 of file evaluator.h.

10.91.4.14 int coco::hessPreparationEvaluator::update (const expression_node & *__data*, bool *__rval*) [inline]

Definition at line 130 of file hess_evaluator.h.

10.91.4.15 int coco::hessPreparationEvaluator::update (bool *__rval*) [inline]

Definition at line 128 of file hess_evaluator.h.

10.91.4.16 return_value coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.91.4.17 int coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.91.4.18 void coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 771 of file evaluator.h.

10.91.4.19 return_value coco::cached_forward_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< hessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.91.5 Member Data Documentation

10.91.5.1 hessPreparationEvaluatorType coco::_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.91.5.2 const variable_indicator* coco::cached_evaluator_base< hessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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:

- hess_evaluator.h

10.92 coco::hessPreparationEvaluatorType Struct Reference

`#include <hess_evaluator.h>`

Public Attributes

- std::vector< std::vector< double > > * d
- std::vector< std::vector< double > > * h

10.92.1 Detailed Description

Definition at line 72 of file hess_evaluator.h.

10.92.2 Member Data Documentation

10.92.2.1 std::vector<std::vector<double> >* coco::hessPreparationEvaluatorType::d

Definition at line 74 of file hess_evaluator.h.

10.92.2.2 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

10.93 coco::ider_eval Class Reference

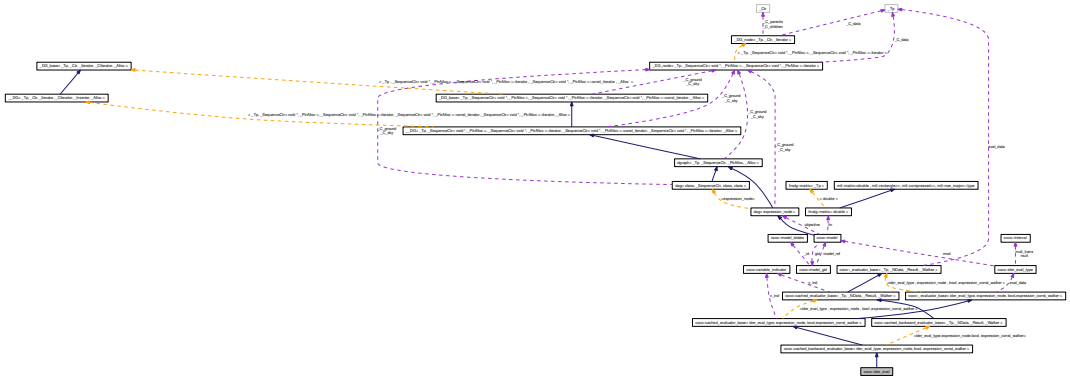
Backward interval gradient evaluation with prepared interval derivative data.

`#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
- typedef ider_eval_type data_type

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 ()

- 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)

Protected Attributes

- const variable_indicator * v_ind
- ider_eval_type eval_data

10.93.1 Detailed Description

This class is a cached backward evaluator performing an interval gradient evaluation with available partial interval derivative information.

Definition at line 1293 of file ider_evaluator.h.

10.93.2 Member Typedef Documentation

10.93.2.1 typedef _Base::const_walker coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 853 of file evaluator.h.

10.93.2.2 typedef ider_eval_type coco::_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.93.2.3 typedef _Base::node_data_type coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 849 of file evaluator.h.

10.93.2.4 typedef _Base::return_value coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 851 of file evaluator.h.

10.93.3 Constructor & Destructor Documentation

10.93.3.1 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 1327 of file ider_evaluator.h.

10.93.3.2 coco::ider_eval::ider_eval (const ider_eval & *__d*) [inline]

Standard Copy Constructor

Definition at line 1343 of file ider_evaluator.h.

10.93.3.3 coco::ider_eval::~ider_eval () [inline]

Standard Destructor

Definition at line 1346 of file ider_evaluator.h.

10.93.4 Member Function Documentation

10.93.4.1 int coco::ider_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1386 of file ider_evaluator.h.

10.93.4.2 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::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1483 of file ider_evaluator.h.

10.93.4.3 void coco::ider_eval::cleanup (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1439 of file ider_evaluator.h.

10.93.4.4 int coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< ider_eval_type, expression_node, bool,expression_const_-walker >.

Definition at line 879 of file evaluator.h.

10.93.4.5 void coco::ider_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1380 of file ider_evaluator.h.

10.93.4.6 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::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1305 of file ider_evaluator.h.

10.93.4.7 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 1351 of file ider_evaluator.h.

10.93.4.8 void coco::ider_eval::new_result (std::vector< interval > & *__grad*) [inline]

This method changes the result vector to __grad.

Definition at line 1361 of file ider_evaluator.h.

10.93.4.9 void coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::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. It is translated into a call to cleanup.

Reimplemented from coco::_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 874 of file evaluator.h.

10.93.4.10 int coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Reimplemented from coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 862 of file evaluator.h.

10.93.4.11 void coco::ider_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1450 of file ider_evaluator.h.

10.93.4.12 void coco::ider_eval::set_mult (const interval & *scal*) [inline]

This method causes the gradient to be multiplied by `scal`.

Definition at line 1367 of file ider_evaluator.h.

10.93.4.13 expression_const_walker coco::ider_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< ider_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 1374 of file ider_evaluator.h.

10.93.4.14 int coco::ider_eval::update (const expression_node & *__data*, const bool & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1464 of file ider_evaluator.h.

10.93.4.15 int coco::ider_eval::update (const bool & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< ider_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1457 of file ider_evaluator.h.

10.93.4.16 return_value coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 891 of file evaluator.h.

10.93.4.17 int coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 885 of file evaluator.h.

10.93.4.18 void coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::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 900 of file evaluator.h.

10.93.4.19 return_value coco::cached_backward_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< ider_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 896 of file evaluator.h.

10.93.5 Member Data Documentation

10.93.5.1 ider_eval_type coco::_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.93.5.2 const variable_indicator* coco::cached_evaluator_base< ider_eval_type , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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:

- ider_evaluator.h

10.94 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

10.94.1 Detailed Description

This class is the visitor data type for the ider_eval evaluator.

Definition at line 1275 of file ider_evaluator.h.

10.94.2 Member Data Documentation

10.94.2.1 unsigned int coco::ider_eval_type::child_n

children counter

Definition at line 1284 of file ider_evaluator.h.

10.94.2.2 std::vector<std::vector<interval> >* coco::ider_eval_type::d_cache

interval derivative cache

Definition at line 1278 of file ider_evaluator.h.

10.94.2.3 std::vector<interval>* coco::ider_eval_type::grad_vec

the interval gradient vector

Definition at line 1279 of file ider_evaluator.h.

10.94.2.4 std::vector<std::vector<interval> >* coco::ider_eval_type::id_data

partial interval derivative data

Definition at line 1277 of file ider_evaluator.h.

10.94.2.5 const model* coco::ider_eval_type::mod

the DAG

Definition at line 1281 of file ider_evaluator.h.

10.94.2.6 interval coco::ider_eval_type::mult

the current value on the path

Definition at line 1282 of file ider_evaluator.h.

10.94.2.7 interval coco::ider_eval_type::mult_trans

transfer variable for mult

Definition at line 1283 of file ider_evaluator.h.

10.94.2.8 const std::vector<interval>* coco::ider_eval_type::x

the evaluation box

Definition at line 1280 of file ider_evaluator.h.

The documentation for this struct was generated from the following file:

- ider_evaluator.h

10.95 coco::iderf_eval Class Reference

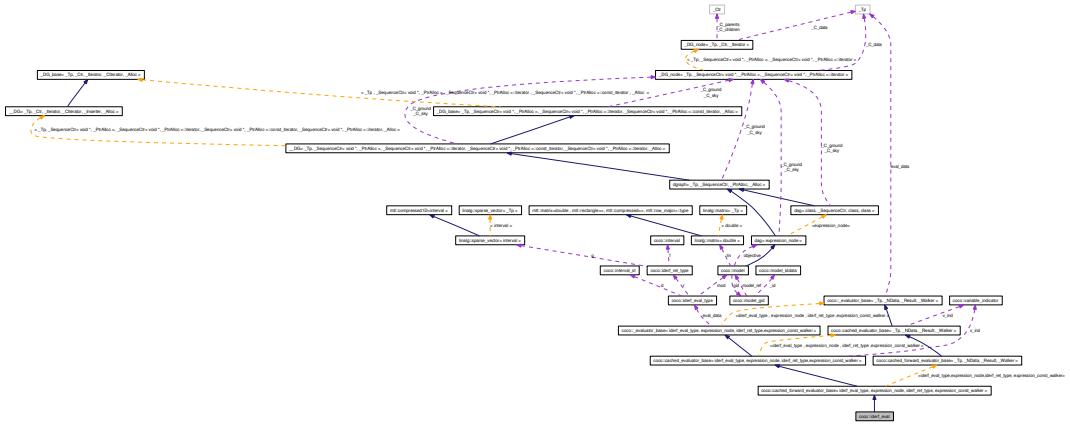
Forward function and derivative range evaluation.

`#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
- typedef iderf_eval_type data_type

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 void cleanup (const node_data_type &__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 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)

Protected Attributes

- const variable_indicator * v_ind
- iderf_eval_type eval_data

10.95.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural interval extension together with a derivative range evaluation.

Definition at line 85 of file iderf_evaluator.h.

10.95.2 Member Typedef Documentation

10.95.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.95.2.2 typedef iderf_eval_type coco::_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.95.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.95.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.95.3 Constructor & Destructor Documentation

10.95.3.1 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 155 of file iderf_evaluator.h.

10.95.3.2 coco::iderf_eval::iderf_eval (const iderf_eval & *__v*) [inline]

Standard Copy Constructor

Definition at line 171 of file iderf_evaluator.h.

10.95.3.3 coco::iderf_eval::~iderf_eval () [inline]

Standard Destructor

Definition at line 174 of file iderf_evaluator.h.

10.95.4 Member Function Documentation

10.95.4.1 void coco::iderf_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 252 of file iderf_evaluator.h.

10.95.4.2 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::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 860 of file iderf_evaluator.h.

10.95.4.3 virtual void coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::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 804 of file evaluator.h.

10.95.4.4 int coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.95.4.5 int coco::iderf_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 193 of file iderf_evaluator.h.

10.95.4.6 void coco::iderf_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 191 of file iderf_evaluator.h.

10.95.4.7 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::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 100 of file iderf_evaluator.h.

10.95.4.8 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 183 of file iderf_evaluator.h.

10.95.4.9 void coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.95.4.10 int coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.95.4.11 void coco::iderf_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 267 of file iderf_evaluator.h.

10.95.4.12 expression_const_walker coco::iderf_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 177 of file iderf_evaluator.h.

10.95.4.13 int coco::iderf_eval::update (const expression_node & *__data*, const iderf_ret_type & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 295 of file iderf_evaluator.h.

10.95.4.14 int coco::iderf_eval::update (const iderf_ret_type & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type, expression_const_walker >.

Definition at line 289 of file iderf_evaluator.h.

10.95.4.15 return_value coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.95.4.16 int coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.95.4.17 void coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::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 771 of file evaluator.h.

10.95.4.18 return_value coco::cached_forward_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< iderf_eval_type, expression_node, iderf_ret_type,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.95.5 Member Data Documentation

10.95.5.1 iderf_eval_type coco::_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.95.5.2 const variable_indicator* coco::cached_evaluator_base< iderf_eval_type , expression_node , iderf_ret_type , expression_const_walker >::v_ind [protected, inherited]

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:

- iderf_evaluator.h

10.96 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

10.96.1 Detailed Description

This class is the visitor data type for the iderf_eval evaluator.

Definition at line 63 of file iderf_evaluator.h.

10.96.2 Member Data Documentation

10.96.2.1 std::vector<iderf_ret_type>* coco::iderf_eval_type::cache

the function range cache

Definition at line 67 of file iderf_evaluator.h.

10.96.2.2 interval_st coco::iderf_eval_type::d

Definition at line 71 of file iderf_evaluator.h.

10.96.2.3 bool coco::iderf_eval_type::do_intersect

compute an intersection with the node range?

Definition at line 75 of file iderf_evaluator.h.

10.96.2.4 int coco::iderf_eval_type::info

Definition at line 71 of file iderf_evaluator.h.

10.96.2.5 const model* coco::iderf_eval_type::mod

the DAG

Definition at line 70 of file iderf_evaluator.h.

10.96.2.6 unsigned int coco::iderf_eval_type::n

children counter

Definition at line 74 of file iderf_evaluator.h.

10.96.2.7 void* coco::iderf_eval_type::p

Definition at line 71 of file iderf_evaluator.h.

10.96.2.8 iderf_ret_type coco::iderf_eval_type::r

return value

Definition at line 73 of file iderf_evaluator.h.

10.96.2.9 const std::vector<interval>* coco::iderf_eval_type::range

the ranges of all nodes

Definition at line 66 of file iderf_evaluator.h.

10.96.2.10 union { ... } coco::iderf_eval_type::u

additional data for complex nodes

10.96.2.11 const std::vector<interval>* coco::iderf_eval_type::x

the box

Definition at line 65 of file iderf_evaluator.h.

The documentation for this struct was generated from the following file:

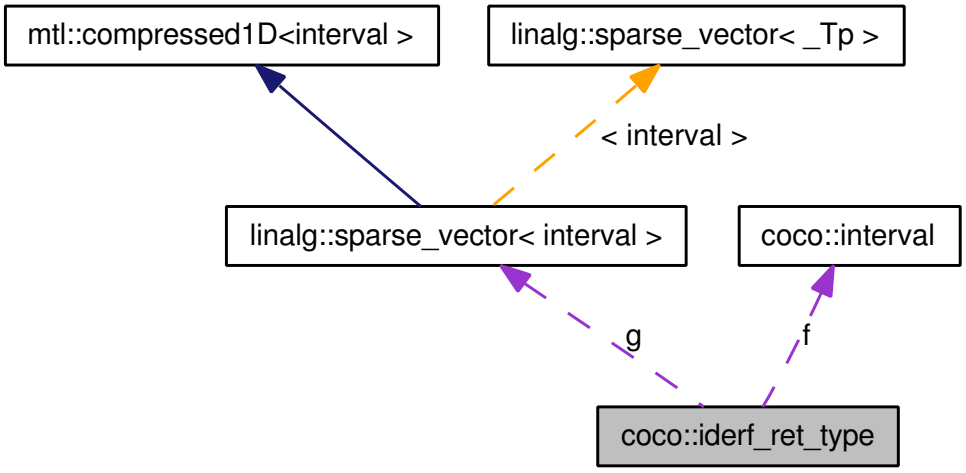
- iderf_evaluator.h

10.97 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
- linalg::sparse_vector< interval > g

10.97.1 Detailed Description

This class is the visitor data type for the return type of the iderf_eval evaluator.

Definition at line 48 of file iderf_evaluator.h.

10.97.2 Member Data Documentation

10.97.2.1 interval coco::iderf_ret_type::f

Definition at line 50 of file iderf_evaluator.h.

10.97.2.2 linalg::sparse_vector<interval> coco::iderf_ret_type::g

Definition at line 51 of file iderf_evaluator.h.

The documentation for this struct was generated from the following file:

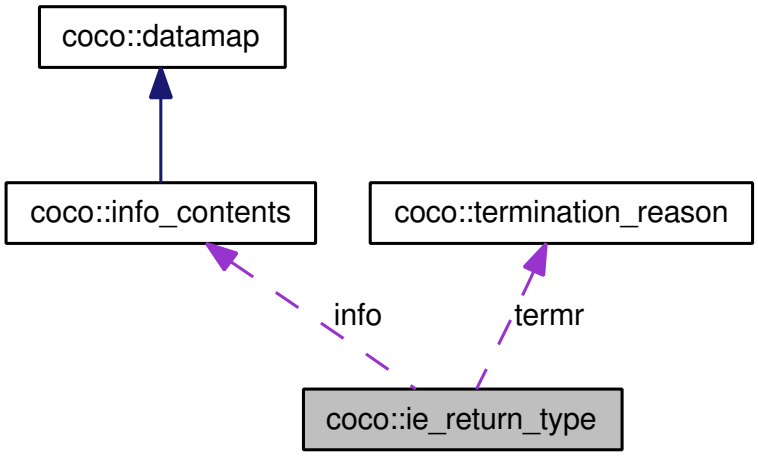
- iderf_evaluator.h

10.98 coco::ie_return_type Class Reference

The return class of all inference engines.

`#include <ie_rettype.h>`

Collaboration diagram for coco::ie_return_type:



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 linalg::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 linalg::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 linalg::matrix< _TS > *&__b, const linalg::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 linalg::matrix< _TS > *&__b, const linalg::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 linalg::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 linalg::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 linalg::matrix< _TS > *&__b, const linalg::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 linalg::matrix< _TS > *&__b, const linalg::matrix< _TS > *__def) const
- template<class _TS >
  bool retrieve_from_info_i (const char *__s, int i, _TS &__b, const _TS &__def) const

10.98.1 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.

Definition at line 49 of file ie_rettype.h.

10.98.2 Constructor & Destructor Documentation

10.98.2.1 coco::ie_return_type::ie_return_type () [inline]

Standard Constructor

Definition at line 71 of file ie_rettype.h.

10.98.2.2 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.

10.98.2.3 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.

10.98.2.4 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.

10.98.2.5 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.

10.98.2.6 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.

10.98.2.7 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.

10.98.2.8 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.

10.98.2.9 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.

10.98.2.10 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.

10.98.2.11 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.

10.98.2.12 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.

10.98.2.13 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.

10.98.2.14 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.

10.98.2.15 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.

10.98.2.16 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.

10.98.2.17 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.

10.98.2.18 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.

10.98.2.19 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.

10.98.2.20 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.

10.98.2.21 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.

10.98.2.22 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.

10.98.2.23 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.

10.98.2.24 virtual coco::ie_return_type::~ie_return_type () [inline, virtual]

Standard Destructor

Definition at line 238 of file ie_rettype.h.

10.98.3 Member Function Documentation

10.98.3.1 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.

10.98.3.2 const info_contents & coco::ie_return_type::get_info () const [inline]

This method accesses the whole info content.

Definition at line 33 of file ie_rettype.hpp.

10.98.3.3 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.

10.98.3.4 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 181 of file ie_rettype.hpp.

10.98.3.5 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 176 of file ie_rettype.hpp.

10.98.3.6 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 171 of file ie_rettype.hpp.

10.98.3.7 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 166 of file ie_rettype.hpp.

10.98.3.8 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 160 of file ie_rettype.hpp.

10.98.3.9 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 154 of file ie_rettype.hpp.

10.98.3.10 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 148 of file ie_rettype.hpp.

10.98.3.11 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 142 of file ie_rettype.hpp.

10.98.3.12 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 192 of file ie_rettype.hpp.

10.98.3.13 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 186 of file ie_rettype.hpp.

10.98.3.14 unsigned int coco::ie_return_type::n_deltas () const [inline]

This method returns the number of deltas defined.

Definition at line 469 of file ie_rettype.h.

10.98.3.15 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 69 of file ie_rettype.hpp.

10.98.3.16 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 48 of file ie_rettype.hpp.

10.98.3.17 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 79 of file ie_rettype.hpp.

10.98.3.18 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 58 of file ie_rettype.hpp.

10.98.3.19 ie_return_type & coco::ie_return_type::operator= (const ie_return_type & *__d*) [inline]

Standard Assignment Operator

Definition at line 105 of file ie_rettype.hpp.

10.98.3.20 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 295 of file ie_rettype.hpp.

10.98.3.21 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * *__s*, const linalg::matrix< _TS > *& *__b*, const linalg::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 288 of file ie_rettype.hpp.

10.98.3.22 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 281 of file ie_rettype.hpp.

10.98.3.23 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 254 of file ie_rettype.hpp.

10.98.3.24 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const linalg::matrix< _TS > *& __b, const linalg::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 247 of file ie_rettype.hpp.

10.98.3.25 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 240 of file ie_rettype.hpp.

10.98.3.26 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 275 of file ie_rettype.hpp.

10.98.3.27 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const char * __s, const linalg::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 268 of file ie_rettype.hpp.

10.98.3.28 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 261 of file ie_rettype.hpp.

10.98.3.29 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 233 of file ie_rettype.hpp.

10.98.3.30 template<class _TS > bool coco::ie_return_type::retrieve_from_info (const std::string & __s, const linalg::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 226 of file ie_rettype.hpp.

10.98.3.31 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 219 of file ie_rettype.hpp.

10.98.3.32 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 379 of file ie_rettype.hpp.

10.98.3.33 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * __s, int i, const linalg::matrix< _TS > *& __b, const linalg::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 372 of file ie_rettype.hpp.

10.98.3.34 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 365 of file ie_rettype.hpp.

10.98.3.35 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 337 of file ie_rettype.hpp.

10.98.3.36 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & *__s*, int *i*, const linalg::matrix< _TS > *& *__b*, const linalg::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 330 of file ie_rettype.hpp.

10.98.3.37 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 323 of file ie_rettype.hpp.

10.98.3.38 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 358 of file ie_rettype.hpp.

10.98.3.39 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const char * *__s*, int *i*, const linalg::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 351 of file ie_rettype.hpp.

10.98.3.40 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 344 of file ie_rettype.hpp.

10.98.3.41 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 316 of file ie_rettype.hpp.

10.98.3.42 template<class _TS > bool coco::ie_return_type::retrieve_from_info_i (const std::string & __s, int i, const linalg::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 309 of file ie_rettype.hpp.

10.98.3.43 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 302 of file ie_rettype.hpp.

10.98.3.44 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 124 of file ie_rettype.hpp.

10.98.3.45 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 118 of file ie_rettype.hpp.

10.98.3.46 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 136 of file ie_rettype.hpp.

10.98.3.47 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 130 of file ie_rettype.hpp.

10.98.3.48 void coco::ie_return_type::set_termination_reason (const termination_reason & *__t*) [inline]

This method sets the termination_reason to __t.

Definition at line 38 of file ie_rettype.hpp.

10.98.3.49 const termination_reason & coco::ie_return_type::term_reason () const [inline]

This method returns the reason of termination.

Definition at line 43 of file ie_rettype.hpp.

10.98.3.50 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 213 of file ie_rettype.hpp.

10.98.3.51 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 208 of file ie_rettype.hpp.

10.98.3.52 void coco::ie_return_type::unset_information (const char * __s) [inline]

This method removes the variable __s from the information structure.

Definition at line 203 of file ie_rettype.hpp.

10.98.3.53 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 198 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

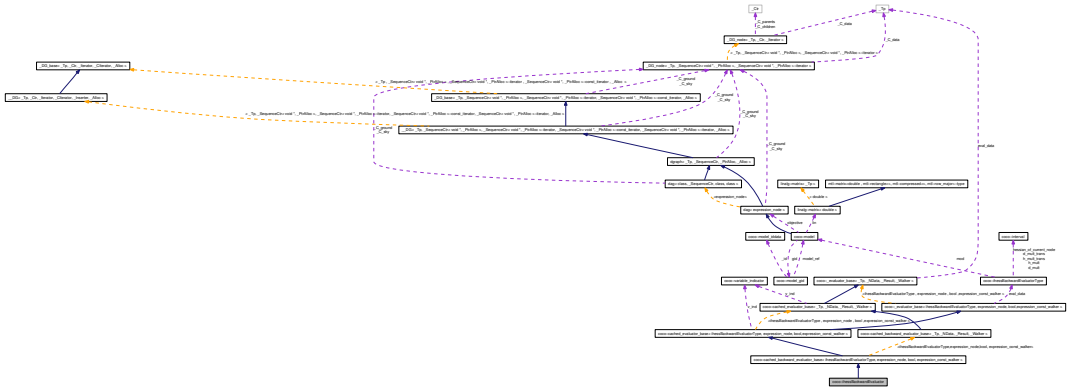
10.99 coco::ihessBackwardEvaluator Class Reference

`#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
- typedef ihessBackwardEvaluatorType data_type

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 void retrieve_from_cache (const node_data_type &__data)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- ihessBackwardEvaluatorType eval_data

10.99.1 Detailed Description

Definition at line 1173 of file ihess_evaluator.h.

10.99.2 Member Typedef Documentation

10.99.2.1 typedef _Base::const_walker coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 853 of file evaluator.h.

10.99.2.2 typedef ihessBackwardEvaluatorType coco::_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.99.2.3 typedef _Base::node_data_type coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 849 of file evaluator.h.

10.99.2.4 typedef _Base::return_value coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 851 of file evaluator.h.

10.99.3 Constructor & Destructor Documentation

10.99.3.1 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 1195 of file ihess_evaluator.h.

10.99.3.2 coco::ihessBackwardEvaluator::ihessBackwardEvaluator (const ihessBackwardEvaluator & *__he*) [inline]

Definition at line 1225 of file ihess_evaluator.h.

10.99.3.3 coco::ihessBackwardEvaluator::~ihessBackwardEvaluator () [inline]

Definition at line 1227 of file ihess_evaluator.h.

10.99.4 Member Function Documentation

10.99.4.1 int coco::ihessBackwardEvaluator::calculate (const expression_node & *__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.

Reimplemented from coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1267 of file ihess_evaluator.h.

10.99.4.2 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::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1506 of file ihess_evaluator.h.

10.99.4.3 void coco::ihessBackwardEvaluator::cleanup (const expression_node & *__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.

Reimplemented from coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1404 of file ihess_evaluator.h.

10.99.4.4 int coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 879 of file evaluator.h.

10.99.4.5 void coco::ihessBackwardEvaluator::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1256 of file ihess_evaluator.h.

10.99.4.6 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::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1182 of file ihess_evaluator.h.

10.99.4.7 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 1229 of file ihess_evaluator.h.

10.99.4.8 void coco::ihessBackwardEvaluator::newResult (std::vector< interval > * *__grad*, std::vector< interval > * *__hess*) [inline]

Definition at line 1239 of file ihess_evaluator.h.

10.99.4.9 void coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated into a call to cleanup.

Reimplemented from coco::_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 874 of file evaluator.h.

10.99.4.10 int coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Reimplemented from coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 862 of file evaluator.h.

10.99.4.11 virtual void coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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 929 of file evaluator.h.

10.99.4.12 void coco::ihessBackwardEvaluator::set_mult (interval *scal*) [inline]

Definition at line 1245 of file ihess_evaluator.h.

10.99.4.13 expression_const_walker coco::ihessBackwardEvaluator::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 1252 of file ihess_evaluator.h.

10.99.4.14 int coco::ihessBackwardEvaluator::update (const expression_node & *__data*, const bool & *__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.

Reimplemented from coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1452 of file ihess_evaluator.h.

10.99.4.15 int coco::ihessBackwardEvaluator::update (const bool & *__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.

Reimplemented from coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 1442 of file ihess_evaluator.h.

10.99.4.16 return_value coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter false.

Reimplemented from coco::_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 891 of file evaluator.h.

10.99.4.17 int coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 885 of file evaluator.h.

10.99.4.18 void coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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 900 of file evaluator.h.

10.99.4.19 return_value coco::cached_backward_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< ihessBackwardEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 896 of file evaluator.h.

10.99.5 Member Data Documentation

10.99.5.1 ihessBackwardEvaluatorType coco::_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.99.5.2 const variable_indicator* coco::cached_evaluator_base< ihessBackwardEvaluatorType , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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:

- ihess_evaluator.h

10.100 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

10.100.1 Detailed Description

Definition at line 1151 of file ihess_evaluator.h.

10.100.2 Member Data Documentation

10.100.2.1 bool coco::ihessBackwardEvaluatorType::calc_grad

Definition at line 1159 of file ihess_evaluator.h.

10.100.2.2 unsigned int coco::ihessBackwardEvaluatorType::child_n

Definition at line 1168 of file ihess_evaluator.h.

10.100.2.3 std::vector<std::vector<interval> >* coco::ihessBackwardEvaluatorType::d_cache

Definition at line 1155 of file ihess_evaluator.h.

10.100.2.4 std::vector<std::vector<interval> >* coco::ihessBackwardEvaluatorType::d_data

Definition at line 1153 of file ihess_evaluator.h.

10.100.2.5 interval coco::ihessBackwardEvaluatorType::d_mult

Definition at line 1161 of file ihess_evaluator.h.

10.100.2.6 interval coco::ihessBackwardEvaluatorType::d_mult_trans

Definition at line 1164 of file ihess_evaluator.h.

10.100.2.7 std::vector<interval>* coco::ihessBackwardEvaluatorType::grad_vec

Definition at line 1157 of file ihess_evaluator.h.

10.100.2.8 std::vector<std::vector<interval> >* coco::ihessBackwardEvaluatorType::h_cache

Definition at line 1156 of file ihess_evaluator.h.

10.100.2.9 std::vector<std::vector<interval> >* coco::ihessBackwardEvaluatorType::h_data

Definition at line 1154 of file ihess_evaluator.h.

10.100.2.10 interval coco::ihessBackwardEvaluatorType::h_mult

Definition at line 1165 of file ihess_evaluator.h.

10.100.2.11 interval coco::ihessBackwardEvaluatorType::h_mult_trans

Definition at line 1166 of file ihess_evaluator.h.

10.100.2.12 std::vector<interval>* coco::ihessBackwardEvaluatorType::hess_vec

Definition at line 1158 of file ihess_evaluator.h.

10.100.2.13 bool coco::ihessBackwardEvaluatorType::hess_zero

Definition at line 1163 of file ihess_evaluator.h.

10.100.2.14 interval coco::ihessBackwardEvaluatorType::hessian_of_current_node

Definition at line 1162 of file ihess_evaluator.h.

10.100.2.15 bool coco::ihessBackwardEvaluatorType::is_linear

Definition at line 1167 of file ihess_evaluator.h.

10.100.2.16 const model* coco::ihessBackwardEvaluatorType::mod

Definition at line 1160 of file ihess_evaluator.h.

The documentation for this struct was generated from the following file:

- ihess_evaluator.h

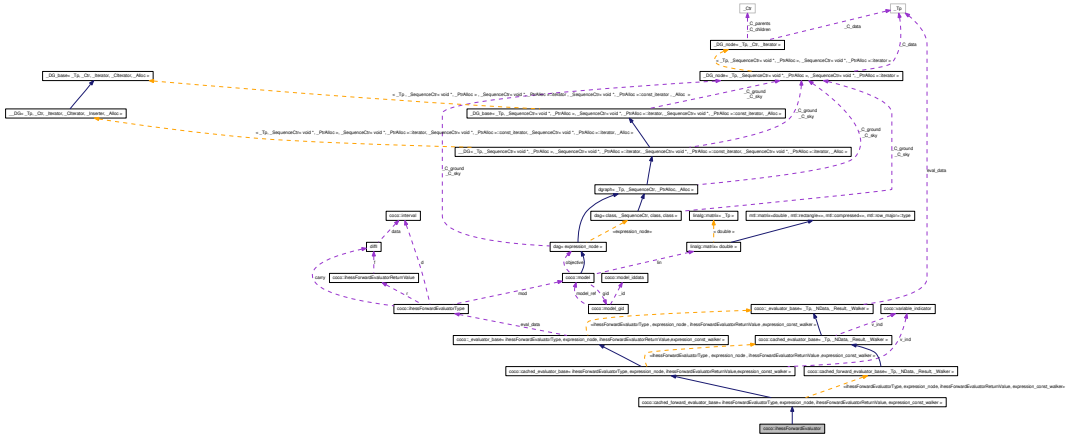
10.101 coco::ihessForwardEvaluator Class Reference

`#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
- typedef ihessForwardEvaluatorType data_type

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)
- 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 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 void cleanup (const node_data_type &__data)
- virtual int update (const return_value &__rval)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- ihessForwardEvaluatorType eval_data

10.101.1 Detailed Description

Definition at line 169 of file ihess_evaluator.h.

10.101.2 Member Typedef Documentation

10.101.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.101.2.2 typedef ihessForwardEvaluatorType coco::_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.101.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.101.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.101.3 Constructor & Destructor Documentation

10.101.3.1 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*) [inline]

Definition at line 251 of file ihess_evaluator.h.

10.101.3.2 coco::ihessForwardEvaluator::ihessForwardEvaluator (const ihessForwardEvaluator & *__x*) [inline]

Definition at line 291 of file ihess_evaluator.h.

10.101.3.3 coco::ihessForwardEvaluator::~ihessForwardEvaluator () [inline]

Definition at line 295 of file ihess_evaluator.h.

10.101.4 Member Function Documentation

10.101.4.1 void coco::ihessForwardEvaluator::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 416 of file ihess_evaluator.h.

10.101.4.2 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::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 1142 of file ihess_evaluator.h.

10.101.4.3 virtual void coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::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 804 of file evaluator.h.

10.101.4.4 int coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.101.4.5 int coco::ihessForwardEvaluator::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 342 of file ihess_evaluator.h.

10.101.4.6 void coco::ihessForwardEvaluator::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 333 of file ihess_evaluator.h.

10.101.4.7 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::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 178 of file ihess_evaluator.h.

10.101.4.8 void coco::ihessForwardEvaluator::newPoint (const std::vector< diffI > & *__x*, const variable_indicator & *__v*) [inline]

Definition at line 300 of file ihess_evaluator.h.

10.101.4.9 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 327 of file ihess_evaluator.h.

10.101.4.10 void coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.101.4.11 int coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.101.4.12 void coco::ihessForwardEvaluator::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 427 of file ihess_evaluator.h.

10.101.4.13 expression_const_walker coco::ihessForwardEvaluator::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 297 of file ihess_evaluator.h.

10.101.4.14 virtual int coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::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 826 of file evaluator.h.

10.101.4.15 int coco::ihessForwardEvaluator::update (const expression_node & *__data*, const ihessForwardEvaluatorReturnValue & *__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.

Reimplemented from coco::cached_forward_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue, expression_const_walker >.

Definition at line 444 of file ihess_evaluator.h.

10.101.4.16 int coco::ihessForwardEvaluator::update (const diffI & *__rval*) `[inline]`

Definition at line 434 of file ihess_evaluator.h.

10.101.4.17 return_value coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.101.4.18 int coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.101.4.19 void coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::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 771 of file evaluator.h.

10.101.4.20 return_value coco::cached_forward_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< ihessForwardEvaluatorType, expression_node, ihessForwardEvaluatorReturnValue,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.101.5 Member Data Documentation

10.101.5.1 ihessForwardEvaluatorType coco::_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.101.5.2 const variable_indicator* coco::cached_evaluator_base< ihessForwardEvaluatorType , expression_node , ihessForwardEvaluatorReturnValue , expression_const_walker >::v_ind [protected, inherited]

For caching a variable indicator is needed to check which nodes need not be re-evaluated.

Definition at line 427 of file evaluator.h.

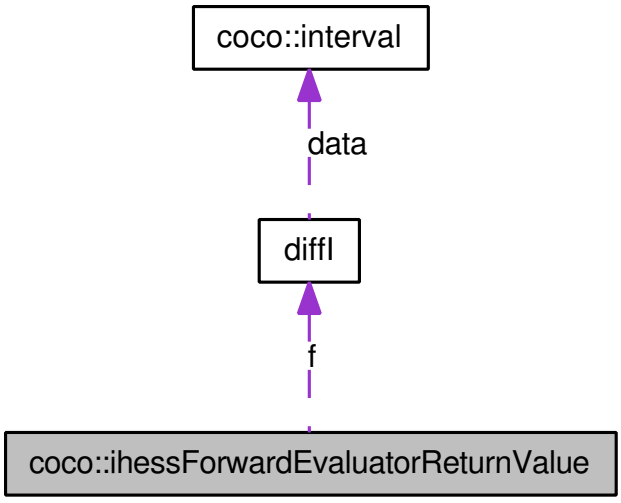
The documentation for this class was generated from the following file:

- ihess_evaluator.h

10.102 coco::ihessForwardEvaluatorReturnValue Struct Reference

`#include <ihess_evaluator.h>`

Collaboration diagram for coco::ihessForwardEvaluatorReturnValue:



Public Attributes

- diffI f
- unsigned int nn
- tristate in_chn

10.102.1 Detailed Description

Definition at line 144 of file ihess_evaluator.h.

10.102.2 Member Data Documentation

10.102.2.1 diffI coco::ihessForwardEvaluatorReturnValue::f

Definition at line 146 of file ihess_evaluator.h.

10.102.2.2 tristate coco::ihessForwardEvaluatorReturnValue::in_chn

Definition at line 149 of file ihess_evaluator.h.

10.102.2.3 unsigned int coco::ihessForwardEvaluatorReturnValue::nn

Definition at line 148 of file ihess_evaluator.h.

The documentation for this struct was generated from the following file:

- ihess_evaluator.h

10.103 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
- ihessForwardEvaluatorReturnValue r
- diffI carry
- const model * mod
- void * p
- interval d
- unsigned int n
- unsigned int info

10.103.1 Detailed Description

Definition at line 152 of file ihess_evaluator.h.

10.103.2 Member Data Documentation

10.103.2.1 std::vector<bool>* coco::ihessForwardEvaluatorType::b

Definition at line 160 of file ihess_evaluator.h.

10.103.2.2 diffI coco::ihessForwardEvaluatorType::carry

Definition at line 162 of file ihess_evaluator.h.

10.103.2.3 interval coco::ihessForwardEvaluatorType::d

Definition at line 165 of file ihess_evaluator.h.

10.103.2.4 std::vector<std::vector<interval> >* coco::ihessForwardEvaluatorType::d_data

Definition at line 158 of file ihess_evaluator.h.

10.103.2.5 std::vector<diffI>* coco::ihessForwardEvaluatorType::f_cache

Definition at line 156 of file ihess_evaluator.h.

10.103.2.6 std::vector<std::vector<interval> >* coco::ihessForwardEvaluatorType::h_data

Definition at line 157 of file ihess_evaluator.h.

10.103.2.7 unsigned int coco::ihessForwardEvaluatorType::info

Definition at line 166 of file ihess_evaluator.h.

10.103.2.8 const model* coco::ihessForwardEvaluatorType::mod

Definition at line 163 of file ihess_evaluator.h.

10.103.2.9 unsigned int coco::ihessForwardEvaluatorType::n

Definition at line 166 of file ihess_evaluator.h.

10.103.2.10 void* coco::ihessForwardEvaluatorType::p

Definition at line 164 of file ihess_evaluator.h.

10.103.2.11 ihessForwardEvaluatorReturnValue coco::ihessForwardEvaluatorType::r

Definition at line 161 of file ihess_evaluator.h.

10.103.2.12 const std::vector<interval>* coco::ihessForwardEvaluatorType::range

the ranges of all nodes

Definition at line 155 of file ihess_evaluator.h.

10.103.2.13 std::vector<unsigned int>* coco::ihessForwardEvaluatorType::t

Definition at line 159 of file ihess_evaluator.h.

10.103.2.14 const std::vector<diffI>* coco::ihessForwardEvaluatorType::x

Definition at line 154 of file ihess_evaluator.h.

The documentation for this struct was generated from the following file:

- ihess_evaluator.h

10.104 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
- typedef ihessPreparationEvaluatorType data_type

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 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 const_walker short_cut_to (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- ihessPreparationEvaluatorType eval_data

10.104.1 Detailed Description

Definition at line 73 of file ihess_evaluator.h.

10.104.2 Member Typedef Documentation

10.104.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.104.2.2 typedef ihessPreparationEvaluatorType coco::_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.104.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.104.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.104.3 Constructor & Destructor Documentation

10.104.3.1 coco::ihessPreparationEvaluator::ihessPreparationEvaluator (std::vector< std::vector< interval > > & *__d*, std::vector< std::vector< interval > > & *__h*, unsigned int *_num_of_nodes*) [inline]

Definition at line 82 of file ihess_evaluator.h.

10.104.3.2 coco::ihessPreparationEvaluator::ihessPreparationEvaluator (const ihessPreparationEvaluator & *__x*) [inline]

Definition at line 101 of file ihess_evaluator.h.

10.104.3.3 coco::ihessPreparationEvaluator::~ihessPreparationEvaluator () [inline]

Definition at line 105 of file ihess_evaluator.h.

10.104.4 Member Function Documentation

10.104.4.1 void coco::ihessPreparationEvaluator::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 126 of file ihess_evaluator.h.

10.104.4.2 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::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 138 of file ihess_evaluator.h.

10.104.4.3 virtual void coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 804 of file evaluator.h.

10.104.4.4 int coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.104.4.5 int coco::ihessPreparationEvaluator::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 116 of file ihess_evaluator.h.

10.104.4.6 void coco::ihessPreparationEvaluator::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 107 of file ihess_evaluator.h.

10.104.4.7 bool coco::ihessPreparationEvaluator::is_cached (const expression_node & *__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 from coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 109 of file ihess_evaluator.h.

10.104.4.8 void coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.104.4.9 int coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.104.4.10 void coco::ihessPreparationEvaluator::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool, expression_const_walker >.

Definition at line 114 of file ihess_evaluator.h.

10.104.4.11 virtual const_walker coco::cached_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 450 of file evaluator.h.

10.104.4.12 virtual int coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 826 of file evaluator.h.

10.104.4.13 virtual int coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 814 of file evaluator.h.

10.104.4.14 int coco::ihessPreparationEvaluator::update (const expression_node & *__data*, bool *__rval*) [inline]

Definition at line 130 of file ihess_evaluator.h.

10.104.4.15 int coco::ihessPreparationEvaluator::update (bool *__rval*) [inline]

Definition at line 128 of file ihess_evaluator.h.

10.104.4.16 return_value coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.104.4.17 int coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.104.4.18 void coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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 771 of file evaluator.h.

10.104.4.19 return_value coco::cached_forward_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< ihessPreparationEvaluatorType, expression_node, bool,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.104.5 Member Data Documentation

10.104.5.1 ihessPreparationEvaluatorType coco::_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.104.5.2 const variable_indicator* coco::cached_evaluator_base< ihessPreparationEvaluatorType , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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:

- ihess_evaluator.h

10.105 coco::ihessPreparationEvaluatorType Struct Reference

`#include <ihess_evaluator.h>`

Public Attributes

- std::vector< std::vector< interval > > * d
- std::vector< std::vector< interval > > * h

10.105.1 Detailed Description

Definition at line 67 of file ihess_evaluator.h.

10.105.2 Member Data Documentation

10.105.2.1 std::vector<std::vector<interval> >* coco::ihessPreparationEvaluatorType::d

Definition at line 69 of file ihess_evaluator.h.

10.105.2.2 std::vector<std::vector<interval> >* coco::ihessPreparationEvaluatorType::h

Definition at line 70 of file ihess_evaluator.h.

The documentation for this struct was generated from the following file:

- ihess_evaluator.h

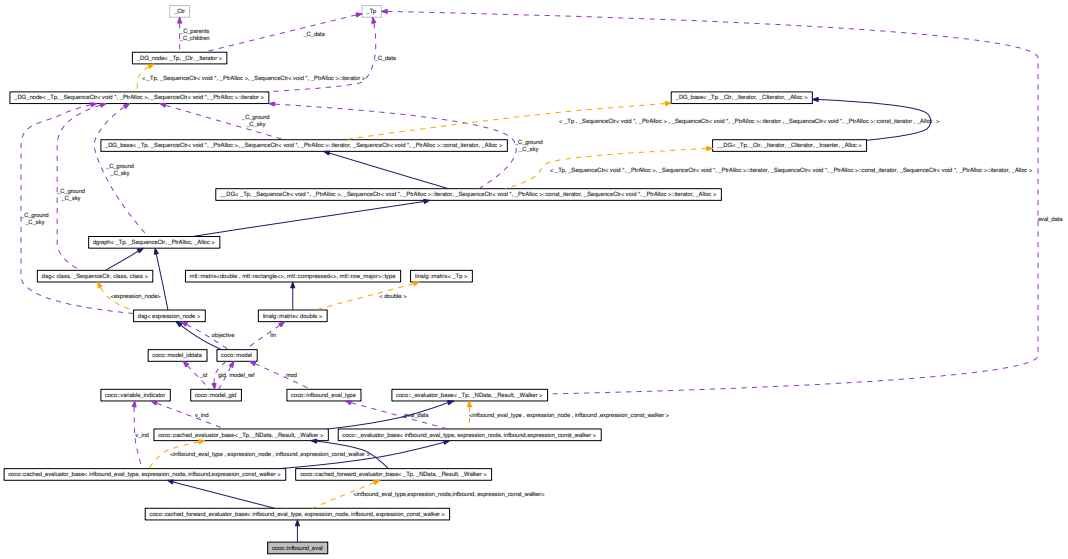
10.106 coco::infbound_eval Class Reference

`#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
- typedef infbound_eval_type data_type

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 void cleanup (const node_data_type &__data)

Protected Member Functions

- bool is_cached (const node_data_type &__data)

Protected Attributes

- const variable_indicator * v_ind
- infbound_eval_type eval_data

10.106.1 Detailed Description

Definition at line 60 of file infb_evaluator.h.

10.106.2 Member Typedef Documentation

10.106.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.106.2.2 typedef infbound_eval_type coco::_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.106.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.106.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.106.3 Constructor & Destructor Documentation

10.106.3.1 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 115 of file infb_evaluator.h.

10.106.3.2 coco::infbound_eval::infbound_eval (const infbound_eval & *__v*) [inline]

Definition at line 127 of file infb_evaluator.h.

10.106.3.3 coco::infbound_eval::~infbound_eval () [inline]

Definition at line 129 of file infb_evaluator.h.

10.106.4 Member Function Documentation

10.106.4.1 void coco::infbound_eval::calculate (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 192 of file infb_evaluator.h.

10.106.4.2 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::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 711 of file infb_evaluator.h.

10.106.4.3 virtual void coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::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 804 of file evaluator.h.

10.106.4.4 int coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.106.4.5 int coco::infbound_eval::initialize (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 136 of file infb_evaluator.h.

10.106.4.6 void coco::infbound_eval::initialize () [inline, virtual]

This method is called at a virtual node before any children are visited.

Reimplemented from coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 134 of file infb_evaluator.h.

10.106.4.7 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::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 69 of file infb_evaluator.h.

10.106.4.8 void coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.106.4.9 int coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.106.4.10 void coco::infbound_eval::retrieve_from_cache (const expression_node & *__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.

Reimplemented from coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 204 of file infb_evaluator.h.

10.106.4.11 expression_const_walker coco::infbound_eval::short_cut_to (const expression_node & *__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.

Reimplemented from coco::cached_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 131 of file infb_evaluator.h.

10.106.4.12 int coco::infbound_eval::update (const expression_node & *__data*, const infbound & *__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.

Reimplemented from coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 229 of file infb_evaluator.h.

10.106.4.13 int coco::infbound_eval::update (const infbound & *__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.

Reimplemented from coco::cached_forward_evaluator_base< infbound_eval_type, expression_node, infbound, expression_const_walker >.

Definition at line 223 of file infb_evaluator.h.

10.106.4.14 return_value coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.106.4.15 int coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.106.4.16 void coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::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 771 of file evaluator.h.

10.106.4.17 return_value coco::cached_forward_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< infbound_eval_type, expression_node, infbound,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.106.5 Member Data Documentation

10.106.5.1 infbound_eval_type coco::_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.106.5.2 const variable_indicator* coco::cached_evaluator_base< infbound_eval_type , expression_node , infbound , expression_const_walker >::v_ind [protected, inherited]

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:

- infb_evaluator.h

10.107 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

10.107.1 Detailed Description

Definition at line 48 of file infb_evaluator.h.

10.107.2 Member Data Documentation

10.107.2.1 std::vector<infbound>* coco::infbound_eval_type::cache

Definition at line 51 of file infb_evaluator.h.

10.107.2.2 infbound coco::infbound_eval_type::d

Definition at line 54 of file infb_evaluator.h.

10.107.2.3 int coco::infbound_eval_type::info

Definition at line 55 of file infb_evaluator.h.

10.107.2.4 const model* coco::infbound_eval_type::mod

Definition at line 52 of file infb_evaluator.h.

10.107.2.5 unsigned int coco::infbound_eval_type::n

Definition at line 57 of file infb_evaluator.h.

10.107.2.6 void* coco::infbound_eval_type::p

Definition at line 53 of file infb_evaluator.h.

10.107.2.7 infbound coco::infbound_eval_type::r

Definition at line 56 of file infb_evaluator.h.

10.107.2.8 const std::vector<infbound>* coco::infbound_eval_type::x

Definition at line 50 of file infb_evaluator.h.

The documentation for this struct was generated from the following file:

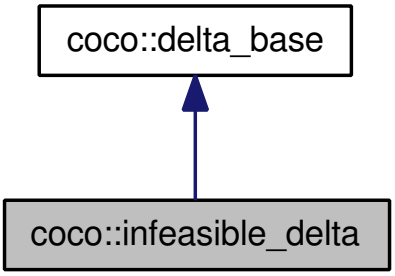
- infb_evaluator.h

10.108 coco::infeasible_delta Class Reference

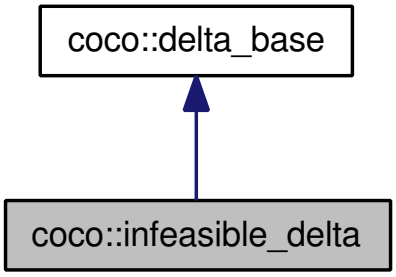
The infeasible delta class for marking a model as infeasible.

`#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) 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)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Protected Attributes

- std::string _action

10.108.1 Detailed Description

This class is used to mark a model as infeasible.

Definition at line 81 of file infeasible_delta.h.

10.108.2 Constructor & Destructor Documentation

10.108.2.1 coco::infeasible_delta::infeasible_delta () [inline]

Standard Constructor

Definition at line 85 of file infeasible_delta.h.

10.108.2.2 coco::infeasible_delta::infeasible_delta (const infeasible_delta & *__d*) [inline]

Standard Copy Constructor

Definition at line 87 of file infeasible_delta.h.

10.108.2.3 coco::infeasible_delta::~infeasible_delta () [inline]

Standard Destructor

Definition at line 95 of file infeasible_delta.h.

10.108.3 Member Function Documentation

10.108.3.1 bool coco::infeasible_delta::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_did*) const [inline, virtual]

Apply the delta with delta_id _d to work node _x, hereby marking the model as infeasible.

Reimplemented from coco::delta_base.

Definition at line 104 of file infeasible_delta.h.

10.108.3.2 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*) 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.

10.108.3.3 virtual void 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.

Reimplemented in coco::table_delta.

Definition at line 189 of file api_deltabase.h.

10.108.3.4 void coco::infeasible_delta::destroy_copy (delta_base * *__d*) const [inline]

Clone Destructor

Definition at line 100 of file infeasible_delta.h.

10.108.3.5 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.108.3.6 delta 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 api_deltabase.h.

10.108.3.7 infeasible_delta* coco::infeasible_delta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::delta_base.

Definition at line 98 of file infeasible_delta.h.

10.108.3.8 bool coco::infeasible_delta::operator!= (const infeasible_delta & *_c*) const [inline]

Definition at line 121 of file infeasible_delta.h.

10.108.3.9 bool coco::infeasible_delta::operator!= (const delta_base & *_c*) const [inline]

Definition at line 115 of file infeasible_delta.h.

10.108.3.10 bool coco::infeasible_delta::operator== (const infeasible_delta & _c) const [inline]

Comparison operators

Definition at line 120 of file infeasible_delta.h.

10.108.3.11 bool coco::infeasible_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Definition at line 112 of file infeasible_delta.h.

10.108.3.12 virtual void 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 api_deltabase.h.

10.108.4 Member Data Documentation

10.108.4.1 std::string coco::delta_base::_action [protected, inherited]

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:

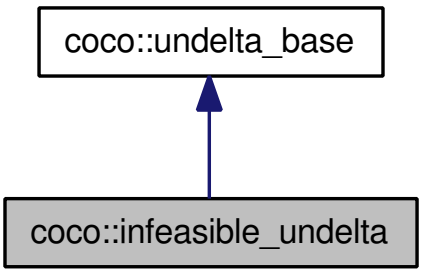
- infeasible_delta.h

10.109 coco::infeasible_undelta Class Reference

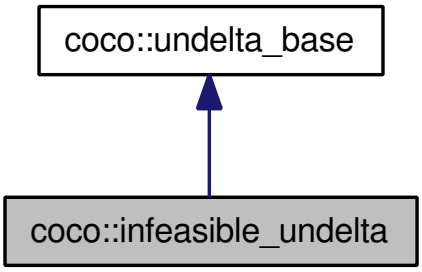
The infeasible undelta class for undoing changes to the feasibility of a model.

`#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 ()
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

10.109.1 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.

Definition at line 41 of file infeasible_delta.h.

10.109.2 Constructor & Destructor Documentation

10.109.2.1 coco::infeasible_undelta::infeasible_undelta (bool *_oi* = `false`) `[inline]`

Constructor initializing old_infeasible with _oi

Definition at line 50 of file infeasible_delta.h.

10.109.2.2 coco::infeasible_undelta::infeasible_undelta (const infeasible_undelta & *__d*) [inline]

Standard Copy Constructor

Definition at line 52 of file infeasible_delta.h.

10.109.2.3 coco::infeasible_undelta::~infeasible_undelta () [inline]

Standard Destructor

Definition at line 61 of file infeasible_delta.h.

10.109.3 Member Function Documentation

10.109.3.1 void coco::infeasible_undelta::destroy_copy (undelta_base * *__d*) const [inline]

Clone Destructor

Definition at line 66 of file infeasible_delta.h.

10.109.3.2 undelta coco::undelta_base::make_undelta () [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 283 of file api_deltabase.h.

10.109.3.3 infeasible_undelta* coco::infeasible_undelta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::undelta_base.

Definition at line 64 of file infeasible_delta.h.

10.109.3.4 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::undelta_base.

Definition at line 69 of file infeasible_delta.h.

10.109.3.5 bool 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:

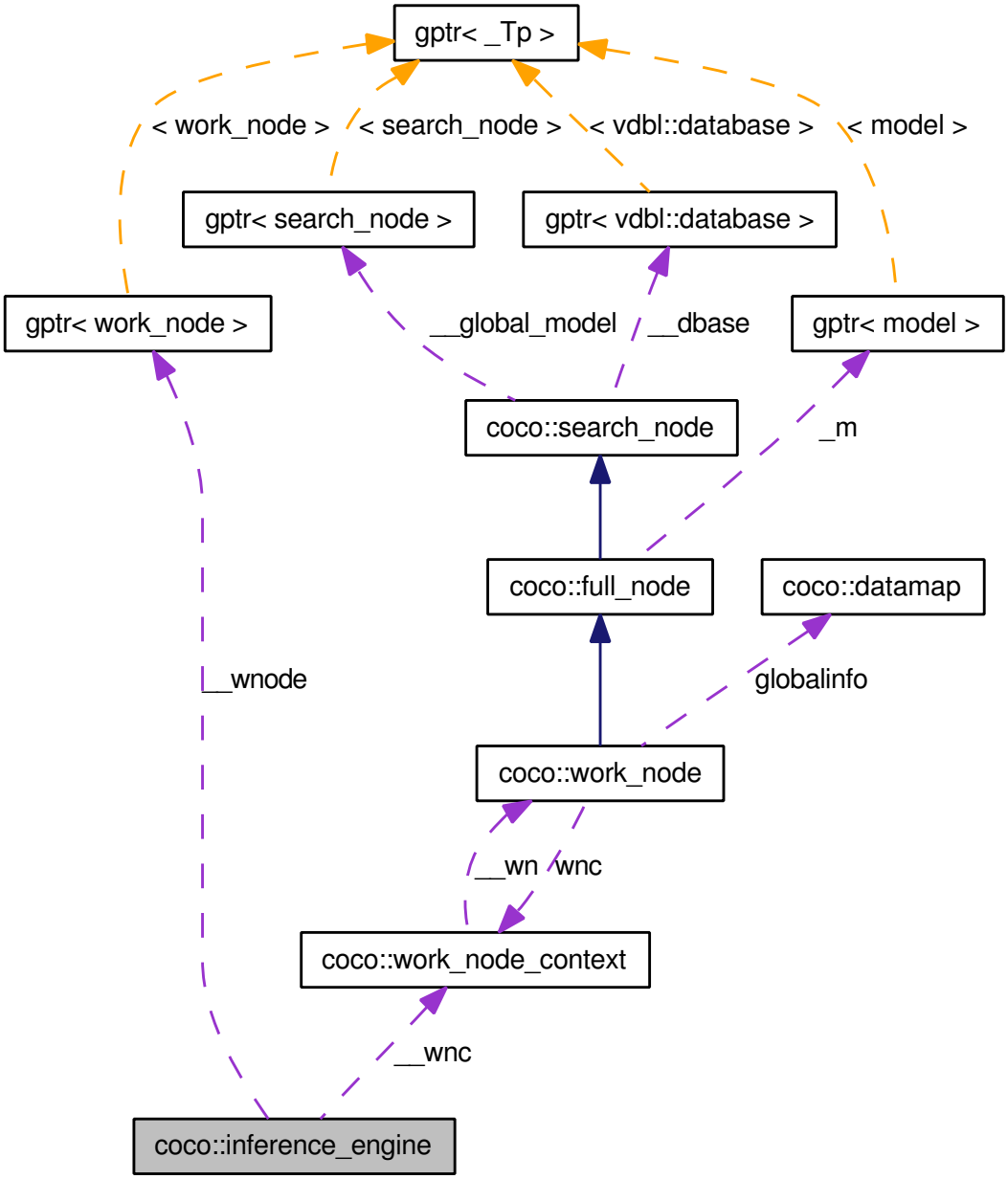
- infeasible_delta.h

10.110 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)
- 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
- std::vector< delta_id > _old_deltas
- std::vector< delta_id > _new_deltas

10.110.1 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,...

Definition at line 76 of file inference_engine.h.

10.110.2 Constructor & Destructor Documentation

10.110.2.1 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 102 of file inference_engine.h.

10.110.2.2 virtual coco::inference_engine::~inference_engine () [inline, virtual]

Standard Destructor

Definition at line 110 of file inference_engine.h.

10.110.3 Member Function Documentation

10.110.3.1 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 149 of file inference_engine.h.

10.110.3.2 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 126 of file inference_engine.h.

10.110.3.3 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 130 of file inference_engine.h.

10.110.3.4 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 142 of file inference_engine.h.

10.110.3.5 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 138 of file inference_engine.h.

10.110.3.6 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 146 of file inference_engine.h.

10.110.3.7 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.

10.110.3.8 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 113 of file inference_engine.h.

10.110.4 Member Data Documentation

10.110.4.1 std::string coco::inference_engine::__name [protected]

This is the identifier string for an inference engine.

Definition at line 80 of file inference_engine.h.

10.110.4.2 const vdbl::viewdbase& coco::inference_engine::__vdb [protected]

This is a view to the search database.

Definition at line 87 of file inference_engine.h.

10.110.4.3 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 85 of file inference_engine.h.

10.110.4.4 const gptr<work_node>* coco::inference_engine::__wnode [protected]

This variable contains a global pointer to the work node.

Definition at line 82 of file inference_engine.h.

10.110.4.5 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 93 of file inference_engine.h.

10.110.4.6 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 90 of file inference_engine.h.

The documentation for this class was generated from the following files:

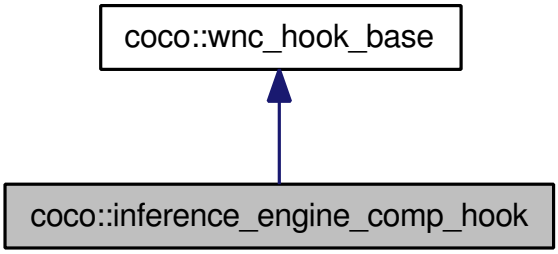
- inference_engine.h
- inference_engine.cc

10.111 coco::inference_engine_comp_hook Class Reference

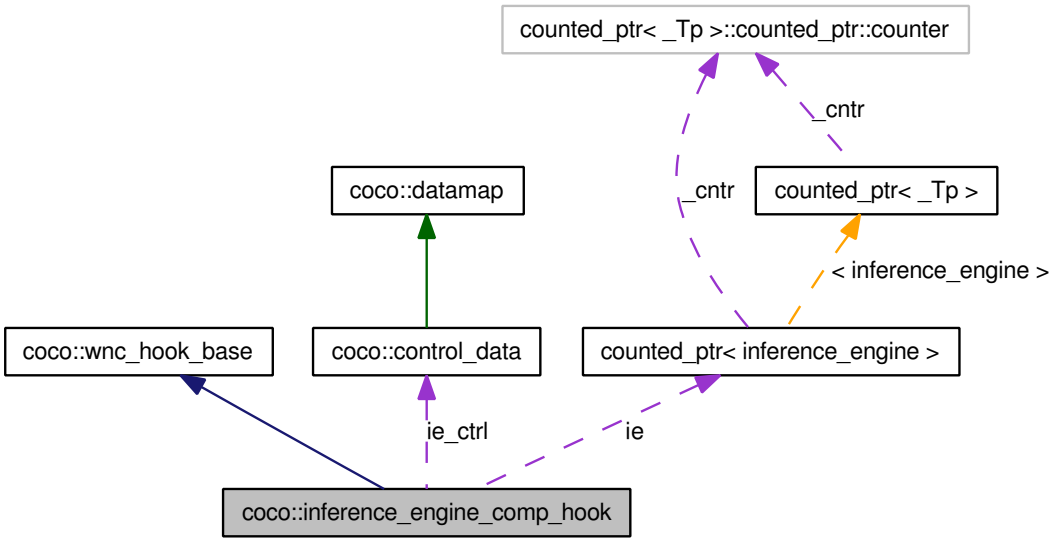
The inference engine meta computation hook (work node computation hook).

`#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

10.111.1 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.

Definition at line 48 of file infeng_hook.h.

10.111.2 Constructor & Destructor Documentation

10.111.2.1 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.

10.111.2.2 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.

10.111.2.3 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.

10.111.2.4 virtual coco::inference_engine_comp_hook::~inference_engine_comp_hook () [inline, virtual]

Standard Destructor

Definition at line 88 of file infeng_hook.h.

10.111.3 Member Function Documentation

10.111.3.1 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.

10.111.3.2 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.

10.111.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *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 238 of file comp_hook.h.

10.111.3.4 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 291 of file infeng_hook.h.

10.111.3.5 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 191 of file infeng_hook.h.

10.111.3.6 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.

10.111.3.7 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.

10.111.3.8 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.

10.111.3.9 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.

10.111.3.10 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:

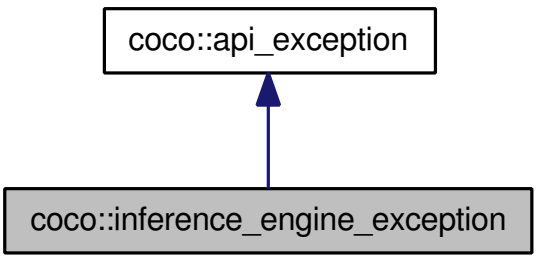
- infeng_hook.h

10.112 coco::inference_engine_exception Class Reference

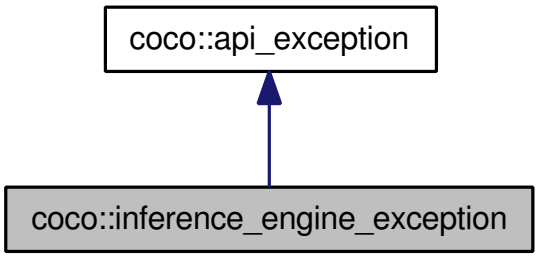
Inference engine exception class.

`#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 api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.112.1 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.

Definition at line 55 of file inference_engine.h.

10.112.2 Constructor & Destructor Documentation

10.112.2.1 coco::inference_engine_exception::inference_engine_exception (const std::string & *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 59 of file inference_engine.h.

10.112.2.2 coco::inference_engine_exception::inference_engine_exception (const char * *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 62 of file inference_engine.h.

10.112.2.3 virtual coco::inference_engine_exception::~inference_engine_exception () throw () [inline, virtual]

Standard Destructor

Definition at line 66 of file inference_engine.h.

10.112.3 Member Function Documentation

10.112.3.1 virtual std::string coco::api_exception::message () const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file api_exception.h.

10.112.3.2 virtual api_exception_type coco::api_exception::type () const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file api_exception.h.

10.112.3.3 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.

10.112.3.4 virtual char const* coco::api_exception::what () const throw () [inline, virtual, inherited]

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:

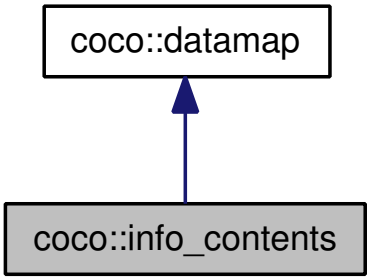
- inference_engine.h

10.113 coco::info_contents Class Reference

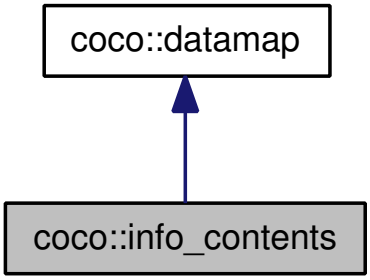
The class for returning additional information from inference modules.

`#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, 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 linalg::matrix< double > *&__v) const
- bool retrieve (const std::string &__s, const linalg::matrix< int > *&__v) const
- bool retrieve (const std::string &__s, const linalg::matrix< interval > *&__v) const
- bool retrieve (const std::string &__s, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< int > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< interval > *&__v) const
- bool retrieve (const char *__s, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve (const std::string &__s, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve (const char *__s, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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 &__is) 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 linalg::matrix< double > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< int > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< interval > *&__v) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< int > *&__v) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< interval > *&__v) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< std::string > *&__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve_i (const std::string &__s, int i, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__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, 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 linalg::matrix< double > *&__v, const linalg::matrix< double > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< int > *&__v, const linalg::matrix< int > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< interval > *&__v, const linalg::matrix< interval > *__def) const
- bool retrieve_i (const char *__s, int i, const linalg::matrix< std::string > *&__v, const linalg::matrix< std::string > *__def) const

10.113.1 Detailed Description

This class is used for returning additional information from inference modules. It is basically a datamap.

Definition at line 42 of file info_contents.h.

10.113.2 Constructor & Destructor Documentation

10.113.2.1 coco::info_contents::info_contents () [inline]

Standard Constructor

Definition at line 50 of file info_contents.h.

10.113.2.2 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.

10.113.2.3 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.

10.113.2.4 coco::info_contents::info_contents (const info_contents & __c) [inline]

Standard Copy Constructor

Definition at line 58 of file info_contents.h.

10.113.2.5 virtual coco::info_contents::~info_contents () [inline, virtual]

Standard Destructor

Definition at line 60 of file info_contents.h.

10.113.3 Member Function Documentation

10.113.3.1 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.

10.113.3.2 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.

10.113.3.3 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.

10.113.3.4 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.

10.113.3.5 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 99 of file datamap.cc.

10.113.3.6 info_contents& coco::info_contents::operator= (const info_contents & __c) [inline]

Standard Assignment Operator

Definition at line 63 of file info_contents.h.

10.113.3.7 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.

10.113.3.8 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.

10.113.3.9 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.

10.113.3.10 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.

10.113.3.11 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< std::string > *& __v, const linalg::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 658 of file datamap.hpp.

10.113.3.12 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< interval > *& __v, const linalg::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 647 of file datamap.hpp.

10.113.3.13 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< int > *& __v, const linalg::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 636 of file datamap.hpp.

10.113.3.14 bool coco::datamap::retrieve (const char * __s, const linalg::matrix< double > *& __v, const linalg::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 625 of file datamap.hpp.

10.113.3.15 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 615 of file datamap.hpp.

10.113.3.16 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 605 of file datamap.hpp.

10.113.3.17 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 595 of file datamap.hpp.

10.113.3.18 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 575 of file datamap.hpp.

10.113.3.19 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 585 of file datamap.hpp.

10.113.3.20 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 565 of file datamap.hpp.

10.113.3.21 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 555 of file datamap.hpp.

10.113.3.22 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 545 of file datamap.hpp.

10.113.3.23 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 535 of file datamap.hpp.

10.113.3.24 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 525 of file datamap.hpp.

10.113.3.25 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 515 of file datamap.hpp.

10.113.3.26 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 505 of file datamap.hpp.

10.113.3.27 bool coco::datamap::retrieve (const std::string & *__s*, const linalg::matrix< std::string > *& *__v*, const linalg::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 489 of file datamap.hpp.

10.113.3.28 bool coco::datamap::retrieve (const std::string & *__s*, const linalg::matrix< interval > *& *__v*, const linalg::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 467 of file datamap.hpp.

10.113.3.29 bool coco::datamap::retrieve (const std::string & *__s*, const linalg::matrix< int > *& *__v*, const linalg::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 445 of file datamap.hpp.

10.113.3.30 bool coco::datamap::retrieve (const std::string & *__s*, const linalg::matrix< double > *& *__v*, const linalg::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 423 of file datamap.hpp.

10.113.3.31 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 401 of file datamap.hpp.

10.113.3.32 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 379 of file datamap.hpp.

10.113.3.33 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 357 of file datamap.hpp.

10.113.3.34 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 335 of file datamap.hpp.

10.113.3.35 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 313 of file datamap.hpp.

10.113.3.36 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 291 of file datamap.hpp.

10.113.3.37 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.

10.113.3.38 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.

10.113.3.39 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.

10.113.3.40 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.

10.113.3.41 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.

10.113.3.42 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.

10.113.3.43 bool coco::datamap::retrieve (const char * *__s*, const linalg::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 653 of file datamap.hpp.

10.113.3.44 bool coco::datamap::retrieve (const char * *__s*, const linalg::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 642 of file datamap.hpp.

10.113.3.45 bool coco::datamap::retrieve (const char * __s, const linalg::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.

10.113.3.46 bool coco::datamap::retrieve (const char * __s, const linalg::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 620 of file datamap.hpp.

10.113.3.47 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 610 of file datamap.hpp.

10.113.3.48 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 600 of file datamap.hpp.

10.113.3.49 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 590 of file datamap.hpp.

10.113.3.50 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 570 of file datamap.hpp.

10.113.3.51 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 580 of file datamap.hpp.

10.113.3.52 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 560 of file datamap.hpp.

10.113.3.53 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 550 of file datamap.hpp.

10.113.3.54 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 540 of file datamap.hpp.

10.113.3.55 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 530 of file datamap.hpp.

10.113.3.56 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 520 of file datamap.hpp.

10.113.3.57 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 510 of file datamap.hpp.

10.113.3.58 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 500 of file datamap.hpp.

10.113.3.59 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 477 of file datamap.hpp.

10.113.3.60 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 455 of file datamap.hpp.

10.113.3.61 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 433 of file datamap.hpp.

10.113.3.62 bool coco::datamap::retrieve (const std::string & __s, const linalg::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 411 of file datamap.hpp.

10.113.3.63 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 389 of file datamap.hpp.

10.113.3.64 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 367 of file datamap.hpp.

10.113.3.65 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 345 of file datamap.hpp.

10.113.3.66 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 323 of file datamap.hpp.

10.113.3.67 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 301 of file datamap.hpp.

10.113.3.68 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 279 of file datamap.hpp.

10.113.3.69 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.

10.113.3.70 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.

10.113.3.71 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.

10.113.3.72 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.

10.113.3.73 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.

10.113.3.74 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.

10.113.3.75 bool coco::datamap::retrieve_i (const char * __s, int *i*, const linalg::matrix< std::string > *& __v, const linalg::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 1143 of file datamap.hpp.

10.113.3.76 bool coco::datamap::retrieve_i (const char * __s, int *i*, const linalg::matrix< interval > *& __v, const linalg::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 1132 of file datamap.hpp.

10.113.3.77 bool coco::datamap::retrieve_i (const char * __s, int *i*, const linalg::matrix< int > *& __v, const linalg::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 1121 of file datamap.hpp.

10.113.3.78 bool coco::datamap::retrieve_i (const char * *__s*, int *i*, const linalg::matrix< double > *&* *__v*, const linalg::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 1110 of file datamap.hpp.

10.113.3.79 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 1100 of file datamap.hpp.

10.113.3.80 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 1090 of file datamap.hpp.

10.113.3.81 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 1080 of file datamap.hpp.

10.113.3.82 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 1070 of file datamap.hpp.

10.113.3.83 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 1060 of file datamap.hpp.

10.113.3.84 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 1050 of file datamap.hpp.

10.113.3.85 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 1040 of file datamap.hpp.

10.113.3.86 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 1030 of file datamap.hpp.

10.113.3.87 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 1020 of file datamap.hpp.

10.113.3.88 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 1010 of file datamap.hpp.

10.113.3.89 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 1000 of file datamap.hpp.

10.113.3.90 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 990 of file datamap.hpp.

10.113.3.91 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const linalg::matrix< std::string > *& *__v*, const linalg::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 975 of file datamap.hpp.

10.113.3.92 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const linalg::matrix< interval > *& *__v*, const linalg::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 955 of file datamap.hpp.

10.113.3.93 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const linalg::matrix< int > *& *__v*, const linalg::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 935 of file datamap.hpp.

10.113.3.94 bool coco::datamap::retrieve_i (const std::string & *__s*, int *i*, const linalg::matrix< double > *& *__v*, const linalg::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 915 of file datamap.hpp.

10.113.3.95 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 895 of file datamap.hpp.

10.113.3.96 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 875 of file datamap.hpp.

10.113.3.97 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 855 of file datamap.hpp.

10.113.3.98 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 835 of file datamap.hpp.

10.113.3.99 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 815 of file datamap.hpp.

10.113.3.100 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 795 of file datamap.hpp.

10.113.3.101 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 775 of file datamap.hpp.

10.113.3.102 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 755 of file datamap.hpp.

10.113.3.103 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 735 of file datamap.hpp.

10.113.3.104 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 715 of file datamap.hpp.

10.113.3.105 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 695 of file datamap.hpp.

10.113.3.106 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 675 of file datamap.hpp.

10.113.3.107 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1138 of file datamap.hpp.

10.113.3.108 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1127 of file datamap.hpp.

10.113.3.109 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1116 of file datamap.hpp.

10.113.3.110 bool coco::datamap::retrieve_i (const char * __s, int i, const linalg::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 1105 of file datamap.hpp.

10.113.3.111 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 1095 of file datamap.hpp.

10.113.3.112 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 1085 of file datamap.hpp.

10.113.3.113 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 1075 of file datamap.hpp.

10.113.3.114 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 1065 of file datamap.hpp.

10.113.3.115 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 1055 of file datamap.hpp.

10.113.3.116 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 1045 of file datamap.hpp.

10.113.3.117 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 1035 of file datamap.hpp.

10.113.3.118 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 1025 of file datamap.hpp.

10.113.3.119 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 1015 of file datamap.hpp.

10.113.3.120 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 1005 of file datamap.hpp.

10.113.3.121 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 995 of file datamap.hpp.

10.113.3.122 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 985 of file datamap.hpp.

10.113.3.123 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 964 of file datamap.hpp.

10.113.3.124 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 944 of file datamap.hpp.

10.113.3.125 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 924 of file datamap.hpp.

10.113.3.126 bool coco::datamap::retrieve_i (const std::string & __s, int i, const linalg::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 904 of file datamap.hpp.

10.113.3.127 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 884 of file datamap.hpp.

10.113.3.128 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 864 of file datamap.hpp.

10.113.3.129 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 844 of file datamap.hpp.

10.113.3.130 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 824 of file datamap.hpp.

10.113.3.131 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 804 of file datamap.hpp.

10.113.3.132 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 784 of file datamap.hpp.

10.113.3.133 bool coco::datamap::retrieve_i (const std::string & *__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 764 of file datamap.hpp.

10.113.3.134 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 744 of file datamap.hpp.

10.113.3.135 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 724 of file datamap.hpp.

10.113.3.136 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 704 of file datamap.hpp.

10.113.3.137 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 684 of file datamap.hpp.

10.113.3.138 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 664 of file datamap.hpp.

10.113.3.139 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.

10.113.3.140 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.

10.113.3.141 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.

10.113.3.142 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.

10.113.3.143 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.

10.113.3.144 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.

10.113.3.145 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.

10.113.3.146 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 34 of file datamap.cc.

10.113.3.147 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.

10.113.3.148 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.

The documentation for this class was generated from the following file:

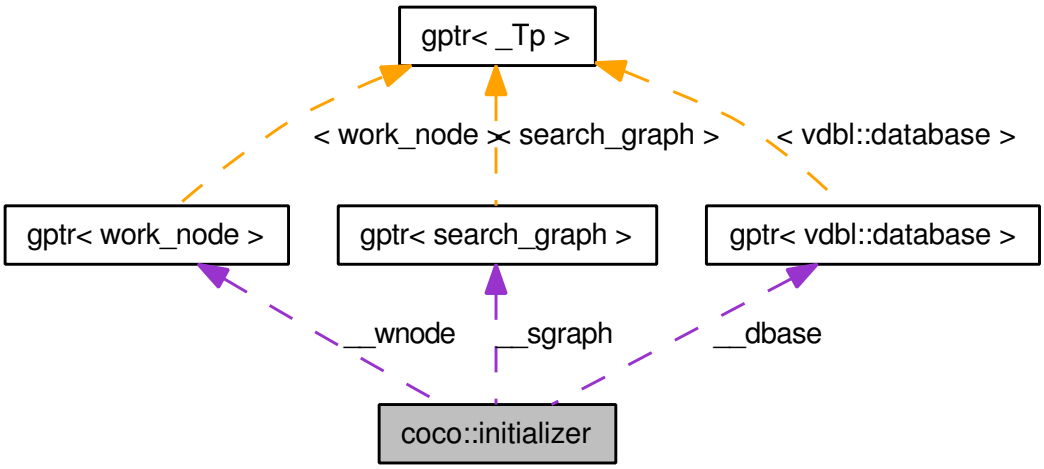
- info_contents.h

10.114 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

10.114.1 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.

Definition at line 74 of file initializer.h.

10.114.2 Constructor & Destructor Documentation

10.114.2.1 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.

10.114.2.2 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.

10.114.2.3 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.

10.114.2.4 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.

10.114.2.5 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.

10.114.2.6 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.

10.114.2.7 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.

10.114.2.8 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.

10.114.2.9 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.

10.114.2.10 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.

10.114.2.11 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.

10.114.2.12 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.

10.114.2.13 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.

10.114.2.14 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.

10.114.2.15 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.

10.114.2.16 virtual coco::initializer::~initializer () [inline, virtual]

Standard Destructor

Definition at line 244 of file initializer.h.

10.114.3 Member Function Documentation

10.114.3.1 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.

10.114.3.2 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.

10.114.4 Member Data Documentation

10.114.4.1 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.

10.114.4.2 std::string coco::initializer::__name [protected]

This is the identifier string for an initializer.

Definition at line 78 of file initializer.h.

10.114.4.3 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.

10.114.4.4 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.

10.114.4.5 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:

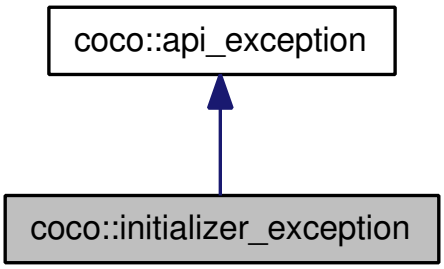
- initializer.h

10.115 coco::initializer_exception Class Reference

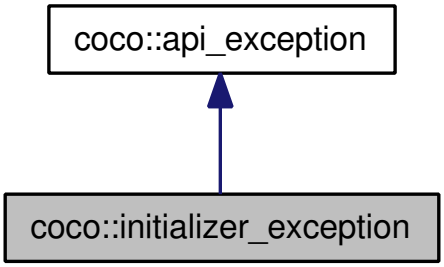
Initializer exception class.

`#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 api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.115.1 Detailed Description

This is the exceptions class thrown by initializer subclasses. Every properly coded COCONUT initializer module should only throw exceptions of this type.

Definition at line 54 of file initializer.h.

10.115.2 Constructor & Destructor Documentation

10.115.2.1 coco::initializer_exception::initializer_exception (const std::string & *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 58 of file initializer.h.

10.115.2.2 coco::initializer_exception::initializer_exception (const char * *msg*) [inline]

Constructor, setting the message to `msg`.

Definition at line 61 of file initializer.h.

10.115.2.3 virtual coco::initializer_exception::~initializer_exception () throw () [inline, virtual]

Standard Destructor

Definition at line 65 of file initializer.h.

10.115.3 Member Function Documentation

10.115.3.1 virtual std::string coco::api_exception::message () const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file api_exception.h.

10.115.3.2 virtual api_exception_type coco::api_exception::type () const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file api_exception.h.

10.115.3.3 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.

10.115.3.4 virtual char const* coco::api_exception::what () const throw () [inline, virtual, inherited]

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:

- initializer.h

10.116 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 & intersect (const interval &x)
- interval & hull (const interval &x)
- interval & hulldiff (const interval &x)
- 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_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
- 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_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 & intersect (const interval &x)
- interval & hull (const interval &x)
- interval & hulldiff (const interval &x)
- 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)

10.116.1 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 112 of file interval_boost.h.

10.116.2 Constructor & Destructor Documentation

10.116.2.1 coco::interval::interval (double *lo*, double *up*) [inline]

Definition at line 134 of file interval_boost.h.

10.116.2.2 coco::interval::interval (double *d* = 0) [inline]

Definition at line 135 of file interval_boost.h.

10.116.2.3 coco::interval::interval (int *d*) [inline]

Definition at line 136 of file interval_boost.h.

10.116.2.4 coco::interval::interval (unsigned *d*) [inline]

Definition at line 137 of file interval_boost.h.

10.116.2.5 coco::interval::interval (long *d*) [inline]

Definition at line 138 of file interval_boost.h.

10.116.2.6 coco::interval::interval (unsigned long *d*) [inline]

Definition at line 139 of file interval_boost.h.

10.116.2.7 coco::interval::interval (const interval & *x*) [inline]

Definition at line 140 of file interval_boost.h.

10.116.2.8 coco::interval::interval (const interval_st & *x*) [inline]

Definition at line 141 of file interval_boost.h.

10.116.2.9 coco::interval::interval (double *lo*, double *up*) [inline]

Constructor, setting the interval to [lo,up].

Definition at line 86 of file interval_filib.h.

10.116.2.10 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.

10.116.2.11 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.

10.116.2.12 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.

10.116.2.13 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.

10.116.2.14 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.

10.116.2.15 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.

10.116.2.16 coco::interval::interval (const interval & *x*) [inline]

Standard Copy Constructor

Definition at line 100 of file interval_filib.h.

10.116.2.17 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.

10.116.2.18 coco::interval::interval (double *lo*, double *up*) [inline]

Constructor, setting the interval to [lo,up].

Definition at line 68 of file interval_profil.h.

10.116.2.19 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.

10.116.2.20 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.

10.116.2.21 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.

10.116.2.22 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.

10.116.2.23 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.

10.116.2.24 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.

10.116.2.25 coco::interval::interval (const interval & *x*) [inline]

Standard Copy Constructor

Definition at line 82 of file interval_profil.h.

10.116.2.26 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.

10.116.3 Member Function Documentation

10.116.3.1 __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.

10.116.3.2 __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.

10.116.3.3 __Base coco::interval::base () const [inline]

Definition at line 152 of file interval_boost.h.

10.116.3.4 bool coco::interval::contains (const interval & *x*) const [inline]

Check whether this interval contains `x`.

Definition at line 172 of file interval_profil.h.

10.116.3.5 bool coco::interval::contains (const interval & *x*) const [inline]

Check whether this interval contains `x`.

Definition at line 215 of file interval_filib.h.

10.116.3.6 bool coco::interval::contains (const interval & *x*) const [inline]

Definition at line 207 of file interval_boost.h.

10.116.3.7 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

10.116.3.8 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

10.116.3.9 double coco::interval::diam () const

10.116.3.10 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.

10.116.3.11 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.

10.116.3.12 bool coco::interval::disjoint (const interval & *x*) const [inline]

Definition at line 173 of file interval_boost.h.

10.116.3.13 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]

10.116.3.14 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]

10.116.3.15 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]

10.116.3.16 bool coco::interval::empty () const [inline]

This method checks whether the interval is empty.

Definition at line 100 of file interval_profil.h.

10.116.3.17 static interval coco::interval::EMPTY () [inline, static]

This returns a new empty interval.

Definition at line 97 of file interval_profil.h.

10.116.3.18 bool coco::interval::empty () const [inline]

This method checks whether the interval is empty.

Definition at line 121 of file interval_filib.h.

10.116.3.19 static interval coco::interval::EMPTY () [inline, static]

This returns a new empty interval.

Definition at line 118 of file interval_filib.h.

10.116.3.20 bool coco::interval::empty () const [inline]

Definition at line 156 of file interval_boost.h.

10.116.3.21 static interval coco::interval::EMPTY () [inline, static]

Definition at line 153 of file interval_boost.h.

10.116.3.22 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 183 of file interval_profil.h.

10.116.3.23 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 226 of file interval_filib.h.

10.116.3.24 void coco::interval::get_bounds (double & *a*, double & *b*) const [inline]

Definition at line 216 of file interval_boost.h.

10.116.3.25 interval& coco::interval::hull (const interval & *x*) [inline]

`y.hull(x)` builds the hull of `y` with `x`.

Definition at line 150 of file interval_profil.h.

10.116.3.26 interval coco::interval::hull (const interval & *x*) const [inline]

`y.hull(x)` returns the hull of `y` and `x`.

Definition at line 193 of file interval_filib.h.

10.116.3.27 interval& coco::interval::hull (const interval & *x*) [inline]

Definition at line 186 of file interval_boost.h.

10.116.3.28 interval& coco::interval::hulldiff (const interval & *x*) [inline]

`y.hulldiff(x)` constructs the hull of the set difference of `y` and `x`.

Definition at line 157 of file interval_profil.h.

10.116.3.29 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 200 of file interval_filib.h.

10.116.3.30 interval& coco::interval::hulldiff (const interval & *x*) [inline]

Definition at line 192 of file interval_boost.h.

10.116.3.31 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 172 of file interval_filib.h.

10.116.3.32 interval& coco::interval::hullwith (const interval & *x*) [inline]

`y.hullwith(x)` builds the hull of `y` with `x`.

Definition at line 165 of file interval_filib.h.

10.116.3.33 interval& coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and `x`.

10.116.3.34 interval& coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and `x`.

10.116.3.35 interval& coco::interval::imax (const interval & *x*)

The method changes this interval to the maximum of itself and `x`.

10.116.3.36 interval& coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and `x`.

10.116.3.37 interval& coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and `x`.

10.116.3.38 interval& coco::interval::imin (const interval & *x*)

The method changes this interval to the minimum of itself and `x`.

10.116.3.39 double coco::interval::inf () const [inline]

This method returns the lower bound of the interval.

Definition at line 113 of file interval_filib.h.

10.116.3.40 double coco::interval::inf () const [inline]

Definition at line 150 of file interval_boost.h.

10.116.3.41 interval& coco::interval::intersect (const interval & *x*) [inline]

`y.intersect(x)` intersects `y` with `x`.

Definition at line 142 of file interval_profil.h.

10.116.3.42 interval coco::interval::intersect (const interval & *x*) const [inline]

`y.intersect(x)` returns the intersection of `y` and `x`.

Definition at line 187 of file interval_filib.h.

10.116.3.43 interval& coco::interval::intersect (const interval & *x*) [inline]

Definition at line 179 of file interval_boost.h.

10.116.3.44 interval& coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

`i.intersect_div(j,k)` computes the interval hull of the set $i \cap (j/k)$.

10.116.3.45 interval& coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

`i.intersect_div(j,k)` computes the interval hull of the set $i \cap (j/k)$.

10.116.3.46 interval& coco::interval::intersect_div (const interval & *__i*, const interval & *__j*)

10.116.3.47 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 $x \mapsto \cos(lx+d)$.

10.116.3.48 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 $x \mapsto \cos(lx+d)$.

10.116.3.49 interval& coco::interval::intersect_invcos_wc (const interval & __i, double __l, double __d)

10.116.3.50 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 $x \mapsto e^{-(\frac{lx-m}{s})^2}$.

10.116.3.51 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 $x \mapsto e^{-(\frac{lx-m}{s})^2}$.

10.116.3.52 interval& coco::interval::intersect_invgauss_wc (const interval & __i, double __l, double __m, double __s)

10.116.3.53 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 $x \mapsto (lx)^n$.

10.116.3.54 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 $x \mapsto (lx)^n$.

10.116.3.55 interval& coco::interval::intersect_invpower_wc (const interval & __i, double __l, int __n)

10.116.3.56 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 $x \mapsto \sin(lx+d)$.

10.116.3.57 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 $x \mapsto \sin(lx+d)$.

10.116.3.58 interval& coco::interval::intersect_invsin_wc (const interval & __i, double __l, double __d)

10.116.3.59 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 $x \mapsto (lx+d)^2$.

10.116.3.60 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 $x \mapsto (lx+d)^2$.

10.116.3.61 interval& coco::interval::intersect_invsqr_wc (const interval & __i, double __l, double __d)

10.116.3.62 interval& coco::interval::intersect_power (const interval & __i, int __n)

`i.intersect_power(j,n)` computes the interval hull of the set $i \cap j^n$.

10.116.3.63 interval& coco::interval::intersect_power (const interval & __i, int __n)

`i.intersect_power(j,n)` computes the interval hull of the set $i \cap j^n$.

10.116.3.64 interval& coco::interval::intersect_power (const interval & __i, int __n)

10.116.3.65 interval& coco::interval::intersectwith (const interval & x) [inline]

`y.intersectwith(x)` intersects `y` with `x`.

Definition at line 159 of file interval_filib.h.

10.116.3.66 interval& coco::interval::ipow (int n)

The method changes this interval to the `n-th` power.

10.116.3.67 interval& coco::interval::ipow (int n)

The method changes this interval to the `n-th` power.

10.116.3.68 interval& coco::interval::ipow (int n)

The method changes this interval to the `n-th` power.

10.116.3.69 bool coco::interval::is_bounded () const [inline]

This method checks whether the interval is bounded.

Definition at line 118 of file interval_profil.h.

10.116.3.70 bool coco::interval::is_bounded () const [inline]

This method checks whether the interval is bounded.

Definition at line 136 of file interval_filib.h.

10.116.3.71 bool coco::interval::is_bounded () const [inline]

Definition at line 162 of file interval_boost.h.

10.116.3.72 bool coco::interval::is_empty () const [inline]

This method checks whether the interval is empty.

Definition at line 102 of file interval_profil.h.

10.116.3.73 bool coco::interval::is_empty () const [inline]

This method checks whether the interval is empty.

Definition at line 123 of file interval_filib.h.

10.116.3.74 bool coco::interval::is_empty () const [inline]

Definition at line 157 of file interval_boost.h.

10.116.3.75 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.

10.116.3.76 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.

10.116.3.77 bool coco::interval::is_entire () const [inline]

Definition at line 161 of file interval_boost.h.

10.116.3.78 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.

10.116.3.79 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.

10.116.3.80 bool coco::interval::is_thin () const [inline]

Definition at line 158 of file interval_boost.h.

10.116.3.81 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.

10.116.3.82 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.

10.116.3.83 bool coco::interval::is_unbounded_above () const [inline]

Definition at line 160 of file interval_boost.h.

10.116.3.84 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.

10.116.3.85 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.

10.116.3.86 bool coco::interval::is_unbounded_below () const [inline]

Definition at line 159 of file interval_boost.h.

10.116.3.87 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

10.116.3.88 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

10.116.3.89 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

10.116.3.90 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]

10.116.3.91 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]

10.116.3.92 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]

10.116.3.93 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]

10.116.3.94 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]

10.116.3.95 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]

10.116.3.96 interval& coco::interval::operator∗= (double *a*)

unary operator∗= for number

10.116.3.97 interval& coco::interval::operator∗= (const interval & *a*)

unary operator∗=

10.116.3.98 interval& coco::interval::operator∗= (double *a*)

unary operator∗= for number

10.116.3.99 interval& coco::interval::operator∗= (const interval & *a*)

unary operator∗=

10.116.3.100 interval& coco::interval::operator∗= (double *a*)

10.116.3.101 interval& coco::interval::operator∗= (const interval & *a*)

10.116.3.102 interval& coco::interval::operator+= (double *a*)

unary += for number

10.116.3.103 interval& coco::interval::operator+= (const interval & *a*)

unary +=

10.116.3.104 interval& coco::interval::operator+= (double *a*)

unary += for number

10.116.3.105 interval& coco::interval::operator+= (const interval & *a*)

unary +=

10.116.3.106 interval& coco::interval::operator+= (double *a*)

10.116.3.107 interval& coco::interval::operator+= (const interval & *a*)

10.116.3.108 interval coco::interval::operator- () const [inline]

Unary interval minus operation

Definition at line 204 of file interval_profil.h.

10.116.3.109 interval coco::interval::operator- () const [inline]

Unary interval minus operation

Definition at line 247 of file interval_filib.h.

10.116.3.110 interval coco::interval::operator- () const [inline]

Definition at line 230 of file interval_boost.h.

10.116.3.111 interval& coco::interval::operator-= (double *a*)

unary -= for number

10.116.3.112 interval& coco::interval::operator-= (const interval & *a*)

unary -=

10.116.3.113 interval& coco::interval::operator-= (double *a*)

unary -= for number

10.116.3.114 interval& coco::interval::operator-= (const interval & *a*)

unary -=

10.116.3.115 interval& coco::interval::operator-= (double *a*)

10.116.3.116 interval& coco::interval::operator-= (const interval & *a*)

10.116.3.117 interval& coco::interval::operator/= (double *a*)

unary operator/= for number

10.116.3.118 interval& coco::interval::operator/= (const interval & *a*)

unary operator/=

10.116.3.119 interval& coco::interval::operator/= (double *a*)

unary operator/= for number

10.116.3.120 interval& coco::interval::operator/= (const interval & *a*)

unary operator/=

10.116.3.121 interval& coco::interval::operator/= (double *a*)

10.116.3.122 interval& coco::interval::operator/= (const interval & *a*)

10.116.3.123 interval& coco::interval::operator= (const __Base & *_x*) [inline]

Assignment from the base interval class

Definition at line 200 of file interval_profil.h.

10.116.3.124 interval& coco::interval::operator= (unsigned long *d*) [inline]

Assign the thin interval [d,d].

Definition at line 197 of file interval_profil.h.

10.116.3.125 interval& coco::interval::operator= (long *d*) [inline]

Assign the thin interval [d,d].

Definition at line 196 of file interval_profil.h.

10.116.3.126 interval& coco::interval::operator= (unsigned *d*) [inline]

Assign the thin interval [d,d].

Definition at line 195 of file interval_profil.h.

10.116.3.127 interval& coco::interval::operator= (int *d*) [inline]

Assign the thin interval [d,d].

Definition at line 194 of file interval_profil.h.

10.116.3.128 interval& coco::interval::operator= (double *d*) [inline]

Assign the thin interval [d,d].

Definition at line 193 of file interval_profil.h.

10.116.3.129 interval& coco::interval::operator= (const __Base & *_x*) [inline]

Assignment from the base interval class

Definition at line 243 of file interval_filib.h.

10.116.3.130 interval& coco::interval::operator= (unsigned long *d*) [inline]

Assign the thin interval [d,d].

Definition at line 240 of file interval_filib.h.

10.116.3.131 interval& coco::interval::operator= (long *d*) [inline]

Assign the thin interval [d,d].

Definition at line 239 of file interval_filib.h.

10.116.3.132 interval& coco::interval::operator= (unsigned *d*) [inline]

Assign the thin interval [d,d].

Definition at line 238 of file interval_filib.h.

10.116.3.133 interval& coco::interval::operator= (int *d*) [inline]

Assign the thin interval [d,d].

Definition at line 237 of file interval_filib.h.

10.116.3.134 interval& coco::interval::operator= (double *d*) [inline]

Assign the thin interval [d,d].

Definition at line 236 of file interval_filib.h.

10.116.3.135 interval& coco::interval::operator= (unsigned long *d*) [inline]

Definition at line 226 of file interval_boost.h.

10.116.3.136 interval& coco::interval::operator= (long *d*) [inline]

Definition at line 225 of file interval_boost.h.

10.116.3.137 interval& coco::interval::operator= (unsigned *d*) [inline]

Definition at line 224 of file interval_boost.h.

10.116.3.138 interval& coco::interval::operator= (int *d*) [inline]

Definition at line 223 of file interval_boost.h.

10.116.3.139 interval& coco::interval::operator= (double *d*) [inline]

Definition at line 222 of file interval_boost.h.

10.116.3.140 interval& coco::interval::operator= (const interval & *x*) [inline]

Definition at line 221 of file interval_boost.h.

10.116.3.141 static int coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.116.3.142 static int const& coco::interval::precision () [static]

This method returns the output precision that is used by the output operator <<.

10.116.3.143 static int coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.116.3.144 static int const& coco::interval::precision () [static]

This method returns the output precision that is used by the output operator <<.

10.116.3.145 static int coco::interval::precision (int const & *p*) [static]

This method sets the output precision to p. The default value is 3.

10.116.3.146 static int const& coco::interval::precision () [static]

This method returns the output precision that is used by the output operator <<.

10.116.3.147 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.

10.116.3.148 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`.

10.116.3.149 double coco::interval::project (double __d) const [inline]

Definition at line 802 of file interval_boost.h.

10.116.3.150 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.

10.116.3.151 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.

10.116.3.152 bool coco::interval::proper_subset (const interval & x) const [inline]

Definition at line 169 of file interval_boost.h.

10.116.3.153 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.

10.116.3.154 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.

10.116.3.155 bool coco::interval::proper_superset (const interval & *x*) const [inline]

Definition at line 171 of file interval_boost.h.

10.116.3.156 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

10.116.3.157 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

10.116.3.158 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

10.116.3.159 double coco::interval::rel_width () const

`i.rel_width()` computes the relative width (diameter) of `i`.

10.116.3.160 double coco::interval::rel_width () const

`i.rel_width()` computes the relative width (diameter) of `i`.

10.116.3.161 double coco::interval::rel_width () const

10.116.3.162 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

10.116.3.163 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

10.116.3.164 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

10.116.3.165 interval& coco::interval::round_to_integer ()

This method rounds the interval inward to integer borders.

10.116.3.166 interval& coco::interval::round_to_integer ()

This method rounds the interval inward to integer borders.

10.116.3.167 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.

10.116.3.168 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.

10.116.3.169 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

10.116.3.170 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.

10.116.3.171 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.

10.116.3.172 void coco::interval::set (double *lo*, double *up*) [inline]

Definition at line 144 of file interval_boost.h.

10.116.3.173 void coco::interval::set_bounds (double *a*, double *b*) [inline]

Set the bounds of the interval to `a` and `b`.

Definition at line 185 of file interval_profil.h.

10.116.3.174 void coco::interval::set_bounds (double *a*, double *b*) [inline]

Set the bounds of the interval to `a` and `b`.

Definition at line 228 of file interval_filib.h.

10.116.3.175 void coco::interval::set_bounds (double *a*, double *b*) [inline]

Definition at line 217 of file interval_boost.h.

10.116.3.176 void coco::interval::set_lb (double *d*) [inline]

Set the lower bound of the interval to `d`.

Definition at line 187 of file interval_profil.h.

10.116.3.177 void coco::interval::set_lb (double *d*) [inline]

Set the lower bound of the interval to `d`.

Definition at line 230 of file interval_filib.h.

10.116.3.178 void coco::interval::set_lb (double *d*) [inline]

Definition at line 218 of file interval_boost.h.

10.116.3.179 void coco::interval::set_ub (double *d*) [inline]

Set the upper bound of the interval to `d`.

Definition at line 189 of file interval_profil.h.

10.116.3.180 void coco::interval::set_ub (double *d*) [inline]

Set the upper bound of the interval to `d`.

Definition at line 232 of file interval_filib.h.

10.116.3.181 void coco::interval::set_ub (double *d*) [inline]

Definition at line 219 of file interval_boost.h.

10.116.3.182 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 181 of file interval_profil.h.

10.116.3.183 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 224 of file interval_filib.h.

10.116.3.184 void coco::interval::setpair (double & *lo*, double & *up*) const [inline]

Definition at line 215 of file interval_boost.h.

10.116.3.185 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.

10.116.3.186 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.

10.116.3.187 bool coco::interval::subset (const interval & *x*) const [inline]

Definition at line 165 of file interval_boost.h.

10.116.3.188 double coco::interval::sup () const [inline]

This method returns the upper bound of the interval.

Definition at line 111 of file interval_filib.h.

10.116.3.189 double coco::interval::sup () const [inline]

Definition at line 151 of file interval_boost.h.

10.116.3.190 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.

10.116.3.191 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.

10.116.3.192 bool coco::interval::superset (const interval & *x*) const [inline]

Definition at line 167 of file interval_boost.h.

10.116.3.193 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

10.116.3.194 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

10.116.3.195 double coco::interval::width () const

10.116.4 Friends And Related Function Documentation

10.116.4.1 interval abs (const interval & *x*) [friend]

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

10.116.4.2 interval abs (const interval & *x*) [friend]

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

10.116.4.3 interval acos (const interval & *x*) [friend]

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

10.116.4.4 interval acos (const interval & *x*) [friend]

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

10.116.4.5 interval acosh (const interval & *x*) [friend]

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

10.116.4.6 interval acosh (const interval & *x*) [friend]

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

10.116.4.7 interval acot (const interval & *x*) [friend]

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

10.116.4.8 interval acot (const interval & *x*) [friend]

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

10.116.4.9 interval acoth (const interval & *x*) [friend]

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

10.116.4.10 interval acoth (const interval & *x*) [friend]

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

10.116.4.11 interval asin (const interval & *x*) [friend]

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

10.116.4.12 interval asin (const interval & *x*) [friend]

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

10.116.4.13 interval asinh (const interval & *x*) [friend]

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

10.116.4.14 interval asinh (const interval & *x*) [friend]

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

10.116.4.15 interval atan (const interval & *x*) [friend]

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

10.116.4.16 interval atan (const interval & *x*) [friend]

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

10.116.4.17 interval atanh (const interval & *x*) [friend]

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

10.116.4.18 interval atanh (const interval & *x*) [friend]

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

10.116.4.19 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.116.4.20 interval cancel (const interval & *a*, const interval & *b*) [friend]

10.116.4.21 interval cos (const interval & *x*) [friend]

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

10.116.4.22 interval cos (const interval & *x*) [friend]

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

10.116.4.23 interval cosh (const interval & *x*) [friend]

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

10.116.4.24 interval cosh (const interval & *x*) [friend]

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

10.116.4.25 interval cot (const interval & *x*) [friend]

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

10.116.4.26 interval cot (const interval & *x*) [friend]

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

10.116.4.27 interval coth (const interval & *x*) [friend]

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

10.116.4.28 interval coth (const interval & *x*) [friend]

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

10.116.4.29 double diam (const interval &) [friend]

diam(a) == a.diam()

10.116.4.30 double dist (const interval & *x*, const interval & *y*) [friend]

Same as x.dist(y)

10.116.4.31 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.

10.116.4.32 interval division_part2 (const interval & *x*, const interval & *y*, bool *b*) [friend]

10.116.4.33 double downward_divides (const double & *a*, const double & *b*) [friend]

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

10.116.4.34 double downward_minus (const double & *a*, const double & *b*) [friend]

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

10.116.4.35 double downward_multiplies (const double & *a*, const double & *b*) [friend]

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

10.116.4.36 double downward_plus (const double & *a*, const double & *b*) [friend]

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

10.116.4.37 interval exp (const interval & *x*) [friend]

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

10.116.4.38 interval exp (const interval & *x*) [friend]

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

10.116.4.39 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.116.4.40 interval exp10 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 10) of the interval x.

10.116.4.41 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.116.4.42 interval exp2 (const interval & *x*) [friend]

Returns an interval enclosure of the exponential (base 2) of the interval x.

10.116.4.43 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.116.4.44 interval expm1 (const interval & *x*) [friend]

Returns an interval enclosure of exp(x)-1

10.116.4.45 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.116.4.46 interval imax (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the maximum of two intervals x and y.

10.116.4.47 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.116.4.48 interval imin (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of the minimum of two intervals x and y.

10.116.4.49 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.116.4.50 interval log (const interval & *x*) [friend]

Returns an interval enclosure of the natural logarithm of the interval x.

10.116.4.51 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.116.4.52 interval log10 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 10) of the interval x.

10.116.4.53 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.116.4.54 interval log1p (const interval & *x*) [friend]

Returns an interval enclosure of log(1+x).

10.116.4.55 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.116.4.56 interval log2 (const interval & *x*) [friend]

Returns an interval enclosure of the logarithm (base 2) of the interval x.

10.116.4.57 double mag (const interval &) [friend]

mag(a) == a.mag()

10.116.4.58 double mid (const interval &) [friend]

mid(a) == a.mid()

10.116.4.59 double mig (const interval &) [friend]

mig(a) == a.mig()

10.116.4.60 template<class _TC > bool operator!= (const interval & *__i*, const _TC & *__d*) [friend]

10.116.4.61 bool operator!= (const interval & *__i*, const interval & *__d*) [friend]

Comparison operator (not equal)

Definition at line 490 of file interval_boost.h.

10.116.4.62 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.116.4.63 interval operator∗ (const interval & *a*, double *b*) [friend]

10.116.4.64 interval operator∗ (double *a*, const interval & *b*) [friend]

10.116.4.65 interval operator∗ (double *b*, const interval & *a*) [friend]

10.116.4.66 interval operator∗ (const interval & *a*, double *b*) [friend]

10.116.4.67 interval operator∗ (const interval & *a*, const interval & *b*) [friend]

This operator computes a∗b as interval.

10.116.4.68 interval operator+ (double *b*, const interval & *a*) [friend]

10.116.4.69 interval operator+ (const interval & *a*, double *b*) [friend]

10.116.4.70 interval operator+ (const interval & *a*, const interval & *b*) [friend]

This operator computes a+b as interval.

10.116.4.71 interval operator+ (double *b*, const interval & *a*) [friend]

10.116.4.72 interval operator+ (const interval & *a*, double *b*) [friend]

10.116.4.73 interval operator+ (const interval & *a*, const interval & *b*) [friend]

This operator computes a+b as interval.

10.116.4.74 interval operator- (double *a*, const interval & *b*) [friend]

10.116.4.75 interval operator- (const interval & *a*, double *b*) [friend]

10.116.4.76 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.116.4.77 interval operator- (double *b*, const interval & *a*) [friend]

10.116.4.78 interval operator- (const interval & *a*, double *b*) [friend]

10.116.4.79 interval operator- (const interval & *a*, const interval & *b*) [friend]

This operator computes a-b as interval.

10.116.4.80 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.116.4.81 interval operator/ (const interval & *a*, double *b*) [friend]

10.116.4.82 interval operator/ (double *a*, const interval & *b*) [friend]

10.116.4.83 interval operator/ (double *b*, const interval & *a*) [friend]

10.116.4.84 interval operator/ (const interval & *a*, double *b*) [friend]

10.116.4.85 interval operator/ (const interval & *a*, const interval & *b*) [friend]

This operator computes a/b as interval.

10.116.4.86 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.116.4.87 std::ostream& operator<< (std::ostream & *s*, const interval & *a*) [friend]

10.116.4.88 template<class _TC > bool operator== (const interval & *__i*, const _TC & *__d*) [friend]

10.116.4.89 bool operator== (const interval & *__i*, const interval & *__d*) [friend]

friends

Comparison operator (equal)

Definition at line 486 of file interval_boost.h.

10.116.4.90 interval pow (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.116.4.91 interval pow (const interval & *x*, const interval & *y*) [friend]

Returns an interval enclosure of x^y = exp(y∗log(x)).

10.116.4.92 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.116.4.93 interval power (const interval & *x*, int *n*) [friend]

Returns an interval enclosure of x^n.

10.116.4.94 double rad (const interval &) [friend]

rad(a) == a.rad()

10.116.4.95 double relDiam (const interval &) [friend]

relDiam(a) == a.relDiam()

10.116.4.96 interval sin (const interval & *x*) [friend]

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

10.116.4.97 interval sin (const interval & *x*) [friend]

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

10.116.4.98 interval sinh (const interval & *x*) [friend]

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

10.116.4.99 interval sinh (const interval & *x*) [friend]

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

10.116.4.100 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.116.4.101 interval sqr (const interval & *x*) [friend]

Returns an interval enclosure of x^2.

10.116.4.102 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.116.4.103 interval sqrt (const interval & *x*) [friend]

Returns an interval enclosure of the square root of the interval x.

10.116.4.104 interval tan (const interval & *x*) [friend]

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

10.116.4.105 interval tan (const interval & *x*) [friend]

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

10.116.4.106 interval tanh (const interval & *x*) [friend]

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

10.116.4.107 interval tanh (const interval & *x*) [friend]

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

10.116.4.108 double upward_divides (const double & *a*, const double & *b*) [friend]

This function divides a by b, rounding upwards.

10.116.4.109 double upward_minus (const double & *a*, const double & *b*) [friend]

This function subtracts b from a, rounding upwards.

10.116.4.110 double upward_multiplies (const double & *a*, const double & *b*) [friend]

This function multiplies a by b, rounding upwards.

10.116.4.111 double upward_plus (const double & *a*, const double & *b*) [friend]

This function adds a and b, rounding upwards.

10.116.4.112 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

10.117 coco::interval_eval Class Reference

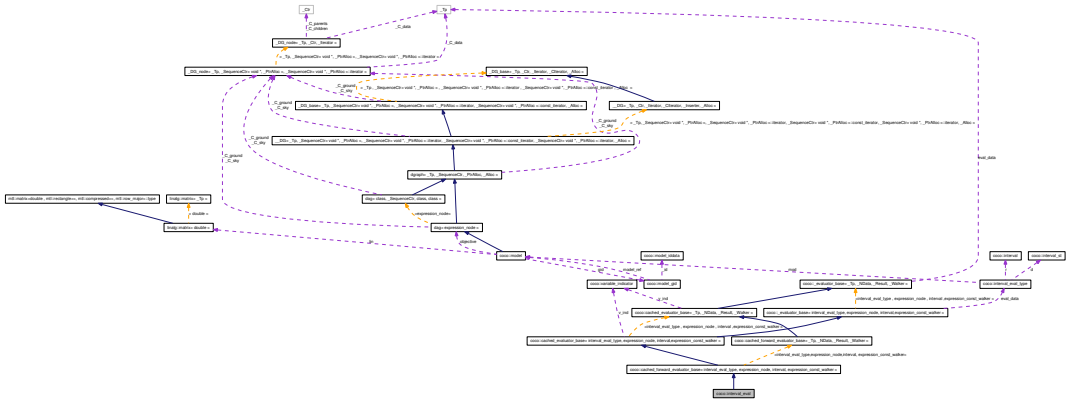
Forward function range evaluation.

`#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
- typedef interval_eval_type data_type

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 void cleanup (const node_data_type &__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 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)

Protected Attributes

- const variable_indicator * v_ind
- interval_eval_type eval_data

10.117.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural interval extension.

Definition at line 71 of file int_evaluator.h.

10.117.2 Member Typedef Documentation

10.117.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.117.2.2 typedef interval_eval_type coco::_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.117.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.117.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.117.3 Constructor & Destructor Documentation

10.117.3.1 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 138 of file int_evaluator.h.

10.117.3.2 coco::interval_eval::interval_eval (const interval_eval & *__v*) [inline]

Standard Copy Constructor

Definition at line 152 of file int_evaluator.h.

10.117.3.3 coco::interval_eval::~interval_eval () [inline]

Standard Destructor

Definition at line 155 of file int_evaluator.h.

10.117.4 Member Function Documentation

10.117.4.1 void coco::interval_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 231 of file int_evaluator.h.

10.117.4.2 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::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 709 of file int_evaluator.h.

10.117.4.3 virtual void coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::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 804 of file evaluator.h.

10.117.4.4 int coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.117.4.5 int coco::interval_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 174 of file int_evaluator.h.

10.117.4.6 void coco::interval_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 172 of file int_evaluator.h.

10.117.4.7 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::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 83 of file int_evaluator.h.

10.117.4.8 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 164 of file int_evaluator.h.

10.117.4.9 void coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< interval_eval_type, expression_node, interval,expression_-const_walker >.

Definition at line 744 of file evaluator.h.

10.117.4.10 int coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.117.4.11 void coco::interval_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 242 of file int_evaluator.h.

10.117.4.12 expression_const_walker coco::interval_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 158 of file int_evaluator.h.

10.117.4.13 int coco::interval_eval::update (const expression_node & *__data*, const interval & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 267 of file int_evaluator.h.

10.117.4.14 int coco::interval_eval::update (const interval & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< interval_eval_type, expression_node, interval, expression_const_walker >.

Definition at line 261 of file int_evaluator.h.

10.117.4.15 return_value coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.117.4.16 int coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< interval_eval_type, expression_node, interval,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.117.4.17 void coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::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 771 of file evaluator.h.

10.117.4.18 return_value coco::cached_forward_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< interval_eval_type, expression_node, interval,expression_-const_walker >.

Definition at line 767 of file evaluator.h.

10.117.5 Member Data Documentation

10.117.5.1 interval_eval_type coco::_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.117.5.2 const variable_indicator* coco::cached_evaluator_base< interval_eval_type , expression_node , interval , expression_const_walker >::v_ind [protected, inherited]

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:

- int_evaluator.h

10.118 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

10.118.1 Detailed Description

This class is the visitor data type for the interval_eval evaluator.

Definition at line 53 of file int_evaluator.h.

10.118.2 Member Data Documentation

10.118.2.1 std::vector<interval>* coco::interval_eval_type::cache

the function range cache

Definition at line 56 of file int_evaluator.h.

10.118.2.2 interval_st coco::interval_eval_type::d

Definition at line 58 of file int_evaluator.h.

10.118.2.3 bool coco::interval_eval_type::do_intersect

compute an intersection with the node range?

Definition at line 62 of file int_evaluator.h.

10.118.2.4 int coco::interval_eval_type::info

Definition at line 58 of file int_evaluator.h.

10.118.2.5 const model* coco::interval_eval_type::mod

the DAG

Definition at line 57 of file int_evaluator.h.

10.118.2.6 unsigned int coco::interval_eval_type::n

children counter

Definition at line 61 of file int_evaluator.h.

10.118.2.7 void* coco::interval_eval_type::p

Definition at line 58 of file int_evaluator.h.

10.118.2.8 interval coco::interval_eval_type::r

return value

Definition at line 60 of file int_evaluator.h.

10.118.2.9 union { ... } coco::interval_eval_type::u

additional data for complex nodes

10.118.2.10 const std::vector<interval>* coco::interval_eval_type::x

the box

Definition at line 55 of file int_evaluator.h.

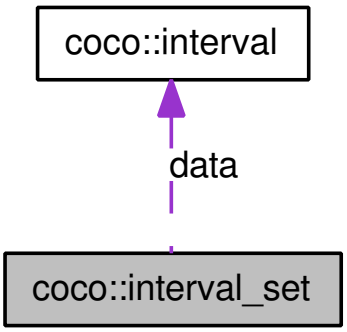
The documentation for this struct was generated from the following file:

- int_evaluator.h

10.119 coco::interval_set Class Reference

`#include <interval_set.h>`

Collaboration diagram for coco::interval_set:



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)
- unsigned int num_of_gaps () const
- bool is_empty () const
- bool is_subset (const interval_set &a) const
- bool is_subset (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
- interval_set & round_to_integer ()
- interval_set & intersectwith (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_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)

10.119.1 Detailed Description

Definition at line 46 of file interval_set.h.

10.119.2 Constructor & Destructor Documentation

10.119.2.1 coco::interval_set::interval_set (void) [inline]

Definition at line 175 of file interval_set.h.

10.119.2.2 coco::interval_set::interval_set (double *a*) [inline]

Definition at line 180 of file interval_set.h.

10.119.2.3 coco::interval_set::interval_set (double *a*, double *b*) [inline]

Definition at line 186 of file interval_set.h.

10.119.2.4 coco::interval_set::interval_set (const interval & *a*) [inline]

Definition at line 192 of file interval_set.h.

10.119.2.5 coco::interval_set::interval_set (const interval & *a*, const interval & *b*) [inline]

Definition at line 198 of file interval_set.h.

10.119.2.6 coco::interval_set::interval_set (const interval_set & *a*) [inline]

Definition at line 204 of file interval_set.h.

10.119.2.7 coco::interval_set::~interval_set () [inline, virtual]

Definition at line 211 of file interval_set.h.

10.119.3 Member Function Documentation

10.119.3.1 void coco::interval_set::add (const interval_set & *a*) [inline]

Definition at line 254 of file interval_set.h.

10.119.3.2 void coco::interval_set::add (const interval & *a*)

Definition at line 93 of file interval_set.cc.

10.119.3.3 void coco::interval_set::conc (const interval & *a*) [inline]

Definition at line 291 of file interval_set.h.

10.119.3.4 bool coco::interval_set::contains (const interval & *a*) const [inline]

Definition at line 280 of file interval_set.h.

10.119.3.5 bool coco::interval_set::contains (double *a*) const [inline]

Definition at line 286 of file interval_set.h.

10.119.3.6 int coco::interval_set::find (const interval & *a*) [inline]

Definition at line 272 of file interval_set.h.

10.119.3.7 unsigned int coco::interval_set::get_fillext () const [inline]

Definition at line 233 of file interval_set.h.

10.119.3.8 unsigned int coco::interval_set::get_maxlen () const [inline]

Definition at line 225 of file interval_set.h.

10.119.3.9 int coco::interval_set::getLength () const [inline]

Definition at line 216 of file interval_set.h.

10.119.3.10 bool coco::interval_set::getSorting () const [inline]

Definition at line 221 of file interval_set.h.

10.119.3.11 interval coco::interval_set::hull () const

Definition at line 291 of file interval_set.cc.

10.119.3.12 void coco::interval_set::intersect_div (const interval_set & *a*, const interval_set & *b*)

Definition at line 346 of file interval_set.cc.

10.119.3.13 void coco::interval_set::intersect_invcos_wc (const interval_set & *a*, double *b*, double *c*)

Definition at line 403 of file interval_set.cc.

10.119.3.14 void coco::interval_set::intersect_invgauss_wc (const interval_set & *a*, double *b*, double *c*, double *d*)

Definition at line 416 of file interval_set.cc.

10.119.3.15 void coco::interval_set::intersect_invpower_wc (const interval_set & *a*, double *b*, int *n*)

Definition at line 377 of file interval_set.cc.

10.119.3.16 void coco::interval_set::intersect_invsin_wc (const interval_set & *a*, double *b*, double *c*)

Definition at line 390 of file interval_set.cc.

10.119.3.17 void coco::interval_set::intersect_invsqr_wc (const interval_set & *a*, double *b*, double *c*)

Definition at line 359 of file interval_set.cc.

10.119.3.18 void coco::interval_set::intersect_power (const interval_set & *a*, int *n*)

Definition at line 333 of file interval_set.cc.

10.119.3.19 interval_set & coco::interval_set::intersectwith (const interval_set & *a*)

Definition at line 320 of file interval_set.cc.

10.119.3.20 bool coco::interval_set::is_empty () const [inline]

Definition at line 241 of file interval_set.h.

10.119.3.21 bool coco::interval_set::is_subset (const interval & *a*) const

Definition at line 89 of file interval_set.cc.

10.119.3.22 bool coco::interval_set::is_subset (const interval_set & *a*) const

Definition at line 75 of file interval_set.cc.

10.119.3.23 interval coco::interval_set::maximum () const

Definition at line 276 of file interval_set.cc.

10.119.3.24 interval coco::interval_set::minimum () const

Definition at line 261 of file interval_set.cc.

10.119.3.25 unsigned int coco::interval_set::num_of_gaps () const [inline]

Definition at line 300 of file interval_set.h.

10.119.3.26 interval_set & coco::interval_set::operator*= (const interval_set & *a*)

Definition at line 439 of file interval_set.cc.

10.119.3.27 interval_set & coco::interval_set::operator+= (const interval_set & *a*)

Definition at line 429 of file interval_set.cc.

10.119.3.28 interval_set & coco::interval_set::operator-= (const interval_set & *a*)

Definition at line 434 of file interval_set.cc.

10.119.3.29 interval_set & coco::interval_set::operator/= (const interval_set & *a*)

Definition at line 444 of file interval_set.cc.

10.119.3.30 interval_set & coco::interval_set::operator= (const interval_set & *a*)

Definition at line 56 of file interval_set.cc.

10.119.3.31 interval & coco::interval_set::operator[ ] (int *pos*) const [inline]

Definition at line 245 of file interval_set.h.

10.119.3.32 interval_set coco::interval_set::remove (const interval & *a*)

Definition at line 230 of file interval_set.cc.

10.119.3.33 interval coco::interval_set::remove (int *pos*)

Definition at line 203 of file interval_set.cc.

10.119.3.34 interval coco::interval_set::removeFirst () [inline]

Definition at line 260 of file interval_set.h.

10.119.3.35 interval_set & coco::interval_set::round_to_integer ()

This method rounds the interval inward to integer borders.

Definition at line 303 of file interval_set.cc.

10.119.3.36 void coco::interval_set::set_fillext (unsigned int *m*) [inline]

Definition at line 237 of file interval_set.h.

10.119.3.37 void coco::interval_set::set_maxlen (unsigned int *m*) [inline]

Definition at line 229 of file interval_set.h.

10.119.3.38 void coco::interval_set::setSorting (bool *s*)

Definition at line 69 of file interval_set.cc.

10.119.3.39 void coco::interval_set::trimintervaloSize () [inline]

10.119.3.40 void coco::interval_set::trimToSize () [inline]

Definition at line 264 of file interval_set.h.

10.119.3.41 double coco::interval_set::width () const

Definition at line 298 of file interval_set.cc.

The documentation for this class was generated from the following files:

- interval_set.h
- interval_set.cc

10.120 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

10.120.1 Detailed Description

This is a constructor-free interval data type for storing intervals within unions.

Definition at line 105 of file interval_boost.h.

10.120.2 Member Function Documentation

10.120.2.1 interval_st& coco::interval_st::operator= (const interval & *__i*)

10.120.2.2 interval_st& coco::interval_st::operator= (const interval & *__i*)

10.120.2.3 interval_st & coco::interval_st::operator= (const interval & *__i*) [inline]

Definition at line 725 of file interval_boost.h.

10.120.3 Member Data Documentation

10.120.3.1 double coco::interval_st::l

Definition at line 107 of file interval_boost.h.

10.120.3.2 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:

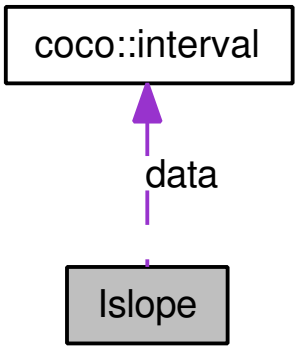
- interval_boost.h
- interval_filib.h
- interval_profil.h

10.121 Islope Class Reference

Class for computing intervalslopes up to order 2.

`#include <Islope.h>`

Collaboration diagram for Islope:



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)

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

10.121.1 Detailed Description

Definition at line 16 of file Islope.h.

10.121.2 Constructor & Destructor Documentation

10.121.2.1 Islope::Islope (void) [inline]

Constructor for the slope of the constant function f(x)=0. The slope order is 1.

Definition at line 115 of file Islope.h.

10.121.2.2 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 128 of file Islope.h.

10.121.2.3 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 160 of file Islope.h.

10.121.2.4 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 193 of file Islope.h.

10.121.2.5 Islope::Islope (const Islope & *a*) [inline]

Copy Constructor

Definition at line 225 of file Islope.h.

10.121.2.6 Islope::~Islope () [inline, virtual]

Deconstructor

Definition at line 1756 of file Islope.h.

10.121.3 Member Function Documentation

10.121.3.1 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 366 of file Islope.h.

10.121.3.2 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 333 of file Islope.h.

10.121.3.3 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 300 of file Islope.h.

10.121.3.4 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 267 of file Islope.h.

10.121.3.5 Islope & Islope::operator= (const Islope & *a*) [inline]

Operator = for the slope class.

Definition at line 236 of file Islope.h.

10.121.3.6 I & Islope::operator[ ] (int *pos*) const [inline]

Definition at line 258 of file Islope.h.

10.121.3.7 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 399 of file Islope.h.

10.121.4 Member Data Documentation

10.121.4.1 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.

10.121.4.2 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 file:

- Islope.h

10.122 coco::Islope_eval Class Reference

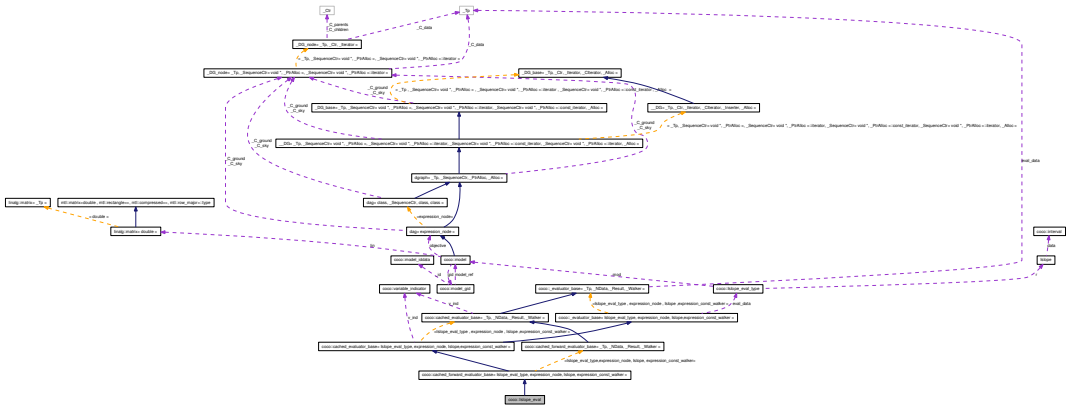
Forward function range evaluation.

`#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
- typedef Islope_eval_type data_type

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 void cleanup (const node_data_type &__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 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)

Protected Attributes

- const variable_indicator * v_ind
- Islope_eval_type eval_data

10.122.1 Detailed Description

This class is a cached forward evaluator performing a function range evaluation by natural Islope extension.

Definition at line 75 of file Islope_evaluator.h.

10.122.2 Member Typedef Documentation

10.122.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.122.2.2 typedef Islope_eval_type coco::_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.122.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.122.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.122.3 Constructor & Destructor Documentation

10.122.3.1 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.

10.122.3.2 coco::Islope_eval::Islope_eval (const Islope_eval & *__v*) [inline]

Standard Copy Constructor

Definition at line 158 of file Islope_evaluator.h.

10.122.3.3 coco::Islope_eval::~Islope_eval () [inline]

Standard Destructor

Definition at line 161 of file Islope_evaluator.h.

10.122.4 Member Function Documentation

10.122.4.1 void coco::Islope_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 245 of file Islope_evaluator.h.

10.122.4.2 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::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 697 of file Islope_evaluator.h.

10.122.4.3 virtual void coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::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 804 of file evaluator.h.

10.122.4.4 int coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< Islope_eval_type, expression_node, Islope,expression_-const_walker >.

Definition at line 750 of file evaluator.h.

10.122.4.5 int coco::Islope_eval::initialize (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 182 of file Islope_evaluator.h.

10.122.4.6 void coco::Islope_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 180 of file Islope_evaluator.h.

10.122.4.7 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::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 88 of file Islope_evaluator.h.

10.122.4.8 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.

10.122.4.9 void coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< Islope_eval_type, expression_node, Islope,expression_-const_walker >.

Definition at line 744 of file evaluator.h.

10.122.4.10 int coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.122.4.11 void coco::Islope_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 252 of file Islope_evaluator.h.

10.122.4.12 expression_const_walker coco::Islope_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< Islope_eval_type, expression_node, Islope,expression_const_walker >.

Definition at line 164 of file Islope_evaluator.h.

10.122.4.13 int coco::Islope_eval::update (const expression_node & *__data*, const Islope & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 264 of file Islope_evaluator.h.

10.122.4.14 int coco::Islope_eval::update (const Islope & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_forward_evaluator_base< Islope_eval_type, expression_node, Islope, expression_const_walker >.

Definition at line 257 of file Islope_evaluator.h.

10.122.4.15 return_value coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< Islope_eval_type, expression_node, Islope,expression_-const_walker >.

Definition at line 762 of file evaluator.h.

10.122.4.16 int coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< Islope_eval_type, expression_node, Islope,expression_-const_walker >.

Definition at line 756 of file evaluator.h.

10.122.4.17 void coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::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 771 of file evaluator.h.

10.122.4.18 return_value coco::cached_forward_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< Islope_eval_type, expression_node, Islope,expression_-const_walker >.

Definition at line 767 of file evaluator.h.

10.122.5 Member Data Documentation

10.122.5.1 Islope_eval_type coco::_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.122.5.2 const variable_indicator* coco::cached_evaluator_base< Islope_eval_type , expression_node , Islope , expression_const_walker >::v_ind [protected, inherited]

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:

- Islope_evaluator.h

10.123 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

10.123.1 Detailed Description

This class is the visitor data type for the Islope_eval evaluator.

Definition at line 57 of file Islope_evaluator.h.

10.123.2 Member Data Documentation

10.123.2.1 std::vector<Islope>* coco::Islope_eval_type::cache

function range the function range cache

Definition at line 60 of file Islope_evaluator.h.

10.123.2.2 int coco::Islope_eval_type::deg

degree of slope

Definition at line 65 of file Islope_evaluator.h.

10.123.2.3 const model* coco::Islope_eval_type::mod

the DAG

Definition at line 61 of file Islope_evaluator.h.

10.123.2.4 unsigned int coco::Islope_eval_type::n

children counter

Definition at line 64 of file Islope_evaluator.h.

10.123.2.5 void* coco::Islope_eval_type::p

additional data for

Definition at line 62 of file Islope_evaluator.h.

10.123.2.6 Islope coco::Islope_eval_type::r

return value

Definition at line 63 of file Islope_evaluator.h.

10.123.2.7 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

10.124 coco::islp_eval Class Reference

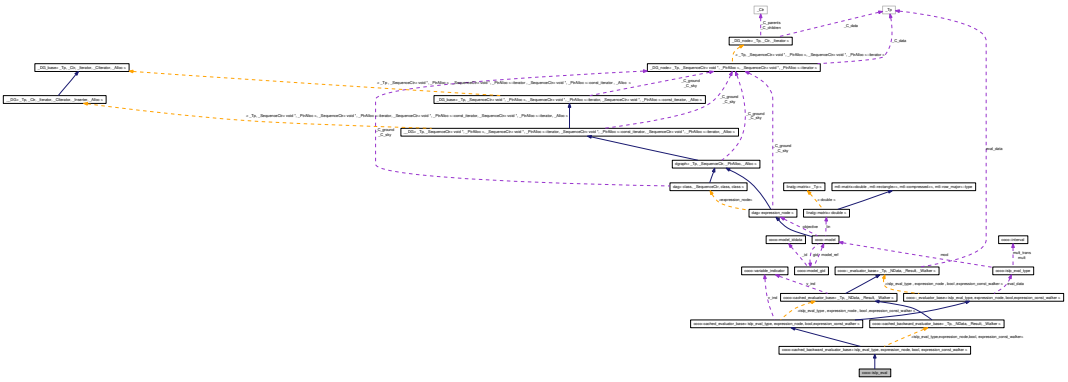
Backward first order slope evaluation with prepared first order slope data.

`#include <islp_evaluator.h>`

Inheritance diagram for coco::islp_eval:



Collaboration diagram for coco::islp_eval:



Public Types

- typedef _Base::node_data_type node_data_type
- typedef _Base::return_value return_value
- typedef _Base::const_walker const_walker
- typedef islp_eval_type data_type

Public Member Functions

- islp_eval (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 (const islp_eval &__d)
- ~islp_eval ()

- 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 ()

- 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)

Protected Attributes

- const variable_indicator * v_ind
- islp_eval_type eval_data

10.124.1 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 1448 of file islp_evaluator.h.

10.124.2 Member Typedef Documentation

10.124.2.1 typedef _Base::const_walker coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< islp_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 853 of file evaluator.h.

10.124.2.2 typedef islp_eval_type coco::_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.124.2.3 typedef _Base::node_data_type coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< islp_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 849 of file evaluator.h.

10.124.2.4 typedef _Base::return_value coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< islp_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 851 of file evaluator.h.

10.124.3 Constructor & Destructor Documentation

10.124.3.1 coco::islp_eval::islp_eval (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 1481 of file islp_evaluator.h.

10.124.3.2 coco::islp_eval::islp_eval (const islp_eval & *__d*) [inline]

Standard Copy Constructor

Definition at line 1496 of file islp_evaluator.h.

10.124.3.3 coco::islp_eval::~islp_eval () [inline]

Standard Destructor

Definition at line 1499 of file islp_evaluator.h.

10.124.4 Member Function Documentation

10.124.4.1 int coco::islp_eval::calculate (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1537 of file islp_evaluator.h.

10.124.4.2 bool coco::islp_eval::calculate_value (bool *eval_all*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1635 of file islp_evaluator.h.

10.124.4.3 void coco::islp_eval::cleanup (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1589 of file islp_evaluator.h.

10.124.4.4 int coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< islp_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 879 of file evaluator.h.

10.124.4.5 void coco::islp_eval::initialize () [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1531 of file islp_evaluator.h.

10.124.4.6 bool coco::islp_eval::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::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1460 of file islp_evaluator.h.

10.124.4.7 void coco::islp_eval::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 1504 of file islp_evaluator.h.

10.124.4.8 void coco::islp_eval::new_result (std::vector< interval > & *__islp*) [inline]

This method changes the result vector to __islp.

Definition at line 1512 of file islp_evaluator.h.

10.124.4.9 void coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::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. It is translated into a call to cleanup.

Reimplemented from coco::_evaluator_base< islp_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 874 of file evaluator.h.

10.124.4.10 int coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Reimplemented from coco::cached_evaluator_base< islp_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 862 of file evaluator.h.

10.124.4.11 void coco::islp_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1601 of file islp_evaluator.h.

10.124.4.12 void coco::islp_eval::set_mult (interval *scal*) [inline]

This method causes the slope to be multiplied by `scal`.

Definition at line 1518 of file islp_evaluator.h.

10.124.4.13 expression_const_walker coco::islp_eval::short_cut_to (const expression_node & *__data*) [inline, virtual]

NOP version, not needed

Reimplemented from coco::cached_evaluator_base< islp_eval_type, expression_node, bool,expression_-const_walker >.

Definition at line 1525 of file islp_evaluator.h.

10.124.4.14 int coco::islp_eval::update (const expression_node & *__data*, const bool & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1616 of file islp_evaluator.h.

10.124.4.15 int coco::islp_eval::update (const bool & *__rval*) [inline, virtual]

This is an evaluator method, as defined for the various evaluators.

Reimplemented from coco::cached_backward_evaluator_base< islp_eval_type, expression_node, bool, expression_const_walker >.

Definition at line 1609 of file islp_evaluator.h.

10.124.4.16 return_value coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter false.

Reimplemented from coco::_evaluator_base< islp_eval_type, expression_node, bool,expression_const_-walker >.

Definition at line 891 of file evaluator.h.

10.124.4.17 int coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< islp_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 885 of file evaluator.h.

10.124.4.18 void coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::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 900 of file evaluator.h.

10.124.4.19 return_value coco::cached_backward_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< islp_eval_type, expression_node, bool,expression_const_walker >.

Definition at line 896 of file evaluator.h.

10.124.5 Member Data Documentation

10.124.5.1 islp_eval_type coco::_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.124.5.2 const variable_indicator* coco::cached_evaluator_base< islp_eval_type , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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:

- islp_evaluator.h

10.125 coco::islp_eval_type Struct Reference

Visitor data for islp_eval.

`#include <islp_evaluator.h>`

Collaboration diagram for coco::islp_eval_type:



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

10.125.1 Detailed Description

This class is the visitor data type for the islp_eval evaluator.

Definition at line 1431 of file islp_evaluator.h.

10.125.2 Member Data Documentation

10.125.2.1 unsigned int coco::islp_eval_type::child_n

children counter

Definition at line 1439 of file islp_evaluator.h.

10.125.2.2 std::vector<std::vector<interval> >* coco::islp_eval_type::islp_cache

slope cache

Definition at line 1434 of file islp_evaluator.h.

10.125.2.3 std::vector<std::vector<interval> >* coco::islp_eval_type::islp_data

partial slope data

Definition at line 1433 of file islp_evaluator.h.

10.125.2.4 std::vector<interval>* coco::islp_eval_type::islp_vec

result vector

Definition at line 1435 of file islp_evaluator.h.

10.125.2.5 const model* coco::islp_eval_type::mod

the DAG

Definition at line 1436 of file islp_evaluator.h.

10.125.2.6 interval coco::islp_eval_type::mult

the current value on the path

Definition at line 1437 of file islp_evaluator.h.

10.125.2.7 interval coco::islp_eval_type::mult_trans

transfer variable for mult

Definition at line 1438 of file islp_evaluator.h.

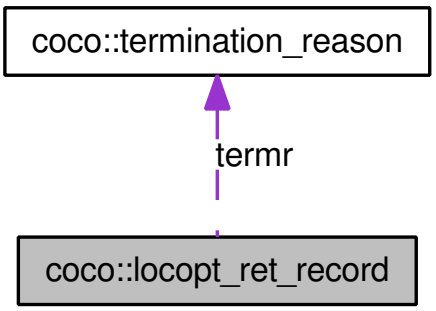
The documentation for this struct was generated from the following file:

- islp_evaluator.h

10.126 coco::locopt_ret_record Class Reference

`#include <locopt_ret.h>`

Collaboration diagram for coco::locopt_ret_record:



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

10.126.1 Detailed Description

Definition at line 85 of file locopt_ret.h.

10.126.2 Constructor & Destructor Documentation

10.126.2.1 coco::locopt_ret_record::locopt_ret_record () [inline]

Constructors

Definition at line 123 of file locopt_ret.h.

10.126.2.2 coco::locopt_ret_record::locopt_ret_record (const termination_reason & *t*) [inline]

Definition at line 126 of file locopt_ret.h.

10.126.2.3 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.

10.126.2.4 coco::locopt_ret_record::locopt_ret_record (const locopt_ret_record & *r*) [inline]

Copy constructor

Definition at line 133 of file locopt_ret.h.

10.126.3 Member Function Documentation

10.126.3.1 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.

10.126.3.2 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.

10.126.3.3 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.

10.126.3.4 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.

10.126.3.5 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:

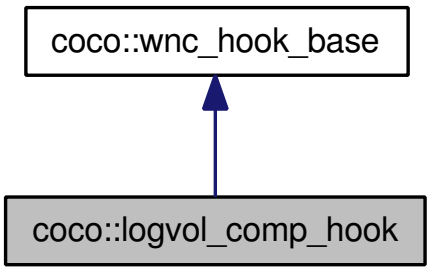
- locopt_ret.h

10.127 coco::logvol_comp_hook Class Reference

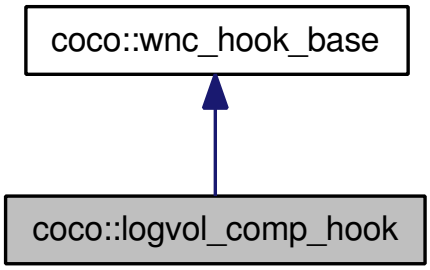
The log-volume computation hook (work node computation hook).

`#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

10.127.1 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".

Definition at line 42 of file logvol_hook.h.

10.127.2 Constructor & Destructor Documentation

10.127.2.1 coco::logvol_comp_hook::logvol_comp_hook () [inline]

Standard Constructor

Definition at line 46 of file logvol_hook.h.

10.127.2.2 virtual coco::logvol_comp_hook::~logvol_comp_hook () [inline, virtual]

Standard Destructor

Definition at line 49 of file logvol_hook.h.

10.127.3 Member Function Documentation

10.127.3.1 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.

10.127.3.2 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.

10.127.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *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 238 of file comp_hook.h.

10.127.3.4 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.

10.127.3.5 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.

10.127.3.6 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.

10.127.3.7 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.

10.127.3.8 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.

10.127.3.9 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.

10.127.3.10 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:

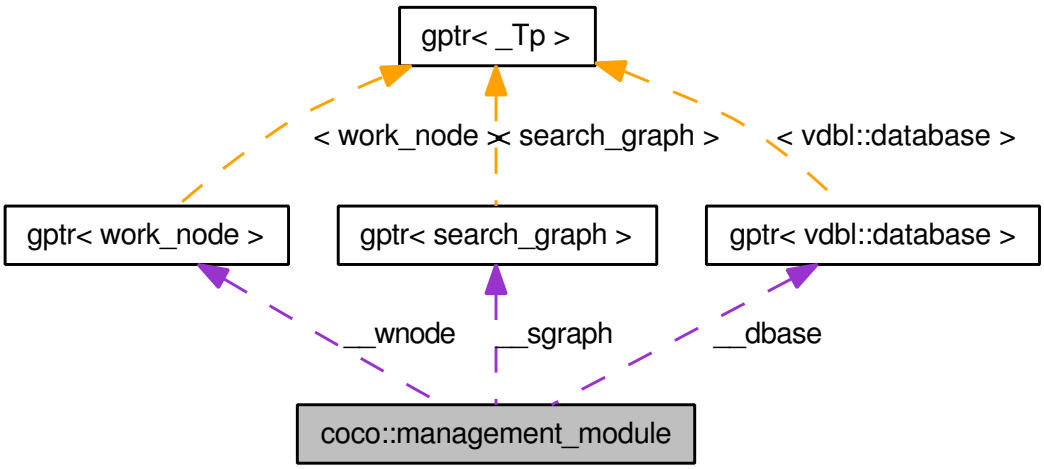
- logvol_hook.h

10.128 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

10.128.1 Detailed Description

This is the base class of all COCONUT management modules. A management module specializes in performing changes to the internal data structures.

Definition at line 73 of file mgmt_module.h.

10.128.2 Constructor & Destructor Documentation

10.128.2.1 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.

10.128.2.2 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.

10.128.2.3 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.

10.128.2.4 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.

10.128.2.5 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.

10.128.2.6 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.

10.128.2.7 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.

10.128.2.8 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.

10.128.2.9 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.

10.128.2.10 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.

10.128.2.11 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.

10.128.2.12 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.

10.128.2.13 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.

10.128.2.14 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.

10.128.2.15 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.

10.128.2.16 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.

10.128.2.17 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.

10.128.2.18 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.

10.128.2.19 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.

10.128.2.20 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.

10.128.2.21 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.

10.128.2.22 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.

10.128.2.23 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.

10.128.2.24 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.

10.128.2.25 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.

10.128.2.26 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.

10.128.2.27 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.

10.128.2.28 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.

10.128.2.29 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.

10.128.2.30 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.

10.128.2.31 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.

10.128.2.32 virtual coco::management_module::~management_module () [inline, virtual]

Standard Destructor

Definition at line 426 of file mgmt_module.h.

10.128.3 Member Function Documentation

10.128.3.1 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.

10.128.3.2 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.

10.128.3.3 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.

10.128.4 Member Data Documentation

10.128.4.1 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.

10.128.4.2 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.

10.128.4.3 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.

10.128.4.4 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.

10.128.4.5 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.

10.128.4.6 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:

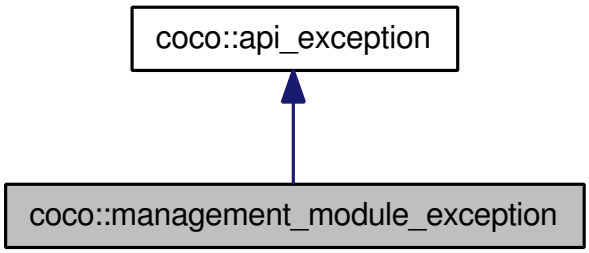
- mgmt_module.h

10.129 coco::management_module_exception Class Reference

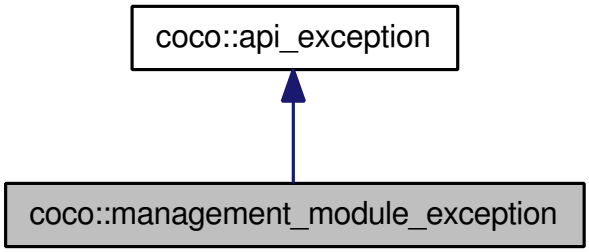
Management module exception class.

`#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 api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.129.1 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.

Definition at line 53 of file mgmt_module.h.

10.129.2 Constructor & Destructor Documentation

10.129.2.1 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.

10.129.2.2 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.

10.129.2.3 virtual coco::management_module_exception::~management_module_exception () throw () [inline, virtual]

Standard Destructor

Definition at line 64 of file mgmt_module.h.

10.129.3 Member Function Documentation

10.129.3.1 virtual std::string coco::api_exception::message () const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file api_exception.h.

10.129.3.2 virtual api_exception_type coco::api_exception::type () const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file api_exception.h.

10.129.3.3 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.

10.129.3.4 virtual char const* coco::api_exception::what () const throw () [inline, virtual, inherited]

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:

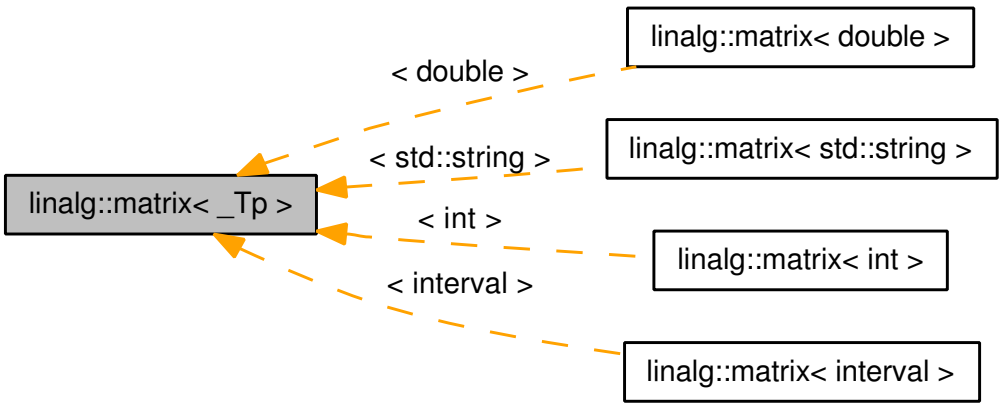
- mgmt_module.h

10.130 linalg::matrix< _Tp > Class Template Reference

Sparse Matrix Template Class (row major).

`#include <linalg.h>`

Inheritance diagram for linalg::matrix< _Tp >:



Public Types

- typedef mtl::matrix< _Tp, mtl::rectangle<>, mtl::compressed<>, mtl::row_major >::type _Base
- typedef _Base::OneD Row

Public Member Functions

- matrix ()
- matrix (const _Self &__m)
- matrix (const size_t &n, const size_t &m)
- void setOutputFormat (const linalg_output_format &f)
- const linalg_output_format & getOutputFormat () const
- _Self & operator= (const _Self &a)
- _Self & operator+= (const _Self &a)
- _Self & operator*= (_Tp a)
- bool operator== (const _Self &a) const
- bool operator!= (const _Self &a) const

Protected Attributes

- linalg_output_format output_format

Friends

- template<typename _Tq >
  std::ostream & std::operator<< (std::ostream &s, const matrix< _Tq > &A)

10.130.1 Detailed Description

template<class _Tp> class linalg::matrix< _Tp >

This class is the base class for all matrix storage/computation in the COCONUT environment. It is based on the MTL, but wraps the MTL classes to make changing the matrix library possible (and hopefully not too awkward. The matrix is general rectangular row-major.

Definition at line 247 of file linalg.h.

10.130.2 Member Typedef Documentation

10.130.2.1 template<class _Tp> typedef mtl::matrix<_Tp, mtl::rectangle<>, mtl::compressed<>, mtl::row_major>::type linalg::matrix< _Tp >::_Base

The base class

Definition at line 253 of file linalg.h.

10.130.2.2 template<class _Tp> typedef _Base::OneD linalg::matrix< _Tp >::Row

This is the row type of the matrix class.

Definition at line 255 of file linalg.h.

10.130.3 Constructor & Destructor Documentation

10.130.3.1 template<class _Tp> linalg::matrix< _Tp >::matrix () [inline]

The Standard Constructor

Definition at line 269 of file linalg.h.

10.130.3.2 template<class _Tp> linalg::matrix< _Tp >::matrix (const _Self & *__m*) [inline]

The Standard Copy Constructor

Definition at line 271 of file linalg.h.

10.130.3.3 template<class _Tp> linalg::matrix< _Tp >::matrix (const size_t & *n*, const size_t & *m*) [inline]

Constructor, which builds a matrix with `n` rows and `m` columns.

Definition at line 275 of file linalg.h.

10.130.4 Member Function Documentation

10.130.4.1 template<class _Tp> const linalg_output_format& linalg::matrix< _Tp >::getOutputFormat () const [inline]

This method returns the output format of the matrix.

Definition at line 284 of file linalg.h.

10.130.4.2 template<class _Tp> bool linalg::matrix< _Tp >::operator!= (const _Self & *a*) const

Standard comparison operator.

10.130.4.3 template<class _Tp> _Self& linalg::matrix< _Tp >::operator*= (_Tp *a*) [inline]

This operator multiplies this matrix by `a`.

Definition at line 324 of file linalg.h.

10.130.4.4 template<class _Tp> _Self& linalg::matrix< _Tp >::operator+= (const _Self & *a*) [inline]

This operator adds a to this matrix.

Definition at line 316 of file linalg.h.

10.130.4.5 template<class _Tp> _Self& linalg::matrix< _Tp >::operator= (const _Self & *a*) [inline]

Standard assignment operator.

Definition at line 307 of file linalg.h.

10.130.4.6 template<class _Tp> bool linalg::matrix< _Tp >::operator== (const _Self & *a*) const

Standard comparison operator.

10.130.4.7 template<class _Tp> void linalg::matrix< _Tp >::setOutputFormat (const linalg_output_format & *f*) [inline]

This method sets the output format of the matrix.

Definition at line 281 of file linalg.h.

10.130.5 Friends And Related Function Documentation

10.130.5.1 template<class _Tp> template<typename _Tq > std::ostream& std::operator<< (std::ostream & *s*, const matrix< _Tq > & *A*) [friend]

10.130.6 Member Data Documentation

10.130.6.1 template<class _Tp> linalg_output_format linalg::matrix< _Tp >::output_format [protected]

This variable describes the output format for matrix output. Its possible values are linalg_output_sparse and linalg_output_full.

Definition at line 265 of file linalg.h.

The documentation for this class was generated from the following file:

- linalg.h

10.131 coco::model Class Reference

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

`#include <model.h>`

Inheritance diagram for coco::model:



Collaboration diagram for coco::model:



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
- typedef **_Base::children_iterator children_iterator**
- typedef **_Base::parents_iterator parents_iterator**
- typedef **_Base::children_const_iterator children_const_iterator**
- typedef **_Base::parents_const_iterator parents_const_iterator**
- typedef **_Base::erased_part erased_part**
- typedef _SequenceCtr< void *, _PtrAlloc > **container_type**
- typedef **_DG_iterator**< _Tp, _Tp &, _Tp *, **container_type**, **children_iterator**, **children_const_-iterator** > **iterator**

- typedef **_DG_iterator**< _Tp, const _Tp &, const _Tp *, **container_type**, **children_iterator**, **children_const_iterator** > **const_iterator**
- typedef std::reverse_iterator< **const_iterator** > **const_reverse_iterator**
- typedef std::reverse_iterator< **iterator** > **reverse_iterator**
- typedef std::pair< **walker**, **walker** > **edge**
- typedef std::pair< **edge**, bool > **enhanced_edge**
- typedef _Tp **value_type**
- typedef _Node **node_type**
- typedef **value_type** * **pointer**
- typedef const **value_type** * **const_pointer**
- typedef **value_type** & **reference**
- typedef const **value_type** & **const_reference**
- typedef size_t **size_type**
- typedef ptrdiff_t **difference_type**

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_constraint_num ()
- void detach_gid ()
- void compress_numbers ()
- void renumber_variables ()
- void renumber_constraints ()
- void prepare_kill ()
- bool micro_simplify ()
- bool basic_simplify ()
- 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_constraint (const **walker** &_w)
- void free_node_num (unsigned int _nnum)
- void free_var_num (int _vnum)
- 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, linalg::sparse_vector< double > &coeffs, double &constant, const std::vector< interval > &_ranges)
- bool get_linear_coeffs (const **const_walker** &expr, linalg::sparse_vector< double > &coeffs, double &constant)
- bool **check_acyclicity** (const **walker** &__parent, const **walker** &__child)
- void **clear** ()
- **walker between** (const **walker** &__parent, const **children_iterator** &__cit, const **walker** &__child, const **parents_iterator** &__pit, const _Tp &__x)
- **walker between** (const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children, const _Tp &__x)
- **walker between** (const **walker** &__parent, const **children_iterator** &__cit, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker between** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const **parents_iterator** &__pit, const _Tp &__x)
- **walker split** (const **walker** &__parent, const **children_iterator** &__ch_it, const **walker** &__child, const **parents_iterator** &__pa_it, const _Tp &__x)
- void **split** (const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children, const _Tp &__x)
- **walker split** (const **walker** &__parent, const **children_iterator** &__ch_it, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker split** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const **parents_iterator** &__pr_it, const _Tp &__x)
- **walker between_back** (const **walker** &__parent, const **walker** &__child, const _Tp &__x)
- **walker between_back** (const **walker** &__parent, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker between_back** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const _Tp &__x)
- **walker split_back** (const **walker** &__parent, const **walker** &__child, const _Tp &__x)
- **walker split_back** (const **walker** &__parent, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker split_back** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const _Tp &__x)
- **walker between_front** (const **walker** &__parent, const **walker** &__child, const _Tp &__x)
- **walker between_front** (const **walker** &__parent, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker between_front** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const _Tp &__x)
- **walker split_front** (const **walker** &__parent, const **walker** &__child, const _Tp &__x)

- **walker split_front** (const **walker** &__parent, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker split_front** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const _Tp &__x)
- void **insert_subgraph** (_Self &__subgraph, const **walker** &__parent, const **children_iterator** &__ch_it, const **walker** &__child, const **parents_iterator** &__pa_it)
- void **insert_subgraph** (_Self &__subgraph, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **insert_subgraph** (_Self &__subgraph, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- void **insert_subgraph** (_Self &__subgraph, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **insert_subgraph** (_Self &__subgraph, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- void **insert_back_subgraph** (_Self &__subgraph, const **walker** &__parent, const **walker** &__child)
- void **insert_front_subgraph** (_Self &__subgraph, const **walker** &__parent, const **walker** &__child)
- void **add_edge** (const **walker** &__parent, const **children_iterator** &__ch_it, const **walker** &__child, const **parents_iterator** &__pa_it)
- void **add_edge** (const **edge** &__edge, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **add_edge** (const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **add_edge** (const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **add_edge_back** (const **walker** &__parent, const **walker** &__child)
- void **add_edge_front** (const **walker** &__parent, const **walker** &__child)
- **allocator_type get_allocator** () const
- **walker ground** ()
- **const_walker ground** () const
- **walker sky** ()
- **const_walker sky** () const
- **children_iterator root_begin** ()
- **children_const_iterator root_begin** () const
- **children_iterator root_end** ()
- **children_const_iterator root_end** () const
- **parents_iterator leaf_begin** ()
- **parents_const_iterator leaf_begin** () const
- **parents_iterator leaf_end** ()
- **parents_const_iterator leaf_end** () const
- bool **empty** () const
- **size_type size** () const
- **size_type max_size** () const
- void **swap** (_Self &__x)
- **walker insert_node_in_graph** (_Node *__n, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_node_in_graph** (_Node *__node, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_node_in_graph** (_Node *__node, const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)

- **walker insert_node_in_graph** (_Node *__node, const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_node_in_graph** (_Node *__n, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_node_in_graph** (_Node *__node, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_node_in_graph** (_Node *__node, const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_node_in_graph** (_Node *__node, const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_in_graph** (const _Tp &__x, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_in_graph** (const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_in_graph** (const _Tp &__x, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_in_graph** (const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_in_graph** (const _Tp &__x, const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_in_graph** (const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_in_graph** (const _Tp &__x, const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_in_graph** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_in_graph** (const _Tp &__x, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_in_graph** (const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_in_graph** (const _Tp &__x, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_in_graph** (const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_in_graph** (const _Tp &__x, const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_in_graph** (const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_in_graph** (const _Tp &__x, const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_in_graph** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- void **replace_edge_to_child** (const **walker** &__parent, const **walker** &__child_old, const **walker** &__child_new)
- void **replace_edge_to_child** (const **walker** &__parent, const **walker** &__child_old, const **walker** &__child_new)
- void **replace_edge_to_parent** (const **walker** &__parent_old, const **walker** &__parent_new, const **walker** &__child)
- void **replace_edge_to_parent** (const **walker** &__parent_old, const **walker** &__parent_new, const **walker** &__child)
- void **remove_edge** (const **edge** &__edge)
- void **remove_edge** (const **walker** &__parent, const **walker** &__child)

- void **remove_edge** (const **walker** &__parent, const **walker** &__child)
- void **remove_edge_and_deattach** (const **walker** &__parent, const **walker** &__child)
- void **remove_edge_and_deattach** (const **walker** &__parent, const **walker** &__child)
- void **sort_child_edges** (**walker** __position, **children_iterator** first, **children_iterator** last, Compare comp)
- void **sort_child_edges** (**walker** __position, Compare comp)
- void **sort_child_edges** (**walker** __position, **children_iterator** first, **children_iterator** last, Compare comp)
- void **sort_child_edges** (**walker** __position, Compare comp)
- void **sort_parent_edges** (**walker** __position, **parents_iterator** first, **parents_iterator** last, Compare comp)
- void **sort_parent_edges** (**walker** __position, Compare comp)
- void **sort_parent_edges** (**walker** __position, **parents_iterator** first, **parents_iterator** last, Compare comp)
- void **sort_parent_edges** (**walker** __position, Compare comp)
- **walker insert_node** (_Node *_node, const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (const _Tp &__x, const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (_Node *_node, const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (const _Tp &__x, const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node_before** (_Node *_node, const **walker** &__position, const container_insert_arg &__It)
- void **insert_node_before** (const _Tp &__x, const **walker** &__position, const container_insert_arg &__It)
- void **insert_node_before** (const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node_before** (_Node *_node, const **walker** &__position, const container_insert_arg &__It)
- void **insert_node_before** (const _Tp &__x, const **walker** &__position, const container_insert_arg &__It)
- void **insert_node_before** (const **walker** &__position, const container_insert_arg &__It)
- void **merge** (const **walker** &__position, const **walker** &__second, bool merge_parent_edges=true, bool merge_child_edges=true)
- void **merge** (const **walker** &__position, const **walker** &__second, bool merge_parent_edges=true, bool merge_child_edges=true)
- void **erase** (const **walker** &__position)
- void **erase** (const **walker** &__position)
- void **partial_erase_to_parent** (const **walker** &__position, const **walker** &__parent, unsigned int idx)
- void **partial_erase_to_parent** (const **walker** &__position, const **walker** &__parent, unsigned int idx)
- void **clear_erased_part** (**erased_part** &_ep)
- void **clear_erased_part** (**erased_part** &_ep)
- **erased_part erase_maximal_subgraph** (const **walker** &__position)
- **erased_part erase_maximal_subgraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_maximal_subgraph** (const **walker** &__position)
- **erased_part erase_maximal_subgraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_minimal_subgraph** (const **walker** &__position)

- **erased_part erase_minimal_subgraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_minimal_subgraph** (const **walker** &__position)
- **erased_part erase_minimal_subgraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_maximal_pregraph** (const **walker** &__position)
- **erased_part erase_maximal_pregraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_maximal_pregraph** (const **walker** &__position)
- **erased_part erase_maximal_pregraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_minimal_pregraph** (const **walker** &__position)
- **erased_part erase_minimal_pregraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_minimal_pregraph** (const **walker** &__position)
- **erased_part erase_minimal_pregraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- bool **erase_child** (const **walker** &__position, const **children_iterator** &__It)
- bool **erase_child** (const **walker** &__position, const **children_iterator** &__It)
- bool **erase_parent** (const **walker** &__position, const **parents_iterator** &__It)
- bool **erase_parent** (const **walker** &__position, const **parents_iterator** &__It)

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_constraints () const
- unsigned int number_of_managed_nodes () const
- unsigned int number_of_managed_variables () const
- unsigned int number_of_managed_constraints () const
- const **walker** & var (unsigned int i) const
- const **walker** & node (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 () const
- double obj_adj () const
- double obj_mult () 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_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_const_num (unsigned int node_num, unsigned int &const_num) const

Public Attributes

- linalg::matrix< double > lin
- std::vector< linalg::matrix< double > > matd
- std::vector< linalg::matrix< interval > > mati
- int ocoeff
- **walker** objective
- std::vector< **walker** > constraints
- bool has_read_errors

Protected Types

- typedef **_Base::allocator_type allocator_type**

Protected Member Functions

- _Node * **_C_create_node** (const _Tp &__x)
- _Node * **_C_create_node** ()
- void **_C_destroy_node** (_Node *__p)
- void **clear_graph** (**_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_node)

- void **clear_graph** (**_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_node)
- **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > * **_C_get_node** ()
- void **_C_put_node** (**_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *__p)
- void **_C_put_node** (**_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *__p)
- void **clear_children** ()
- void **clear_parents** ()
- void **add_all_children** (_Output_Iterator fi, **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_parent)
- void **add_all_children** (_Output_Iterator fi, **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_parent)
- void **add_all_parents** (_Output_Iterator fi, **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_child)
- void **add_all_parents** (_Output_Iterator fi, **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_child)

Protected Attributes

- **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > * **_C_ground**
- **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > * **_C_sky**
- int **_C_mark**

Friends

- class dag_delta
- class dag_undelta

10.131.1 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.

Definition at line 55 of file model.h.

10.131.2 Member Typedef Documentation

10.131.2.1 typedef _Base::allocator_type dag< class, _SequenceCtr, class, class >::allocator_type [protected, inherited]

Reimplemented from **dgraph**< **_Tp, _SequenceCtr, _PtrAlloc, _Alloc** >.

10.131.2.2 typedef _Base::children_const_iterator dag< class, _SequenceCtr, class, class >::children_const_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.3 typedef _Base::children_iterator dag< class, _SequenceCtr, class, class >::children_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.4 typedef _DG_iterator< _Tp, const _Tp &, const _Tp *, container_type, children_iterator, children_const_iterator > dag< class, _SequenceCtr, class, class >::const_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.5 typedef const value_type * dag< class, _SequenceCtr, class, class >::const_pointer [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.6 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.

10.131.2.7 typedef const value_type & dag< class, _SequenceCtr, class, class >::const_reference [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.8 typedef std::reverse_iterator< const_iterator > dag< class, _SequenceCtr, class, class >::const_reverse_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.9 typedef dag<expression_node>::const_walker coco::model::const_walker

Const walker (generalized const pointer) to the DAG.

Reimplemented from **dag< expression_node >**.

Definition at line 65 of file model.h.

10.131.2.10 typedef _SequenceCtr< void *, _PtrAlloc > dag< class, _SequenceCtr, class, class >::container_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.11 typedef ptrdiff_t dag< class, _SequenceCtr, class, class >::difference_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.12 typedef std::pair< walker, walker > dag< class, _SequenceCtr, class, class >::edge [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.13 typedef std::pair< edge, bool > dag< class, _SequenceCtr, class, class >::enhanced_edge [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.14 typedef _Base::erased_part dag< class, _SequenceCtr, class, class >::erased_part [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.15 typedef _DG_iterator< _Tp, _Tp &, _Tp *, container_type, children_iterator, children_const_iterator > dag< class, _SequenceCtr, class, class >::iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.16 typedef _Node dag< class, _SequenceCtr, class, class >::node_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.17 typedef _Base::parents_const_iterator dag< class, _SequenceCtr, class, class >::parents_const_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.18 typedef _Base::parents_iterator dag< class, _SequenceCtr, class, class >::parents_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.19 typedef value_type * dag< class, _SequenceCtr, class, class >::pointer [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.20 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.

10.131.2.21 typedef value_type & dag< class, _SequenceCtr, class, class >::reference [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.22 typedef std::reverse_iterator< iterator > dag< class, _SequenceCtr, class, class >::reverse_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.23 typedef size_t dag< class, _SequenceCtr, class, class >::size_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.24 typedef _Tp dag< class, _SequenceCtr, class, class >::value_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.2.25 typedef dag<expression_node>::walker coco::model::walker

Walker (generalized pointer) to the DAG.

Reimplemented from **dag< expression_node >**.

Definition at line 63 of file model.h.

10.131.3 Constructor & Destructor Documentation

10.131.3.1 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 520 of file model.hpp.

10.131.3.2 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 196 of file model.cc.

10.131.3.3 coco::model::model (int *__num_of_vars*) [inline]

This constructor generates a new model prepared for __num_of_vars variables.

Definition at line 513 of file model.hpp.

10.131.3.4 coco::model::model (const model & *__m*) [inline]

Standard Copy Constructor, which generates a new model_gid.

Definition at line 542 of file model.hpp.

10.131.3.5 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 558 of file model.hpp.

10.131.3.6 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 433 of file model.cc.

10.131.3.7 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 443 of file model.cc.

10.131.3.8 coco::model::~model () [inline]

Standard Destructor

Definition at line 574 of file model.hpp.

10.131.4 Member Function Documentation

10.131.4.1 _Node * dag< class, _SequenceCtr, class, class >::_C_create_node () [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.2 _Node * dag< class, _SequenceCtr, class, class >::_C_create_node (const _Tp & __x) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.3 void dag< class, _SequenceCtr, class, class >::_C_destroy_node (_Node * __p) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.4 _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * dag< class, _SequenceCtr, class, class >::_C_get_node () [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.5 void _DG_base< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _Alloc >::_C_put_node (_DG_node< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator > * __p) [protected, inherited]

10.131.4.6 void dag< class, _SequenceCtr, class, class >::_C_put_node (_DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * __p) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.7 void _DG_base< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _Alloc >::add_all_children (_Output_Iterator *fi*, _DG_node< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator > * *_parent*) [protected, inherited]

10.131.4.8 void dag< class, _SequenceCtr, class, class >::add_all_children (_Output_Iterator *fi*, _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * *_parent*) [protected, inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.9 void _DG_base< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _Alloc >::add_all_parents (_Output_Iterator *fi*, _DG_node< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator > * *_child*) [protected, inherited]

10.131.4.10 void dag< class, _SequenceCtr, class, class >::add_all_parents (_Output_Iterator *fi*, _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * *_child*) [protected, inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.11 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::add_edge (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

10.131.4.12 void dag< class, _SequenceCtr, class, class >::add_edge (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.13 void dag< class, _SequenceCtr, class, class >::add_edge (const edge & *__edge*, const container_insert_arg & *__itc*, const container_insert_arg & *__itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.14 void dag< class, _SequenceCtr, class, class >::add_edge (const walker & *__parent*, const children_iterator & *__ch_it*, const walker & *__child*, const parents_iterator & *__pa_it*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.15 void dag< class, _SequenceCtr, class, class >::add_edge_back (const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.16 void dag< class, _SequenceCtr, class, class >::add_edge_front (const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.17 void coco::model::arrange_constraints () [inline]

A call to arrange_constraints sorts the constraints according to their complexity.

Definition at line 1518 of file model.hpp.

10.131.4.18 bool coco::model::basic_simplify ()

This method calls the full model simplifier.

Definition at line 3730 of file model.cc.

10.131.4.19 walker dag< class, _SequenceCtr, class, class >::between (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const parents_iterator & *__pit*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.20 walker dag< class, _SequenceCtr, class, class >::between (const walker & *__parent*, const children_iterator & *__cit*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.21 walker dag< class, _SequenceCtr, class, class >::between (const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.22 walker dag< class, _SequenceCtr, class, class >::between (const walker & *__parent*, const children_iterator & *__cit*, const walker & *__child*, const parents_iterator & *__pit*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.23 walker dag< class, _SequenceCtr, class, class >::between_back (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.24 walker dag< class, _SequenceCtr, class, class >::between_back (const walker & *__parent*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.25 walker dag< class, _SequenceCtr, class, class >::between_back (const walker & *__parent*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.26 walker dag< class, _SequenceCtr, class, class >::between_front (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.27 walker dag< class, _SequenceCtr, class, class >::between_front (const walker & *__parent*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.28 walker dag< class, _SequenceCtr, class, class >::between_front (const walker & *__parent*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.29 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 1834 of file model.hpp.

10.131.4.30 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 1814 of file model.hpp.

10.131.4.31 bool dag< class, _SequenceCtr, class, class >::check_acyclicity (const walker & *__parent*, const walker & *__child*) [inherited]

10.131.4.32 void coco::model::check_counters (bool *reset*) const

Check consistency of the number of parents and children of all nodes.

Definition at line 4017 of file model.cc.

10.131.4.33 void dag< class, _SequenceCtr, class, class >::clear () [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.34 void dag< class, _SequenceCtr, class, class >::clear_children () [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.35 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::clear_erased_part (erased_part & *_ep*) [inherited]

10.131.4.36 void dag< class, _SequenceCtr, class, class >::clear_erased_part (erased_part & *_ep*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.37 void _DG_base< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _Alloc >::clear_graph (_DG_node< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator > * *_node*) [protected, inherited]

10.131.4.38 void dag< class, _SequenceCtr, class, class >::clear_graph (_DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * *_node*) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.39 void dag< class, _SequenceCtr, class, class >::clear_parents () [protected, inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.40 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 1935 of file model.hpp.

10.131.4.41 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 1533 of file model.hpp.

10.131.4.42 const std::string coco::model::const_name (unsigned int *n*) const [inline]

This method returns the name of the constraint `n`.

Definition at line 625 of file model.hpp.

10.131.4.43 model::walker coco::model::constant (const std::vector< double > & *_constant*) [inline]

This method constructs a new vector constant with value `_constant`.

Definition at line 1733 of file model.hpp.

10.131.4.44 model::walker coco::model::constant (double *_constant*) [inline]

This method constructs a new scalar constant with value `_constant`.

Definition at line 1722 of file model.hpp.

10.131.4.45 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 660 of file model.hpp.

10.131.4.46 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 511 of file model.hpp.

10.131.4.47 void coco::model::detect_0chain ()

This method identifies all 0-chains (univariate functions) in the DAG.

Definition at line 3980 of file model.cc.

10.131.4.48 bool dag< class, _SequenceCtr, class, class >::empty () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.49 model::walker coco::model::empty_reference () const [inline]

This method returns an empty node reference.

Definition at line 508 of file model.hpp.

10.131.4.50 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase (const walker & *__position*) [inherited]

10.131.4.51 void dag< class, _SequenceCtr, class, class >::erase (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.52 bool __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_child (const walker & *__position*, const children_iterator & *__It*) [inherited]

10.131.4.53 bool dag< class, _SequenceCtr, class, class >::erase_child (const walker & *__position*, const children_iterator & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.54 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_maximal_pregraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

10.131.4.55 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_maximal_pregraph (const walker & *__position*) [inherited]

10.131.4.56 erased_part dag< class, _SequenceCtr, class, class >::erase_maximal_pregraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.57 erased_part dag< class, _SequenceCtr, class, class >::erase_maximal_pregraph (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.58 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_maximal_subgraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

10.131.4.59 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_maximal_subgraph (const walker & *__position*) [inherited]

10.131.4.60 erased_part dag< class, _SequenceCtr, class, class >::erase_maximal_subgraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.61 erased_part dag< class, _SequenceCtr, class, class >::erase_maximal_subgraph (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.62 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_minimal_pregraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

10.131.4.63 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_minimal_pregraph (const walker & *__position*) [inherited]

10.131.4.64 erased_part dag< class, _SequenceCtr, class, class >::erase_minimal_pregraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.65 erased_part dag< class, _SequenceCtr, class, class >::erase_minimal_pregraph (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.66 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_minimal_subgraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

10.131.4.67 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_minimal_subgraph (const walker & *__position*) [inherited]

10.131.4.68 erased_part dag< class, _SequenceCtr, class, class >::erase_minimal_subgraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.69 erased_part dag< class, _SequenceCtr, class, class >::erase_minimal_subgraph (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.70 bool __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_parent (const walker & *__position*, const parents_iterator & *__It*) [inherited]

10.131.4.71 bool dag< class, _SequenceCtr, class, class >::erase_parent (const walker & *__position*, const parents_iterator & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.72 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 631 of file model.hpp.

10.131.4.73 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 1622 of file model.hpp.

10.131.4.74 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 1694 of file model.hpp.

10.131.4.75 void coco::model::free_node_num (unsigned int *_nnum*) [inline]

This method removes the node _nnum from all model data structures.

Definition at line 1670 of file model.hpp.

10.131.4.76 void coco::model::free_nodes_on_destroy () [inline]

This method marks tells the model to call free_all during model destruction.

Definition at line 600 of file model.hpp.

10.131.4.77 void coco::model::free_var_num (int *_vnum*) [inline]

This method removes the variable _vnum from all model data structures.

Definition at line 1682 of file model.hpp.

10.131.4.78 allocator_type dag< class, _SequenceCtr, class, class >::get_allocator () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.79 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 596 of file model.hpp.

10.131.4.80 bool coco::model::get_linear_coeffs (const const_walker & *expr*, linalg::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 2807 of file model.hpp.

10.131.4.81 bool coco::model::get_linear_coeffs (const const_walker & *expr*, linalg::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 2796 of file model.hpp.

10.131.4.82 model::walker coco::model::ghost (unsigned int *_nnum*) [inline]

This method constructs a new ghost with node number `_nnum`.

Definition at line 1744 of file model.hpp.

10.131.4.83 const_ref_iterator coco::model::ghost_begin () const [inline]

Get a const iterator to the first ghost.

Definition at line 363 of file model.h.

10.131.4.84 ref_iterator coco::model::ghost_begin () [inline]

Get an iterator to the first ghost.

Definition at line 361 of file model.h.

10.131.4.85 const_ref_iterator coco::model::ghost_end () const [inline]

Get a const iterator beyond the last ghost.

Definition at line 367 of file model.h.

10.131.4.86 ref_iterator coco::model::ghost_end () [inline]

Get an iterator beyond the last ghost.

Definition at line 365 of file model.h.

10.131.4.87 model_gid* coco::model::gid_data () const [inline]

This method returns the model_gid structure this model belongs to.

Definition at line 199 of file model.h.

10.131.4.88 const_walker dag< class, _SequenceCtr, class, class >::ground () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.89 walker dag< class, _SequenceCtr, class, class >::ground () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.90 void dag< class, _SequenceCtr, class, class >::insert_back_subgraph (_Self & *__subgraph*, const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.91 void dag< class, _SequenceCtr, class, class >::insert_front_subgraph (_Self & *__subgraph*, const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.92 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

10.131.4.93 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

10.131.4.94 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inherited]

10.131.4.95 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const _Tp & *__x*, const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inherited]

10.131.4.96 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

10.131.4.97 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

10.131.4.98 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

10.131.4.99 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const _Tp & *__x*, const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

10.131.4.100 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.101 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.102 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.103 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const _Tp & *__x*, const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.104 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.105 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.106 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.107 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const _Tp & *__x*, const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.108 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node (const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.131.4.109 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node (const _Tp & *__x*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.131.4.110 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node (_Node * *_node*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.131.4.111 walker dag< class, _SequenceCtr, class, class >::insert_node (const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.112 walker dag< class, _SequenceCtr, class, class >::insert_node (const _Tp & *__x*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.113 walker dag< class, _SequenceCtr, class, class >::insert_node (_Node * *_node*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.114 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_before (const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.131.4.115 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_before (const _Tp & *__x*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.131.4.116 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_before (_Node * *_node*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.131.4.117 void dag< class, _SequenceCtr, class, class >::insert_node_before (const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.118 void dag< class, _SequenceCtr, class, class >::insert_node_before (const _Tp & *__x*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.119 walker dag< class, _SequenceCtr, class, class >::insert_node_before (_Node * *_node*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.120 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_in_graph (_Node * *__node*, const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

10.131.4.121 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_in_graph (_Node * __node, const walker & __parent, const container_insert_arg & __pref, const __SequenceCtr< walker, _Allocator > & __children) [inherited]

10.131.4.122 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_in_graph (_Node * __node, const __SequenceCtr1< walker, _Allocator1 > & __parents, const __SequenceCtr2< walker, _Allocator2 > & __children) [inherited]

10.131.4.123 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_in_graph (_Node * __n, const walker & __parent, const walker & __child, const container_insert_arg & __Itc, const container_insert_arg & __Itp) [inherited]

10.131.4.124 walker dag< class, _SequenceCtr, class, class >::insert_node_in_graph (_Node * __node, const __SequenceCtr< walker, _Allocator > & __parents, const walker & __child, const container_insert_arg & __cref) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.125 walker dag< class, _SequenceCtr, class, class >::insert_node_in_graph (_Node * __node, const walker & __parent, const container_insert_arg & __pref, const __SequenceCtr< walker, _Allocator > & __children) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.126 walker dag< class, _SequenceCtr, class, class >::insert_node_in_graph (_Node * __node, const __SequenceCtr1< walker, _Allocator1 > & __parents, const __SequenceCtr2< walker, _Allocator2 > & __children) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.127 walker dag< class, _SequenceCtr, class, class >::insert_node_in_graph (_Node * __n, const walker & __parent, const walker & __child, const container_insert_arg & __Itc, const container_insert_arg & __Itp) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.128 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_subgraph (_Self & __subgraph, const __SequenceCtr1< walker, _Allocator1 > & __parents, const __SequenceCtr2< walker, _Allocator2 > & __children) [inherited]

10.131.4.129 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_subgraph (_Self & __subgraph, const walker & __parent, const walker & __child, const container_insert_arg & __Itc, const container_insert_arg & __Itp) [inherited]

10.131.4.130 void dag< class, _SequenceCtr, class, class >::insert_subgraph (_Self & __subgraph, const __SequenceCtr1< walker, _Allocator1 > & __parents, const __SequenceCtr2< walker, _Allocator2 > & __children) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.131 void dag< class, _SequenceCtr, class, class >::insert_subgraph (_Self & __subgraph, const walker & __parent, const walker & __child, const container_insert_arg & __Itc, const container_insert_arg & __Itp) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.132 void dag< class, _SequenceCtr, class, class >::insert_subgraph (_Self & __subgraph, const walker & __parent, const children_iterator & __ch_it, const walker & __child, const parents_iterator & __pa_it) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.133 bool coco::model::is_empty (const walker & _w) const [inline]

Check whether this node is empty (unused).

Definition at line 507 of file model.hpp.

10.131.4.134 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 611 of file model.hpp.

10.131.4.135 parents_const_iterator dag< class, _SequenceCtr, class, class >::leaf_begin () const [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.136 parents_iterator dag< class, _SequenceCtr, class, class >::leaf_begin () [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.137 parents_const_iterator dag< class, _SequenceCtr, class, class >::leaf_end () const [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.138 parents_iterator dag< class, _SequenceCtr, class, class >::leaf_end () [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.139 size_type dag< class, _SequenceCtr, class, class >::max_size () const [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.140 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::merge (const walker & *__position*, const walker & *__second*, bool *merge_parent_edges* = `true`, bool *merge_child_edges* = `true`) [inherited]

10.131.4.141 void dag< class, _SequenceCtr, class, class >::merge (const walker & *__position*, const walker & *__second*, bool *merge_parent_edges* = `true`, bool *merge_child_edges* = `true`) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.142 bool coco::model::micro_simplify ()

This method calls the micro simplifier which only initializes the most important data and semantics information.

Definition at line 3174 of file model.cc.

10.131.4.143 const std::string coco::model::model_name () const [inline]

This method returns the name of the model.

Definition at line 622 of file model.hpp.

10.131.4.144 size_t coco::model::n_fixed_vars () const [inline]

This method returns the number of variables fixed by the preprocessor.

Definition at line 630 of file model.hpp.

10.131.4.145 size_t coco::model::n_unused_constrs () const [inline]

This method returns the number of unused constraints.

Definition at line 636 of file model.hpp.

10.131.4.146 size_t coco::model::n_unused_vars () const [inline]

This method returns the number of unused variables.

Definition at line 633 of file model.hpp.

10.131.4.147 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 1895 of file model.hpp.

10.131.4.148 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 1876 of file model.hpp.

10.131.4.149 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 1717 of file model.hpp.

10.131.4.150 int coco::model::next_constraint_num () [inline]

This method returns the next free constraint number.

Definition at line 1525 of file model.hpp.

10.131.4.151 int coco::model::next_num () [inline]

This method returns the next free node number.

Definition at line 1523 of file model.hpp.

10.131.4.152 int coco::model::next_variable_num () [inline]

This method returns the next free variable number.

Definition at line 1524 of file model.hpp.

10.131.4.153 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 654 of file model.hpp.

10.131.4.154 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 648 of file model.hpp.

10.131.4.155 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 190 of file model.h.

10.131.4.156 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 182 of file model.h.

10.131.4.157 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 186 of file model.h.

10.131.4.158 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 651 of file model.hpp.

10.131.4.159 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 645 of file model.hpp.

10.131.4.160 double coco::model::obj_adj () const [inline]

This method returns the additive constant in the objective function.

Definition at line 628 of file model.hpp.

10.131.4.161 double coco::model::obj_mult () const [inline]

This method returns the scalar multiplier of the objective function.

Definition at line 629 of file model.hpp.

10.131.4.162 const std::string coco::model::obj_name () const [inline]

This method returns the name of the objective function.

Definition at line 627 of file model.hpp.

10.131.4.163 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 642 of file model.hpp.

10.131.4.164 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::partial_erase_to_parent (const walker & *__position*, const walker & *__parent*, unsigned int *idx*) [inherited]

10.131.4.165 void dag< class, _SequenceCtr, class, class >::partial_erase_to_parent (const walker & *__position*, const walker & *__parent*, unsigned int *idx*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.166 void coco::model::prepare_kill () [inline]

This method takes care that upon deletion everything goes away.

Definition at line 327 of file model.h.

10.131.4.167 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::remove_edge (const walker & *__parent*, const walker & *__child*) [inherited]

10.131.4.168 void dag< class, _SequenceCtr, class, class >::remove_edge (const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.169 void dag< class, _SequenceCtr, class, class >::remove_edge (const edge & *__edge*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.170 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::remove_edge_and_deattach (const walker & *__parent*, const walker & *__child*) [inherited]

10.131.4.171 void dag< class, _SequenceCtr, class, class >::remove_edge_and_deattach (const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.172 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 1712 of file model.hpp.

10.131.4.173 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 1707 of file model.hpp.

10.131.4.174 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 243 of file model.cc.

10.131.4.175 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.

10.131.4.176 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 1527 of file model.hpp.

10.131.4.177 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::replace_edge_to_child (const walker & *__parent*, const walker & *__child_old*, const walker & *__child_new*) [inherited]

10.131.4.178 void dag< class, _SequenceCtr, class, class >::replace_edge_to_child (const walker & *__parent*, const walker & *__child_old*, const walker & *__child_new*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.179 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::replace_edge_to_parent (const walker & *__parent_old*, const walker & *__parent_new*, const walker & *__child*) [inherited]

10.131.4.180 void dag< class, _SequenceCtr, class, class >::replace_edge_to_parent (const walker & *__parent_old*, const walker & *__parent_new*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.181 children_const_iterator dag< class, _SequenceCtr, class, class >::root_begin () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.182 children_iterator dag< class, _SequenceCtr, class, class >::root_begin () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.183 children_const_iterator dag< class, _SequenceCtr, class, class >::root_end () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.184 children_iterator dag< class, _SequenceCtr, class, class >::root_end () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.185 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 1941 of file model.hpp.

10.131.4.186 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 1513 of file model.hpp.

10.131.4.187 size_type dag< class, _SequenceCtr, class, class >::size () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.188 const_walker dag< class, _SequenceCtr, class, class >::sky () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.189 walker dag< class, _SequenceCtr, class, class >::sky () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.190 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::sort_child_edges (walker *__position*, Compare *comp*) [inherited]

10.131.4.191 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::sort_child_edges (walker *__position*, children_iterator *first*, children_iterator *last*, Compare *comp*) [inherited]

10.131.4.192 void dag< class, _SequenceCtr, class, class >::sort_child_edges (walker *__position*, Compare *comp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.193 void dag< class, _SequenceCtr, class, class >::sort_child_edges (walker *__position*, children_iterator *first*, children_iterator *last*, Compare *comp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.194 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::sort_parent_edges (walker *__position*, Compare *comp*) [inherited]

10.131.4.195 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::sort_parent_edges (walker *__position*, parents_iterator *first*, parents_iterator *last*, Compare *comp*) [inherited]

10.131.4.196 void dag< class, _SequenceCtr, class, class >::sort_parent_edges (walker *__position*, Compare *comp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.197 void dag< class, _SequenceCtr, class, class >::sort_parent_edges (walker *__position*, parents_iterator *first*, parents_iterator *last*, Compare *comp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.198 walker dag< class, _SequenceCtr, class, class >::split (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const parents_iterator & *__pr_it*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.199 walker dag< class, _SequenceCtr, class, class >::split (const walker & *__parent*, const children_iterator & *__ch_it*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.200 void dag< class, _SequenceCtr, class, class >::split (const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.201 walker dag< class, _SequenceCtr, class, class >::split (const walker & *__parent*, const children_iterator & *__ch_it*, const walker & *__child*, const parents_iterator & *__pa_it*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.202 walker dag< class, _SequenceCtr, class, class >::split_back (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.203 walker dag< class, _SequenceCtr, class, class >::split_back (const walker & *__parent*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.204 walker dag< class, _SequenceCtr, class, class >::split_back (const walker & *__parent*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.205 walker dag< class, _SequenceCtr, class, class >::split_front (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.206 walker dag< class, _SequenceCtr, class, class >::split_front (const walker & *__parent*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.4.207 walker dag< class, _SequenceCtr, class, class >::split_front (const walker & *__parent*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.208 model::walker coco::model::store_constraint (const walker & *_w*) [inline]

This method stores a new constraint in all model data structures

Definition at line 1595 of file model.hpp.

10.131.4.209 model::walker coco::model::store_ghost (const walker & *_w*) [inline]

This method stores a new ghost in all model data structures

Definition at line 1607 of file model.hpp.

10.131.4.210 model::walker coco::model::store_node (const walker & *_w*) [inline]

This method stores a new node in all model data structures

Definition at line 1563 of file model.hpp.

10.131.4.211 model::walker coco::model::store_variable (const walker & *_w*) [inline]

This method stores a new variable in all model data structures

Definition at line 1579 of file model.hpp.

10.131.4.212 void dag< class, _SequenceCtr, class, class >::swap (_Self & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.131.4.213 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 640 of file model.hpp.

10.131.4.214 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 1865 of file model.hpp.

10.131.4.215 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 1851 of file model.hpp.

10.131.4.216 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 638 of file model.hpp.

10.131.4.217 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 634 of file model.hpp.

10.131.4.218 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 657 of file model.hpp.

10.131.4.219 const std::string coco::model::var_name (unsigned int *n*) const [inline]

This method returns the name of the variable n.

Definition at line 623 of file model.hpp.

10.131.4.220 model::walker coco::model::variable (unsigned int *_vnum*)

This method constructs a new variable with variable number _vnum.

Definition at line 3881 of file model.cc.

10.131.4.221 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 3925 of file model.cc.

10.131.4.222 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 3950 of file model.cc.

10.131.4.223 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 3798 of file model.cc.

10.131.5 Friends And Related Function Documentation

10.131.5.1 friend class dag_delta [friend]

Definition at line 536 of file model.h.

10.131.5.2 friend class dag_undelta [friend]

Definition at line 537 of file model.h.

10.131.6 Member Data Documentation

10.131.6.1 _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * dag< class, _SequenceCtr, class, class >::_C_ground [protected, inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.6.2 int dag< class, _SequenceCtr, class, class >::_C_mark [protected, inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.6.3 _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * dag< class, _SequenceCtr, class, class >::_C_sky [protected, inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.131.6.4 std::vector<walker> coco::model::constraints

This is a vector of walkers pointing to the result nodes of the constraints.

Definition at line 112 of file model.h.

10.131.6.5 bool coco::model::has_read_errors

This member is set to if an I/O exception was raised during model reading.

Definition at line 115 of file model.h.

10.131.6.6 linalg::matrix<double> coco::model::lin

The `lin` member is the matrix of linear constraints.

Definition at line 93 of file model.h.

10.131.6.7 std::vector<linalg::matrix<double> > coco::model::matd

The `matd` member is a vector of double matrices.

Definition at line 95 of file model.h.

10.131.6.8 std::vector<linalg::matrix<interval> > coco::model::mati

The `matd` member is a vector of interval matrices.

Definition at line 97 of file model.h.

10.131.6.9 walker coco::model::objective

This is a walker pointing to the result node of the objective function.

Definition at line 109 of file model.h.

10.131.6.10 int coco::model::ocoeff

This objective coefficient is used to transfer any constraint satisfaction problem or optimization problem to a minimization problem. The values are

1	for a minimization problem
-1	for a maximization problem
0	for a constraint satisfaction problem

Definition at line 107 of file model.h.

The documentation for this class was generated from the following files:

- model.h
- model.cc
- model.hpp

10.132 coco::model::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

10.132.1 Detailed Description

This function class is used to compare nodes for sorting and binary search.

Definition at line 1541 of file model.hpp.

10.132.2 Member Function Documentation

10.132.2.1 bool coco::model::model::__docompare_nodes::operator() (const model::walker & *_a*, const model::walker & *_b*) const [inline]

Definition at line 1547 of file model.hpp.

10.132.2.2 bool coco::model::model::__docompare_nodes::operator() (unsigned int *_a*, const model::walker & *_b*) const [inline]

Definition at line 1545 of file model.hpp.

10.132.2.3 bool coco::model::model::__docompare_nodes::operator() (const model::walker & *_a*, unsigned int *_b*) const [inline]

Definition at line 1543 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

10.133 coco::model::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

10.133.1 Detailed Description

This function class is used to compare variables for sorting and binary search.

Definition at line 1553 of file model.hpp.

10.133.2 Member Function Documentation

10.133.2.1 bool coco::model::model::__docompare_variables::operator() (const model::walker & *_a*, const model::walker & *_b*) const [inline]

Definition at line 1559 of file model.hpp.

10.133.2.2 bool coco::model::model::__docompare_variables::operator() (unsigned int *_a*, const model::walker & *_b*) const [inline]

Definition at line 1557 of file model.hpp.

10.133.2.3 bool coco::model::model::__docompare_variables::operator() (const model::walker & *_a*, unsigned int *_b*) const [inline]

Definition at line 1555 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

10.134 coco::model::model::detect_0chain_visitor Class Reference

`#include <model.hpp>`

Inheritance diagram for coco::model::model::detect_0chain_visitor:



Collaboration diagram for coco::model::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
- typedef model::detect_0chain_visitor_st data_type

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 void cleanup (const node_data_type &__data)
- virtual **const_walker** short_cut_to (const node_data_type &__data)

  - 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)

Protected Attributes

- const variable_indicator * v_ind
- model::detect_0chain_visitor_st eval_data

10.134.1 Detailed Description

The detect_0chain_visitor is used to find 0-chains, i.e. univariate functions within the DAG.

Definition at line 1984 of file model.hpp.

10.134.2 Member Typedef Documentation

10.134.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 723 of file evaluator.h.

10.134.2.2 typedef model::detect_0chain_visitor_st coco::_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.134.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 719 of file evaluator.h.

10.134.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 721 of file evaluator.h.

10.134.3 Constructor & Destructor Documentation

10.134.3.1 coco::model::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 1998 of file model.hpp.

10.134.3.2 coco::model::model::detect_0chain_visitor::detect_0chain_visitor (const detect_0chain_visitor & *__x*) [inline]

Standard Copy Constructor

Definition at line 2011 of file model.hpp.

10.134.3.3 coco::model::model::detect_0chain_visitor::~detect_0chain_visitor () [inline]

Standard Destructor

Definition at line 2015 of file model.hpp.

10.134.4 Member Function Documentation

10.134.4.1 void coco::model::model::detect_0chain_visitor::calculate (const expression_node & *__data*) [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::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2047 of file model.hpp.

10.134.4.2 std::pair<unsigned int,unsigned int> coco::model::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::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2076 of file model.hpp.

10.134.4.3 virtual void coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::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 804 of file evaluator.h.

10.134.4.4 int coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 750 of file evaluator.h.

10.134.4.5 int coco::model::model::detect_0chain_visitor::initialize (const expression_node & *__data*) [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::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2029 of file model.hpp.

10.134.4.6 void coco::model::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::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2020 of file model.hpp.

10.134.4.7 bool coco::model::model::detect_0chain_visitor::is_cached (const expression_node & *__data*) [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::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2022 of file model.hpp.

10.134.4.8 void coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 744 of file evaluator.h.

10.134.4.9 int coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 732 of file evaluator.h.

10.134.4.10 void coco::model::model::detect_0chain_visitor::retrieve_from_cache (const expression_node & *__data*) [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::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2025 of file model.hpp.

10.134.4.11 virtual const_walker coco::cached_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::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 450 of file evaluator.h.

10.134.4.12 int coco::model::model::detect_0chain_visitor::update (const expression_node & *__data*, const std::pair< unsigned int, unsigned int > & *__rval*) [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::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2055 of file model.hpp.

10.134.4.13 int coco::model::model::detect_0chain_visitor::update (const std::pair< unsigned int, unsigned int > & *__rval*) [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::cached_forward_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >, model::const_walker >.

Definition at line 2053 of file model.hpp.

10.134.4.14 return_value coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 762 of file evaluator.h.

10.134.4.15 int coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 756 of file evaluator.h.

10.134.4.16 void coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::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 771 of file evaluator.h.

10.134.4.17 return_value coco::cached_forward_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< model::detect_0chain_visitor_st, expression_node, std::pair< unsigned int, unsigned int >,model::const_walker >.

Definition at line 767 of file evaluator.h.

10.134.5 Member Data Documentation

10.134.5.1 model::detect_0chain_visitor_st coco::_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.134.5.2 const variable_indicator* coco::cached_evaluator_base< model::detect_0chain_visitor_st , expression_node , std::pair< unsigned int, unsigned int > , model::const_walker >::v_ind [protected, inherited]

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:

- model.hpp

10.135 coco::model::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

10.135.1 Detailed Description

This struct is the visitor data of the model::detect_0chain_visitor.

Definition at line 1972 of file model.hpp.

10.135.2 Member Data Documentation

10.135.2.1 std::vector<unsigned int>* coco::model::model::detect_0chain_visitor_st::d

vector of node depths

Definition at line 1977 of file model.hpp.

10.135.2.2 unsigned int* coco::model::model::detect_0chain_visitor_st::f

counter

Definition at line 1975 of file model.hpp.

10.135.2.3 bool coco::model::model::detect_0chain_visitor_st::new_number

is this a new 0-chain?

Definition at line 1979 of file model.hpp.

10.135.2.4 std::vector<unsigned int>* coco::model::model::detect_0chain_visitor_st::nn

vector of 0-chain bases

Definition at line 1978 of file model.hpp.

10.135.2.5 std::pair<unsigned int,unsigned int> coco::model::model::detect_0chain_visitor_st::r

return value (node number, depth)

Definition at line 1974 of file model.hpp.

10.135.2.6 std::vector<unsigned int>* coco::model::model::detect_0chain_visitor_st::t

vector of node types

Definition at line 1976 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

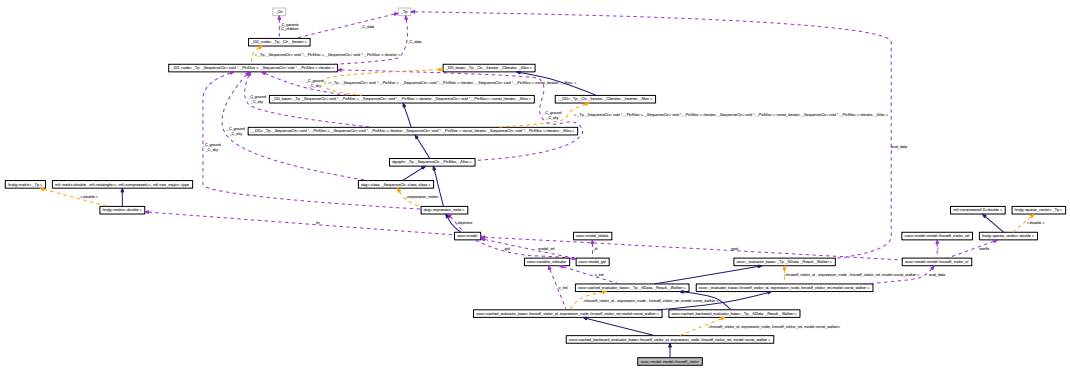
10.136 coco::model::model::lincoeff_visitor Class Reference

`#include <model.hpp>`

Inheritance diagram for coco::model::model::lincoeff_visitor:



Collaboration diagram for coco::model::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
- typedef lincoeff_visitor_st data_type

Public Member Functions

- lincoeff_visitor (const std::vector< interval > *_rg, linalg::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 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)

- 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)

Protected Attributes

- const variable_indicator * v_ind
- lincoeff_visitor_st eval_data

10.136.1 Detailed Description

This cached backward evaluator is used to collect the coefficients of a linear node on a DAG.

Definition at line 2109 of file model.hpp.

10.136.2 Member Typedef Documentation

10.136.2.1 typedef _Base::const_walker coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 853 of file evaluator.h.

10.136.2.2 typedef lincoeff_visitor_st coco::_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.136.2.3 typedef _Base::node_data_type coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 849 of file evaluator.h.

10.136.2.4 typedef _Base::return_value coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 851 of file evaluator.h.

10.136.3 Constructor & Destructor Documentation

10.136.3.1 coco::model::model::lincoeff_visitor::lincoeff_visitor (const std::vector< interval > * *_rg*, linalg::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 2122 of file model.hpp.

10.136.3.2 coco::model::model::lincoeff_visitor::lincoeff_visitor (const lincoeff_visitor & *__x*) [inline]

Standard Copy Constructor

Definition at line 2142 of file model.hpp.

10.136.3.3 coco::model::model::lincoeff_visitor::~lincoeff_visitor () [inline]

Standard Destructor

Definition at line 2145 of file model.hpp.

10.136.4 Member Function Documentation

10.136.4.1 int coco::model::model::lincoeff_visitor::calculate (const expression_node & *__data*) [inline, virtual]

This is a method as needed by a cached backward evaluator.

See also:

cached_backward_evaluator_base.

Reimplemented from coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 2205 of file model.hpp.

10.136.4.2 lincoeff_visitor_ret coco::model::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::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 2791 of file model.hpp.

10.136.4.3 virtual void coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::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 925 of file evaluator.h.

10.136.4.4 int coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 879 of file evaluator.h.

10.136.4.5 void coco::model::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 2172 of file model.hpp.

10.136.4.6 void coco::model::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::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 2161 of file model.hpp.

10.136.4.7 bool coco::model::model::lincoeff_visitor::is_cached (const expression_node & *__data*) [inline, virtual]

This is a method as needed by a cached backward evaluator.

See also:

cached_backward_evaluator_base.

Reimplemented from coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 2150 of file model.hpp.

10.136.4.8 void coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::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. It is translated into a call to cleanup.

Reimplemented from coco::_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 874 of file evaluator.h.

10.136.4.9 int coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls calculate.

Reimplemented from coco::cached_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 862 of file evaluator.h.

10.136.4.10 void coco::model::model::lincoeff_visitor::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

This is a method as needed by a cached backward evaluator.

See also:

cached_backward_evaluator_base.

Reimplemented from coco::cached_backward_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret, model::const_walker >.

Definition at line 2183 of file model.hpp.

10.136.4.11 virtual const_walker coco::model::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.

Reimplemented from coco::cached_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 2156 of file model.hpp.

10.136.4.12 virtual int coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::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 951 of file evaluator.h.

10.136.4.13 virtual int coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::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 940 of file evaluator.h.

10.136.4.14 int coco::model::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 2783 of file model.hpp.

10.136.4.15 int coco::model::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 2311 of file model.hpp.

10.136.4.16 return_value coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 891 of file evaluator.h.

10.136.4.17 int coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 885 of file evaluator.h.

10.136.4.18 void coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::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 900 of file evaluator.h.

10.136.4.19 return_value coco::cached_backward_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< lincoeff_visitor_st, expression_node, lincoeff_visitor_ret,model::const_walker >.

Definition at line 896 of file evaluator.h.

10.136.5 Member Data Documentation

10.136.5.1 lincoeff_visitor_st coco::_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.136.5.2 const variable_indicator* coco::cached_evaluator_base< lincoeff_visitor_st , expression_node , lincoeff_visitor_ret , model::const_walker >::v_ind [protected, inherited]

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:

- model.hpp

10.137 coco::model::model::lincoeff_visitor_ret Struct Reference

`#include <model.hpp>`

Public Attributes

- bool was_lin
- bool was_const
- int node_num
- double const_trans

10.137.1 Detailed Description

This is the return type of the model::lincoeff_visitor visitor.

Definition at line 2082 of file model.hpp.

10.137.2 Member Data Documentation

10.137.2.1 double coco::model::model::lincoeff_visitor_ret::const_trans

Definition at line 2087 of file model.hpp.

10.137.2.2 int coco::model::model::lincoeff_visitor_ret::node_num

Definition at line 2086 of file model.hpp.

10.137.2.3 bool coco::model::model::lincoeff_visitor_ret::was_const

Definition at line 2085 of file model.hpp.

10.137.2.4 bool coco::model::model::lincoeff_visitor_ret::was_lin

Definition at line 2084 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

10.138 coco::model::model::lincoeff_visitor_st Struct Reference

`#include <model.hpp>`

Collaboration diagram for coco::model::model::lincoeff_visitor_st:



Public Attributes

- const model * mod
- const std::vector< interval > * ranges
- linalg::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

10.138.1 Detailed Description

This is the visitor data of the model::lincoeff_visitor visitor.

Definition at line 2091 of file model.hpp.

10.138.2 Member Data Documentation

10.138.2.1 linalg::sparse_vector<double>* coco::model::model::lincoeff_visitor_st::coeffs

the coefficents of the linear function

Definition at line 2095 of file model.hpp.

10.138.2.2 double* coco::model::model::lincoeff_visitor_st::constant

the constant of the linear function

Definition at line 2097 of file model.hpp.

10.138.2.3 int coco::model::model::lincoeff_visitor_st::in_nlin

marker for non-linear parts

Definition at line 2103 of file model.hpp.

10.138.2.4 double coco::model::model::lincoeff_visitor_st::mm_help

helper for max, min

Definition at line 2100 of file model.hpp.

10.138.2.5 const model* coco::model::model::lincoeff_visitor_st::mod

the DAG

Definition at line 2093 of file model.hpp.

10.138.2.6 unsigned int coco::model::model::lincoeff_visitor_st::n

children counter

Definition at line 2104 of file model.hpp.

10.138.2.7 int coco::model::model::lincoeff_visitor_st::old_nlin

old marker for non-linear parts

Definition at line 2102 of file model.hpp.

10.138.2.8 double coco::model::model::lincoeff_visitor_st::old_trans

the old scaling factor

Definition at line 2098 of file model.hpp.

10.138.2.9 lincoeff_visitor_ret coco::model::model::lincoeff_visitor_st::r

the return value

Definition at line 2101 of file model.hpp.

10.138.2.10 const std::vector<interval>* coco::model::model::lincoeff_visitor_st::ranges

the ranges of all nodes

Definition at line 2094 of file model.hpp.

10.138.2.11 double coco::model::model::lincoeff_visitor_st::trans

the scaling factor

Definition at line 2099 of file model.hpp.

The documentation for this struct was generated from the following file:

- model.hpp

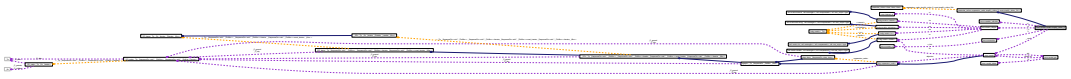
10.139 coco::model::model::simplify_visitor_0 Class Reference

`#include <model.hpp>`

Inheritance diagram for coco::model::model::simplify_visitor_0:



Collaboration diagram for coco::model::model::simplify_visitor_0:



Public Member Functions

- simplify_visitor_0 ()
- simplify_visitor_0 (int __nv, std::vector< unsigned int > *__dn, std::vector< unsigned int > *__dg, std::vector< std::pair< unsigned int, unsigned int > > *__de, std::vector< bool > *__vt, model *__m)
- 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 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)
- virtual bool **postorder** (const _Node &__n)
- virtual void **collect** (const _Node &__n, collect_value __r)

10.139.1 Detailed Description

This class is a postorder visitor used for the phase 0 of the simplifier.

Definition at line 716 of file model.hpp.

10.139.2 Constructor & Destructor Documentation

10.139.2.1 coco::model::model::simplify_visitor_0::simplify_visitor_0 () `[inline]`

Definition at line 742 of file model.hpp.

10.139.2.2 coco::model::model::simplify_visitor_0::simplify_visitor_0 (int *__nv*, std::vector< unsigned int > * *__dn*, std::vector< unsigned int > * *__dg*, std::vector< std::pair< unsigned int, unsigned int > > * *__de*, std::vector< bool > * *__vt*, model * *__m*) [inline]

Definition at line 750 of file model.hpp.

10.139.2.3 coco::model::model::simplify_visitor_0::simplify_visitor_0 (const simplify_visitor_0 & *__x*) [inline]

Definition at line 762 of file model.hpp.

10.139.3 Member Function Documentation

10.139.3.1 virtual void postorder_visitor< class, class, class >::collect (const _Node & *__n*, collect_value *__r*) [virtual, inherited]

10.139.3.2 void coco::model::model::simplify_visitor_0::collect (expression_node & *r*, const simplify_visitor_0 & *__s*)

Definition at line 1522 of file model.cc.

10.139.3.3 void coco::model::model::simplify_visitor_0::init () [inline, virtual]

Reimplemented from **postorder_visitor< expression_node, simplify_visitor_0, const simplify_visitor_0 & >**.

Definition at line 777 of file model.hpp.

10.139.3.4 virtual bool postorder_visitor< class, class, class >::postorder (const _Node & *__n*) [virtual, inherited]

10.139.3.5 bool coco::model::model::simplify_visitor_0::postorder (expression_node & *r*)

Definition at line 1349 of file model.cc.

10.139.3.6 void coco::model::model::simplify_visitor_0::postorder_help (const expression_node & *r*, unsigned int *n_chld*) [inline]

Definition at line 2818 of file model.hpp.

10.139.3.7 bool coco::model::model::simplify_visitor_0::preorder (expression_node & *r*) [inline]

Definition at line 782 of file model.hpp.

10.139.3.8 void coco::model::model::simplify_visitor_0::simple_sum_prod_update (expression_node & *r*, const simplify_visitor_0 & *__s*) [inline]

Definition at line 823 of file model.hpp.

10.139.3.9 void coco::model::model::simplify_visitor_0::transfer_ghost_down (unsigned int *nnum*, unsigned int *pnum*) [inline]

Definition at line 872 of file model.hpp.

10.139.3.10 simplify_visitor_0 coco::model::model::simplify_visitor_0::value () [inline]

Definition at line 820 of file model.hpp.

10.139.3.11 void coco::model::model::simplify_visitor_0::vcollect (const simplify_visitor_0 & *__s*)

Definition at line 1418 of file model.cc.

10.139.3.12 void coco::model::model::simplify_visitor_0::vinit () [inline, virtual]

Reimplemented from **postorder_visitor**< **expression_node**, **simplify_visitor_0**, const **simplify_visitor_0** & >.

Definition at line 774 of file model.hpp.

10.139.3.13 simplify_visitor_0 coco::model::model::simplify_visitor_0::vvalue () [inline, virtual]

Reimplemented from **postorder_visitor< expression_node, simplify_visitor_0, const simplify_visitor_0 & >**.

Definition at line 821 of file model.hpp.

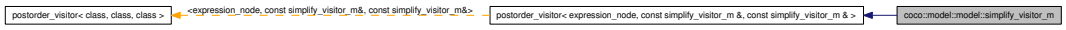
The documentation for this class was generated from the following files:

- model.hpp
- model.cc

10.140 coco::model::model::simplify_visitor_m Class Reference

`#include <model.hpp>`

Inheritance diagram for coco::model::model::simplify_visitor_m:



Collaboration diagram for coco::model::model::simplify_visitor_m:



Public Member Functions

- simplify_visitor_m ()
- simplify_visitor_m (int __nv, std::vector< unsigned int > *__dn, std::vector< std::pair< 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)
- virtual bool **postorder** (const _Node &__n)
- virtual void **collect** (const _Node &__n, collect_value __r)

10.140.1 Detailed Description

This class is a postorder visitor used for the micro-simplifier, which only initializes the most important counters and semantics structures.

Definition at line 919 of file model.hpp.

10.140.2 Constructor & Destructor Documentation

10.140.2.1 coco::model::model::simplify_visitor_m::simplify_visitor_m () [inline]

Definition at line 941 of file model.hpp.

10.140.2.2 coco::model::model::simplify_visitor_m::simplify_visitor_m (int *__nv*, std::vector< unsigned int > * *__dn*, std::vector< std::pair< unsigned int, unsigned int > > * *__de*, std::vector< bool > * *__vt*, model * *__m*) [inline]

Definition at line 947 of file model.hpp.

10.140.2.3 coco::model::model::simplify_visitor_m::simplify_visitor_m (const simplify_visitor_m & *__x*) [inline]

Definition at line 956 of file model.hpp.

10.140.3 Member Function Documentation

10.140.3.1 virtual void postorder_visitor< class, class, class >::collect (const _Node & *__n*, collect_value *__r*) [virtual, inherited]

10.140.3.2 void coco::model::model::simplify_visitor_m::collect (expression_node & *r*, const simplify_visitor_m & *__s*)

Definition at line 2836 of file model.cc.

10.140.3.3 void coco::model::model::simplify_visitor_m::init () [inline, virtual]

Reimplemented from **postorder_visitor< expression_node, const simplify_visitor_m &, const simplify_visitor_m & >**.

Definition at line 969 of file model.hpp.

10.140.3.4 virtual bool postorder_visitor< class, class, class >::postorder (const _Node & *__n*) [virtual, inherited]

10.140.3.5 bool coco::model::model::simplify_visitor_m::postorder (expression_node & *r*)

Definition at line 2804 of file model.cc.

10.140.3.6 void coco::model::model::simplify_visitor_m::postorder_help (const expression_node & *r*, unsigned int *n_chld*) [inline]

Definition at line 2928 of file model.hpp.

10.140.3.7 bool coco::model::model::simplify_visitor_m::preorder (expression_node & *r*) [inline]

Definition at line 974 of file model.hpp.

10.140.3.8 const simplify_visitor_m& coco::model::model::simplify_visitor_m::value () [inline]

Definition at line 987 of file model.hpp.

10.140.3.9 void coco::model::model::simplify_visitor_m::vcollect (const simplify_visitor_m & *__s*) [inline]

Definition at line 990 of file model.hpp.

10.140.3.10 void coco::model::model::simplify_visitor_m::vinit () [inline, virtual]

Reimplemented from **postorder_visitor< expression_node, const simplify_visitor_m &, const simplify_visitor_m & >**.

Definition at line 966 of file model.hpp.

10.140.3.11 const simplify_visitor_m& coco::model::model::simplify_visitor_m::vvalue () [inline, virtual]

Reimplemented from **postorder_visitor< expression_node, const simplify_visitor_m &, const simplify_visitor_m & >**.

Definition at line 988 of file model.hpp.

The documentation for this class was generated from the following files:

- model.hpp
- model.cc

10.141 coco::model::model::sort_constraints Class Reference

`#include <model.hpp>`

Public Member Functions

- bool operator() (const **walker** &_c1, const **walker** &_c2) const

10.141.1 Detailed Description

This function class is used to compare constraints for constraint sorting.

Definition at line 665 of file model.hpp.

10.141.2 Member Function Documentation

10.141.2.1 bool coco::model::model::sort_constraints::operator() (const walker & *_c1*, const walker & *_c2*) const [inline]

Definition at line 669 of file model.hpp.

The documentation for this class was generated from the following file:

- model.hpp

10.142 coco::model_gid Class Reference

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

`#include <model.h>`

Collaboration diagram for coco::model_gid:



Public Member Functions

- void remove_node_ref (unsigned int _n)
- void remove_var_ref (unsigned int _n)
- void remove_const_ref (unsigned int _n)
- 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_constraints () const
- void number_of_nodes (unsigned int _n)
- void number_of_variables (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_gconstref (unsigned int n, const **model::walker** &__w)
- 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_const_id ()
- void remove_const_id (unsigned int n)
- void compress_numbers (bool renumber_vars=false, bool renumber_const=false)
- const **model::walker** & node (unsigned int i) const
- const **model::walker** & variable (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_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 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)
- const std::string obj_name () const
- void obj_name (const std::string &vn)
- void obj_name (const char *vn)
- double obj_adj () const
- void obj_adj (double adj)
- double obj_mult () const
- void obj_mult (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_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

10.142.1 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.

Definition at line 750 of file model.h.

10.142.2 Constructor & Destructor Documentation

10.142.2.1 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 304 of file model.hpp.

10.142.2.2 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 314 of file model.hpp.

10.142.2.3 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 324 of file model.hpp.

10.142.2.4 coco::model_gid::~model_gid () [inline]

Standard Destructor

Definition at line 835 of file model.h.

10.142.3 Member Function Documentation

10.142.3.1 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 1033 of file model.h.

10.142.3.2 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 1029 of file model.h.

10.142.3.3 void coco::model_gid::compress_numbers (bool *renumber_vars* = `false`, bool *renumber_const* = `false`) [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. The parameters `renumber_vars` and `renumber_const` determine whether the variable and constraint numbers, respectively, are renumbered in addition to the node numbers.

Definition at line 902 of file model.h.

10.142.3.4 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 965 of file model.h.

10.142.3.5 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 962 of file model.h.

10.142.3.6 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 959 of file model.h.

10.142.3.7 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 910 of file model.h.

10.142.3.8 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 915 of file model.h.

10.142.3.9 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 922 of file model.h.

10.142.3.10 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 998 of file model.h.

10.142.3.11 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 996 of file model.h.

10.142.3.12 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 993 of file model.h.

10.142.3.13 unsigned int coco::model_gid::get_const_id () [inline]

This method returns a free constraint number a removes it from the list of free constraint numbers.

Definition at line 891 of file model.h.

10.142.3.14 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 481 of file model.hpp.

10.142.3.15 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 879 of file model.h.

10.142.3.16 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 885 of file model.h.

10.142.3.17 bool coco::model_gid::have_gconst_ref (unsigned int *_cnum*) const [inline]

This method checks whether a global constraint reference exists for constraint number `_nnum`.

Definition at line 934 of file model.h.

10.142.3.18 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 926 of file model.h.

10.142.3.19 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 930 of file model.h.

10.142.3.20 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 919 of file model.h.

10.142.3.21 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 492 of file model.hpp.

10.142.3.22 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 441 of file model.hpp.

10.142.3.23 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 415 of file model.hpp.

10.142.3.24 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 428 of file model.hpp.

10.142.3.25 void coco::model_gid::model_name (const char * *n*) [inline]

This method sets the name of the model to n.

Definition at line 948 of file model.h.

10.142.3.26 void coco::model_gid::model_name (const std::string & *n*) [inline]

This method sets the name of the model to `n`.

Definition at line 946 of file model.h.

10.142.3.27 const std::string coco::model_gid::model_name () const [inline]

This method returns the name of the model.

Definition at line 944 of file model.h.

10.142.3.28 size_t coco::model_gid::n_fixed_vars () const [inline]

This method returns the number of variables fixed by the presolve.

Definition at line 991 of file model.h.

10.142.3.29 size_t coco::model_gid::n_unused_constrs () const [inline]

This method returns the number of unused constraints.

Definition at line 1012 of file model.h.

10.142.3.30 size_t coco::model_gid::n_unused_vars () const [inline]

This method returns the number of unused variables.

Definition at line 1002 of file model.h.

10.142.3.31 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 906 of file model.h.

10.142.3.32 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 861 of file model.h.

10.142.3.33 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 851 of file model.h.

10.142.3.34 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 855 of file model.h.

10.142.3.35 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 844 of file model.h.

10.142.3.36 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 858 of file model.h.

10.142.3.37 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 847 of file model.h.

10.142.3.38 void coco::model_gid::obj_adj (double *adj*) [inline]

With this method the additive constant in the objective function is set to `adj`.

Definition at line 983 of file model.h.

10.142.3.39 double coco::model_gid::obj_adj () const [inline]

This method returns the additive constant in the objective function.

Definition at line 980 of file model.h.

10.142.3.40 void coco::model_gid::obj_mult (double *mult*) [inline]

With this method the scalar multiplier of the objective function is set to `mult`.

Definition at line 988 of file model.h.

10.142.3.41 double coco::model_gid::obj_mult () const [inline]

This method returns the scalar multiplier of the objective function.

Definition at line 985 of file model.h.

10.142.3.42 void coco::model_gid::obj_name (const char * *vn*) [inline]

A call to this method sets the name of the objective function to `vn`.

Definition at line 978 of file model.h.

10.142.3.43 void coco::model_gid::obj_name (const std::string & *vn*) [inline]

A call to this method sets the name of the objective function to `vn`.

Definition at line 976 of file model.h.

10.142.3.44 const std::string coco::model_gid::obj_name () const [inline]

This method returns the name of the objective function.

Definition at line 974 of file model.h.

10.142.3.45 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 504 of file model.hpp.

10.142.3.46 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 894 of file model.h.

10.142.3.47 void coco::model_gid::remove_const_ref (unsigned int *_n*) [inline]

This method removes the global constraint reference of node _n.

Definition at line 459 of file model.hpp.

10.142.3.48 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 882 of file model.h.

10.142.3.49 void coco::model_gid::remove_node_ref (unsigned int *_n*) [inline]

This method removes the global node reference of node _n.

Definition at line 812 of file model.h.

10.142.3.50 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 888 of file model.h.

10.142.3.51 void coco::model_gid::remove_var_ref (unsigned int *_n*) [inline]

This method removes the global variable reference of node _n.

Definition at line 816 of file model.h.

10.142.3.52 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 501 of file model.hpp.

10.142.3.53 void coco::model_gid::unused_constr (const char * *vn*) [inline]

This method stores the unused constraint with name vn.

Definition at line 1019 of file model.h.

10.142.3.54 void coco::model_gid::unused_constr (const std::string & *vn*) [inline]

This method stores the unused constraint with name vn.

Definition at line 1017 of file model.h.

10.142.3.55 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 1014 of file model.h.

10.142.3.56 void coco::model_gid::unused_var (const char * *vn*) [inline]

This method stores the unused variable with name vn.

Definition at line 1009 of file model.h.

10.142.3.57 void coco::model_gid::unused_var (const std::string & *vn*) [inline]

This method stores the unused variable with name vn.

Definition at line 1007 of file model.h.

10.142.3.58 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 1004 of file model.h.

10.142.3.59 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 955 of file model.h.

10.142.3.60 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 953 of file model.h.

10.142.3.61 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 951 of file model.h.

10.142.3.62 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 908 of file model.h.

10.142.4 Friends And Related Function Documentation

10.142.4.1 friend class model_iddata [friend]

Definition at line 1037 of file model.h.

The documentation for this class was generated from the following files:

- model.h
- model.cc
- model.hpp

10.143 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_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_constraints () const
- void number_of_nodes (unsigned int _n)
- void number_of_variables (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_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 () const
- void obj_name (const std::string &vn)
- double obj_adj () const
- void obj_adj (double adj)
- double obj_mult () const
- void obj_mult (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_constrs () const
- const std::string & unused_constr (unsigned int n) const
- void unused_constr (const std::string &vn)

Friends

- class model_gid

10.143.1 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, ...).

Definition at line 552 of file model.h.

10.143.2 Constructor & Destructor Documentation

10.143.2.1 coco::model_iddata::model_iddata (unsigned int *n* = 0) [inline]

Standard Constructor, which prepares for n variables.

Definition at line 612 of file model.h.

10.143.2.2 coco::model_iddata::~model_iddata () [inline]

Standard Destructor

Definition at line 623 of file model.h.

10.143.3 Member Function Documentation

10.143.3.1 void coco::model_iddata::compress_numbers (bool *renum_vars*, 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 77 of file model.cc.

10.143.3.2 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 235 of file model.hpp.

10.143.3.3 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 223 of file model.hpp.

10.143.3.4 bool coco::model_iddata::delete_ref (model_gid & *__m*) [inline]

This method deletes the back reference to the model_gid __m.

Definition at line 42 of file model.hpp.

10.143.3.5 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 271 of file model.hpp.

10.143.3.6 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 261 of file model.hpp.

10.143.3.7 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 124 of file model.hpp.

10.143.3.8 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 56 of file model.hpp.

10.143.3.9 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 89 of file model.hpp.

10.143.3.10 void coco::model_iddata::model_name (const std::string & *n*) [inline]

This method sets the name of the model to n.

Definition at line 194 of file model.hpp.

10.143.3.11 const std::string coco::model_iddata::model_name () const [inline]

This method returns the name of the model.

Definition at line 186 of file model.hpp.

10.143.3.12 size_t coco::model_iddata::n_fixed_vars () const [inline]

This method returns the number of variables fixed by the preprocessor.

Definition at line 717 of file model.h.

10.143.3.13 size_t coco::model_iddata::n_unused_constrs () const [inline]

This method returns the number of unused constraints.

Definition at line 731 of file model.h.

10.143.3.14 size_t coco::model_iddata::n_unused_vars () const [inline]

This method returns the number of unused variables.

Definition at line 724 of file model.h.

10.143.3.15 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 626 of file model.h.

10.143.3.16 void coco::model_iddata::number_of_constraints (unsigned int *_n*) [inline]

This method sets the number of constraints to _n.

Definition at line 177 of file model.hpp.

10.143.3.17 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 646 of file model.h.

10.143.3.18 void coco::model_iddata::number_of_nodes (unsigned int *_n*) [inline]

This method sets the number of nodes to _n.

Definition at line 160 of file model.hpp.

10.143.3.19 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 638 of file model.h.

10.143.3.20 void coco::model_iddata::number_of_variables (unsigned int *_n*) [inline]

This method sets the number of variables to _n.

Definition at line 168 of file model.hpp.

10.143.3.21 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 642 of file model.h.

10.143.3.22 void coco::model_iddata::obj_adj (double *adj*) [inline]

With this method the additive constant in the objective function is set to adj.

Definition at line 257 of file model.hpp.

10.143.3.23 double coco::model_iddata::obj_adj () const [inline]

This method returns the additive constant in the objective function.

Definition at line 256 of file model.hpp.

10.143.3.24 void coco::model_iddata::obj_mult (double *mult*) [inline]

With this method the scalar multiplier of the objective function is set to `mult`.

Definition at line 259 of file model.hpp.

10.143.3.25 double coco::model_iddata::obj_mult () const [inline]

This method returns the scalar multiplier of the objective function.

Definition at line 258 of file model.hpp.

10.143.3.26 void coco::model_iddata::obj_name (const std::string & *vn*) [inline]

A call to this method sets the name of the objective function to `vn`.

Definition at line 255 of file model.hpp.

10.143.3.27 const std::string coco::model_iddata::obj_name () const [inline]

This method returns the name of the objective function.

Definition at line 247 of file model.hpp.

10.143.3.28 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 137 of file model.hpp.

10.143.3.29 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 69 of file model.hpp.

10.143.3.30 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 102 of file model.hpp.

10.143.3.31 void coco::model_iddata::unused_constr (const std::string & *vn*) [inline]

This method stores the unused constraint with name vn.

Definition at line 299 of file model.hpp.

10.143.3.32 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 290 of file model.hpp.

10.143.3.33 void coco::model_iddata::unused_var (const std::string & *vn*) [inline]

This method stores the unused variable with name vn.

Definition at line 285 of file model.hpp.

10.143.3.34 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 276 of file model.hpp.

10.143.3.35 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 211 of file model.hpp.

10.143.3.36 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 199 of file model.hpp.

10.143.4 Friends And Related Function Documentation

10.143.4.1 friend class model_gid [friend]

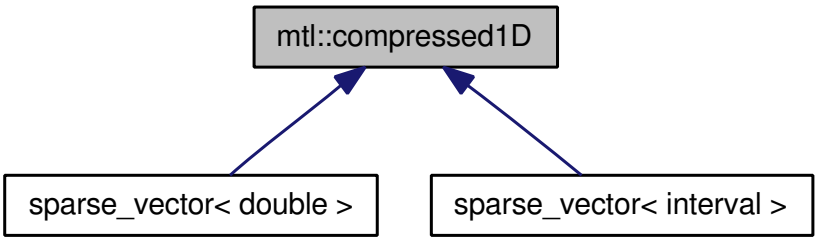
Definition at line 738 of file model.h.

The documentation for this class was generated from the following files:

- model.h
- model.cc
- model.hpp

10.144 mtl::compressed1D Class Reference

Inheritance diagram for mtl::compressed1D:

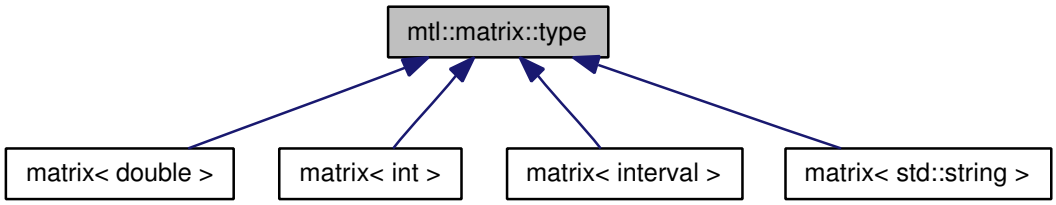


The documentation for this class was generated from the following file:

- linalg.h

10.145 mtl::matrix::type Class Reference

Inheritance diagram for mtl::matrix::type:



The documentation for this class was generated from the following file:

- linalg.h

10.146 coco::my_rounded_math< T > Struct Template Reference

`#include <interval_boost.h>`

10.146.1 Detailed Description

template<class T> struct coco::my_rounded_math< T >

Definition at line 90 of file interval_boost.h.

The documentation for this struct was generated from the following file:

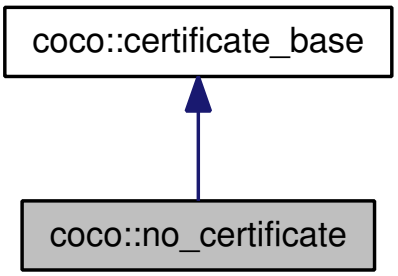
- interval_boost.h

10.147 coco::no_certificate Class Reference

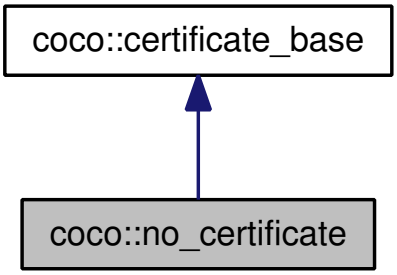
The not-certified certificate.

`#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)
- virtual std::string get_contents () const

Protected Attributes

- std::string _contents

10.147.1 Detailed Description

The empty certificate for inference engines, which do not produce certificates!

Definition at line 84 of file api_cert.h.

10.147.2 Constructor & Destructor Documentation

10.147.2.1 coco::no_certificate::no_certificate () [inline]

Standard Constructor

Definition at line 88 of file api_cert.h.

10.147.2.2 coco::no_certificate::no_certificate (const no_certificate & __c) [inline]

Standard Copy Constructor

Definition at line 90 of file api_cert.h.

10.147.2.3 virtual coco::no_certificate::~no_certificate () [inline, virtual]

Standard Destructor

Definition at line 98 of file api_cert.h.

10.147.3 Member Function Documentation

10.147.3.1 void coco::no_certificate::destroy_copy (certificate_base * __c) const [inline]

Clone Destructor

Definition at line 109 of file api_cert.h.

10.147.3.2 virtual std::string coco::certificate_base::get_contents () const [inline, virtual, inherited]

Retrieve the contents information (the certificate type) for this certificate.

Reimplemented in coco::split_certificate, and coco::rigorous_module_certificate.

Definition at line 165 of file api_certbase.h.

10.147.3.3 certificate 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 api_certbase.h.

10.147.3.4 no_certificate* coco::no_certificate::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::certificate_base.

Definition at line 101 of file api_cert.h.

10.147.3.5 bool coco::no_certificate::operator!= (const no_certificate & _c) const [inline]

Definition at line 126 of file api_cert.h.

10.147.3.6 bool coco::no_certificate::operator!= (const certificate_base & _c) const [inline]

Definition at line 120 of file api_cert.h.

10.147.3.7 bool coco::no_certificate::operator== (const no_certificate & _c) const [inline]

Comparison operators

Definition at line 125 of file api_cert.h.

10.147.3.8 bool coco::no_certificate::operator== (const certificate_base & _c) const [inline]

Comparison operators

Definition at line 117 of file api_cert.h.

10.147.3.9 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::certificate_base.

Definition at line 114 of file api_cert.h.

10.147.4 Member Data Documentation

10.147.4.1 std::string coco::certificate_base::_contents [protected, inherited]

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:

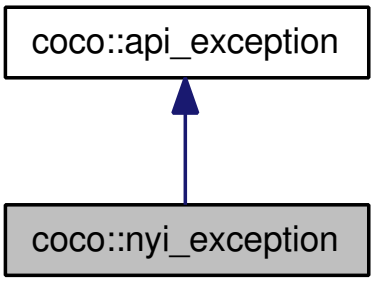
- api_cert.h

10.148 coco::nyi_exception Class Reference

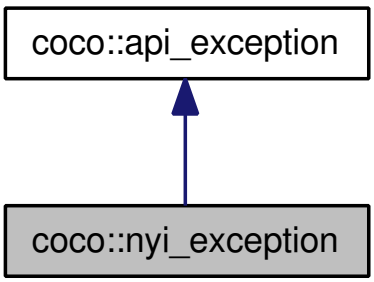
Not Yet Implemented exception class.

`#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 api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.148.1 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 api_exception.h.

10.148.2 Constructor & Destructor Documentation

10.148.2.1 coco::nyi_exception::nyi_exception (char const * *m*) [inline]

Constructor, setting the message to `m`

Definition at line 121 of file api_exception.h.

10.148.2.2 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.

10.148.2.3 virtual coco::nyi_exception::~nyi_exception () throw () [inline, virtual]

Standard Destructor

Definition at line 126 of file api_exception.h.

10.148.3 Member Function Documentation

10.148.3.1 virtual std::string coco::api_exception::message () const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file api_exception.h.

10.148.3.2 virtual api_exception_type coco::api_exception::type () const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file api_exception.h.

10.148.3.3 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.

10.148.3.4 virtual char const* coco::api_exception::what () const throw () [inline, virtual, inherited]

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:

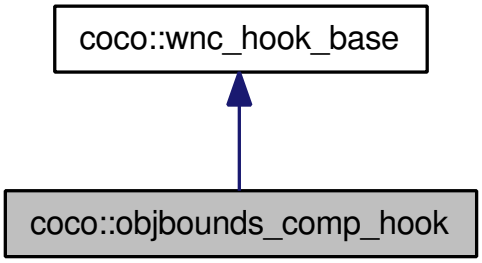
- api_exception.h

10.149 coco::objbounds_comp_hook Class Reference

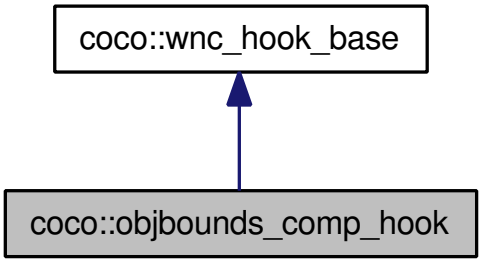
The objective-bounds computation hook (work node computation hook).

`#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

10.149.1 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"

Definition at line 46 of file objbounds_hook.h.

10.149.2 Constructor & Destructor Documentation

10.149.2.1 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.

10.149.2.2 virtual coco::objbounds_comp_hook::~objbounds_comp_hook () [inline, virtual]

Standard Destructor

Definition at line 60 of file objbounds_hook.h.

10.149.3 Member Function Documentation

10.149.3.1 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.

10.149.3.2 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.

10.149.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *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 238 of file comp_hook.h.

10.149.3.4 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.

10.149.3.5 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.

10.149.3.6 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.

10.149.3.7 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.

10.149.3.8 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.

10.149.3.9 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.

10.149.3.10 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:

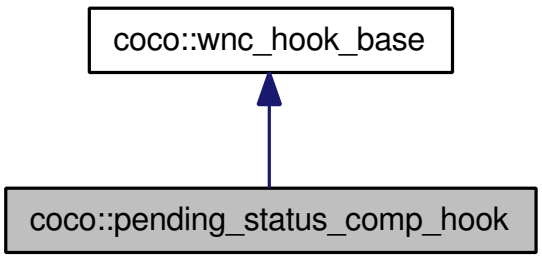
- objbounds_hook.h

10.150 coco::pending_status_comp_hook Class Reference

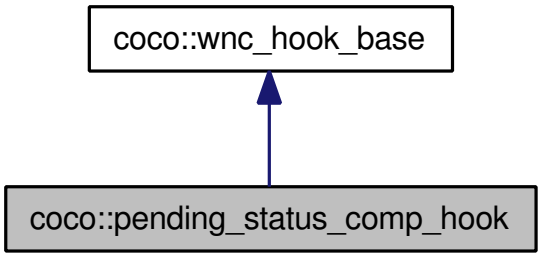
The pending status computation hook (work node computation hook).

`#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

10.150.1 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".

Definition at line 44 of file pending_status_hook.h.

10.150.2 Constructor & Destructor Documentation

10.150.2.1 coco::pending_status_comp_hook::pending_status_comp_hook () [inline]

Standard Constructor

Definition at line 48 of file pending_status_hook.h.

10.150.2.2 virtual coco::pending_status_comp_hook::~pending_status_comp_hook () [inline, virtual]

Standard Destructor

Definition at line 51 of file pending_status_hook.h.

10.150.3 Member Function Documentation

10.150.3.1 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.

10.150.3.2 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.

10.150.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *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 238 of file comp_hook.h.

10.150.3.4 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.

10.150.3.5 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.

10.150.3.6 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.

10.150.3.7 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.

10.150.3.8 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.

10.150.3.9 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.

10.150.3.10 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:

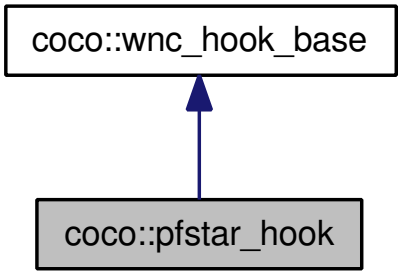
- pending_status_hook.h

10.151 coco::pfstar_hook Class Reference

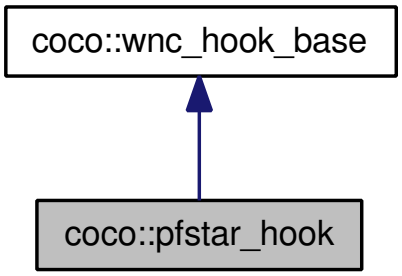
The pfstar computation hook (work node computation hook).

`#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

10.151.1 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".

Definition at line 43 of file pfstar_hook.h.

10.151.2 Constructor & Destructor Documentation

10.151.2.1 coco::pfstar_hook::pfstar_hook () [inline]

Standard Constructor

Definition at line 47 of file pfstar_hook.h.

10.151.2.2 virtual coco::pfstar_hook::~pfstar_hook () [inline, virtual]

Standard Destructor

Definition at line 52 of file pfstar_hook.h.

10.151.3 Member Function Documentation

10.151.3.1 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.

10.151.3.2 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.

10.151.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *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 238 of file comp_hook.h.

10.151.3.4 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.

10.151.3.5 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.

10.151.3.6 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.

10.151.3.7 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.

10.151.3.8 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.

10.151.3.9 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.

10.151.3.10 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:

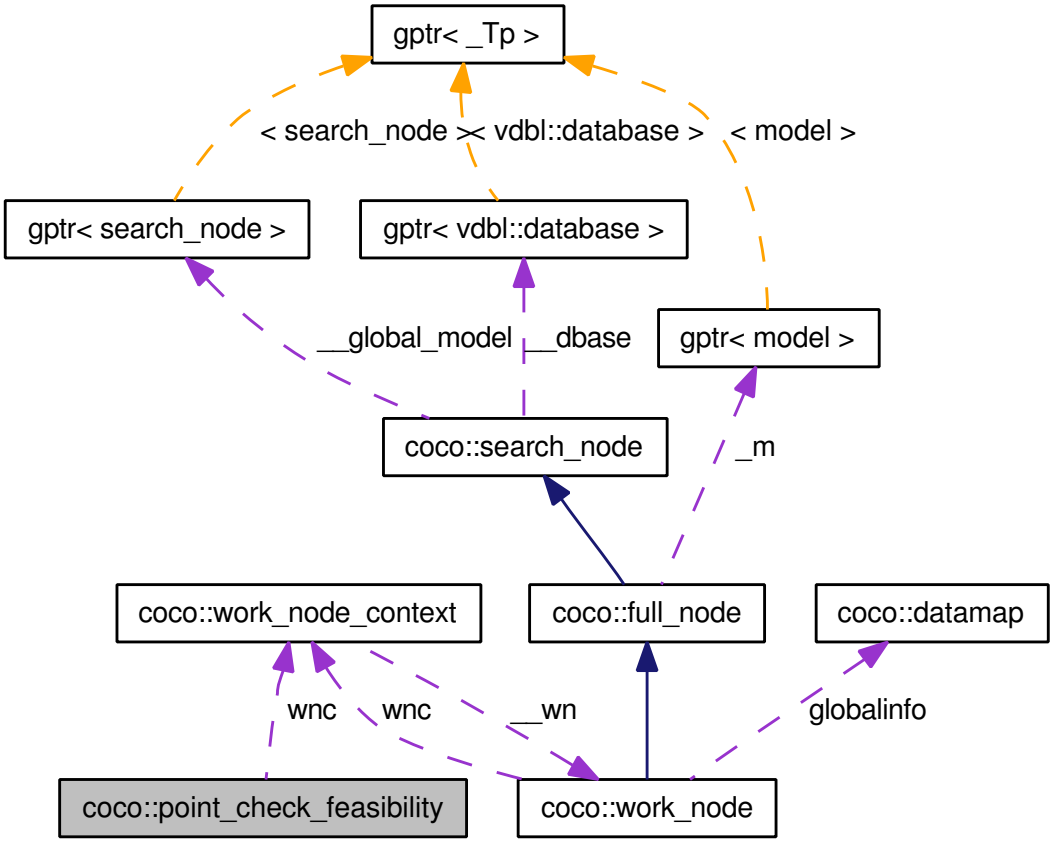
- pfstar_hook.h

10.152 coco::point_check_feasibility Class Reference

Stored procedure checking the feasibility of a point.

```
#include <dbtools.h>
```

Collaboration diagram for coco::point_check_feasibility:



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)

10.152.1 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.

Definition at line 44 of file dbtools.h.

10.152.2 Member Typedef Documentation

10.152.2.1 typedef work_node_context coco::point_check_feasibility::context

The evaluation context type

Definition at line 48 of file dbtools.h.

10.152.2.2 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.

10.152.3 Constructor & Destructor Documentation

10.152.3.1 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.

10.152.3.2 coco::point_check_feasibility::point_check_feasibility (const point_check_feasibility & *_i*) [inline]

Standard Copy Constructor

Definition at line 72 of file dbtools.h.

10.152.3.3 virtual coco::point_check_feasibility::~point_check_feasibility () [inline, virtual]

Standard Destructor

Definition at line 76 of file dbtools.h.

10.152.4 Member Function Documentation

10.152.4.1 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.

10.152.4.2 bool coco::point_check_feasibility::operator() () const

Evaluation operator, which computes the value of the stored procedure

Definition at line 35 of file dbtools.cc.

10.152.4.3 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 files:

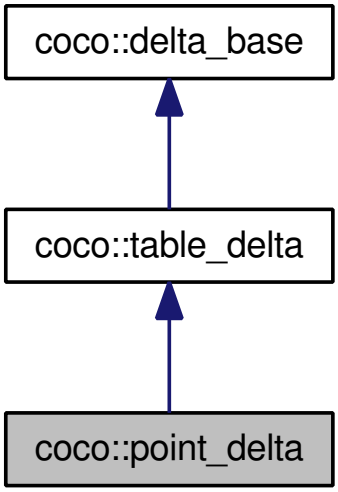
- dbtools.h
- dbtools.cc

10.153 coco::point_delta Class Reference

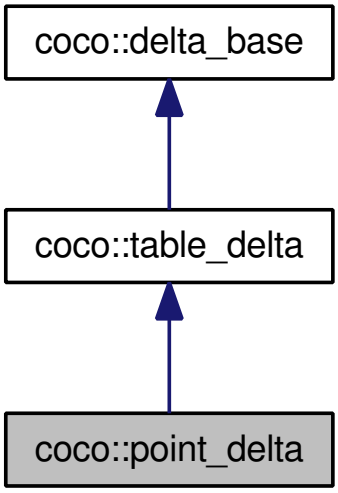
A delta class which adds new points to the search database.

`#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) 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)
- void convert (work_node &_x, delta_base *&_u)
- bool operator== (const table_delta &_c) const
- bool operator!= (const table_delta &_c) const
- delta make_delta (const std::string &a)
- const std::string & get_action () const
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Protected Attributes

- std::string _action

10.153.1 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.

Definition at line 44 of file point_delta.h.

10.153.2 Member Typedef Documentation

10.153.2.1 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.

10.153.2.2 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.

10.153.3 Constructor & Destructor Documentation

10.153.3.1 coco::point_delta::point_delta () [inline]

Standard Constructor

Definition at line 52 of file point_delta.h.

10.153.3.2 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.

10.153.3.3 coco::point_delta::point_delta (const point_delta & *__d*) [inline]

Standard Copy Constructor

Definition at line 77 of file point_delta.h.

10.153.4 Member Function Documentation

10.153.4.1 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.

10.153.4.2 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.

10.153.4.3 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.

10.153.4.4 bool coco::point_delta::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_did*) 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.

10.153.4.5 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*) 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.

10.153.4.6 void coco::table_delta::convert (work_node & *_x*, delta_base *& *_u*) [virtual, 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.

Reimplemented from coco::delta_base.

Definition at line 42 of file table_delta.cc.

10.153.4.7 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 44 of file point_delta.cc.

10.153.4.8 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.

10.153.4.9 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.153.4.10 delta 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 api_deltabase.h.

10.153.4.11 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.

10.153.4.12 bool coco::table_delta::operator!= (const table_delta & _c) const [inline, inherited]

Definition at line 143 of file table_delta.h.

10.153.4.13 bool coco::point_delta::operator!= (const point_delta & _c) const [inline]

Definition at line 112 of file point_delta.h.

10.153.4.14 bool coco::point_delta::operator!= (const delta_base & _c) const [inline]

Reimplemented from coco::table_delta.

Definition at line 105 of file point_delta.h.

10.153.4.15 bool coco::table_delta::operator== (const table_delta & _c) const [inline, inherited]

Comparison operators

Definition at line 142 of file table_delta.h.

10.153.4.16 bool coco::point_delta::operator== (const point_delta & _c) const [inline]

Comparison operators

Definition at line 110 of file point_delta.h.

10.153.4.17 bool coco::point_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Reimplemented from coco::table_delta.

Definition at line 102 of file point_delta.h.

10.153.4.18 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.

10.153.4.19 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.

10.153.4.20 virtual void 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 api_deltabase.h.

10.153.5 Member Data Documentation

10.153.5.1 std::string coco::delta_base::_action [protected, inherited]

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 files:

- point_delta.h
- point_delta.cc

10.154 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
- typedef std::vector< std::vector< double > > * data_type

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)
- 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 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 const_walker short_cut_to (const node_data_type &__data)

- 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)

- 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)

Protected Attributes

- const variable_indicator * v_ind
- std::vector< std::vector< double > > * eval_data

10.154.1 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.

Definition at line 62 of file der_evaluator.h.

10.154.2 Member Typedef Documentation

10.154.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.154.2.2 typedef std::vector< std::vector< double > > * coco::_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.154.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.154.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.154.3 Constructor & Destructor Documentation

10.154.3.1 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 74 of file der_evaluator.h.

10.154.3.2 coco::prep_d_eval::prep_d_eval (const prep_d_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 84 of file der_evaluator.h.

10.154.3.3 coco::prep_d_eval::~prep_d_eval () [inline]

Standard Destructor

Definition at line 87 of file der_evaluator.h.

10.154.3.4 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 74 of file der_evaluator_n.h.

10.154.3.5 coco::prep_d_eval::prep_d_eval (const prep_d_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 84 of file der_evaluator_n.h.

10.154.3.6 coco::prep_d_eval::~prep_d_eval () [inline]

Standard Destructor

Definition at line 87 of file der_evaluator_n.h.

10.154.4 Member Function Documentation

10.154.4.1 void coco::prep_d_eval::calculate (const expression_node & *__data*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 110 of file der_evaluator_n.h.

10.154.4.2 void coco::prep_d_eval::calculate (const expression_node & *__data*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from 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.

10.154.4.3 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::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 117 of file der_evaluator_n.h.

10.154.4.4 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::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 117 of file der_evaluator.h.

10.154.4.5 virtual void coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::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 804 of file evaluator.h.

10.154.4.6 int coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.154.4.7 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::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 108 of file der_evaluator_n.h.

10.154.4.8 int coco::prep_d_eval::initialize (const expression_node & *__data*) [inline, virtual]

Initialize method for normal nodes

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 96 of file der_evaluator_n.h.

10.154.4.9 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::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 108 of file der_evaluator.h.

10.154.4.10 int coco::prep_d_eval::initialize (const expression_node & *__data*) [inline, virtual]

Initialize method for normal nodes

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 96 of file der_evaluator.h.

10.154.4.11 bool coco::prep_d_eval::is_cached (const expression_node & *__data*) [inline, virtual]

Check whether this node has already been visited

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 90 of file der_evaluator_n.h.

10.154.4.12 bool coco::prep_d_eval::is_cached (const expression_node & *__data*) [inline, virtual]

Check whether this node has already been visited

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 90 of file der_evaluator.h.

10.154.4.13 void coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.154.4.14 int coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.154.4.15 void coco::prep_d_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 106 of file der_evaluator_n.h.

10.154.4.16 void coco::prep_d_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool, expression_const_walker >.

Definition at line 106 of file der_evaluator.h.

10.154.4.17 virtual const_walker coco::cached_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::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 450 of file evaluator.h.

10.154.4.18 virtual int coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::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 826 of file evaluator.h.

10.154.4.19 virtual int coco::cached_forward_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::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 814 of file evaluator.h.

10.154.4.20 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 114 of file der_evaluator_n.h.

10.154.4.21 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 112 of file der_evaluator_n.h.

10.154.4.22 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 114 of file der_evaluator.h.

10.154.4.23 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 112 of file der_evaluator.h.

10.154.4.24 return_value coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter false.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.154.4.25 int coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.154.4.26 void coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node , bool , expression_const_walker >::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 771 of file evaluator.h.

10.154.4.27 return_value coco::cached_forward_evaluator_base< std::vector< std::vector< double > > *, expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< double > > *, expression_node, bool,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.154.5 Member Data Documentation

10.154.5.1 std::vector< std::vector< double > > * coco::_evaluator_base< std::vector< std::vector< double > > *, expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.154.5.2 const variable_indicator* coco::cached_evaluator_base< std::vector< std::vector< double > > * , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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 files:

- der_evaluator.h
- der_evaluator_n.h

10.155 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
- typedef std::vector< std::vector< interval > > * data_type

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 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 const_walker short_cut_to (const node_data_type &__data)

- 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)

Protected Attributes

- const variable_indicator * v_ind
- std::vector< std::vector< interval > > * eval_data

10.155.1 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.

Definition at line 63 of file ider_evaluator.h.

10.155.2 Member Typedef Documentation

10.155.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.155.2.2 typedef std::vector< std::vector< interval > > * coco::_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.155.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.155.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.155.3 Constructor & Destructor Documentation

10.155.3.1 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 76 of file ider_evaluator.h.

10.155.3.2 coco::prep_id_eval::prep_id_eval (const prep_id_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 86 of file ider_evaluator.h.

10.155.3.3 coco::prep_id_eval::~prep_id_eval () [inline]

Standard Destructor

Definition at line 89 of file ider_evaluator.h.

10.155.4 Member Function Documentation

10.155.4.1 void coco::prep_id_eval::calculate (const expression_node & *__data*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 112 of file ider_evaluator.h.

10.155.4.2 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::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 119 of file ider_evaluator.h.

10.155.4.3 virtual void coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::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 804 of file evaluator.h.

10.155.4.4 int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.155.4.5 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::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 110 of file ider_evaluator.h.

10.155.4.6 int coco::prep_id_eval::initialize (const expression_node & *__data*) [inline, virtual]

Initialize method for normal nodes

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 98 of file ider_evaluator.h.

10.155.4.7 bool coco::prep_id_eval::is_cached (const expression_node & *__data*) [inline, virtual]

Check whether this node has already been visited

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 92 of file ider_evaluator.h.

10.155.4.8 void coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.155.4.9 int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.155.4.10 void coco::prep_id_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 108 of file ider_evaluator.h.

10.155.4.11 virtual const_walker coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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 450 of file evaluator.h.

10.155.4.12 virtual int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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 826 of file evaluator.h.

10.155.4.13 virtual int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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 814 of file evaluator.h.

10.155.4.14 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 116 of file ider_evaluator.h.

10.155.4.15 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 114 of file ider_evaluator.h.

10.155.4.16 return_value coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.155.4.17 int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.155.4.18 void coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::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 771 of file evaluator.h.

10.155.4.19 return_value coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.155.5 Member Data Documentation

10.155.5.1 std::vector< std::vector< interval > > * coco::_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.155.5.2 const variable_indicator* coco::cached_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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:

- ider_evaluator.h

10.156 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
- typedef std::vector< std::vector< interval > > * data_type

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 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 const_walker short_cut_to (const node_data_type &__data)

- 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)

Protected Attributes

- const variable_indicator * v_ind
- std::vector< std::vector< interval > > * eval_data

10.156.1 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.

Definition at line 74 of file islp_evaluator.h.

10.156.2 Member Typedef Documentation

10.156.2.1 typedef _Base::const_walker coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::const_walker [inherited]

This is the type of the walker, which is used for the short-cuts.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 723 of file evaluator.h.

10.156.2.2 typedef std::vector< std::vector< interval > > * coco::_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::data_type [inherited]

The data_type specifies the type of the internal data of the evaluator.

Definition at line 305 of file evaluator.h.

10.156.2.3 typedef _Base::node_data_type coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::node_data_type [inherited]

The node_data_type is the datatype of the nodes of the graph.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 719 of file evaluator.h.

10.156.2.4 typedef _Base::return_value coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::return_value [inherited]

This type is the result type of the evaluator.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 721 of file evaluator.h.

10.156.3 Constructor & Destructor Documentation

10.156.3.1 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 87 of file islp_evaluator.h.

10.156.3.2 coco::prep_islp_eval::prep_islp_eval (const prep_islp_eval & *__x*) [inline]

Standard Copy Constructor

Definition at line 97 of file islp_evaluator.h.

10.156.3.3 coco::prep_islp_eval::~prep_islp_eval () [inline]

Standard Destructor

Definition at line 100 of file islp_evaluator.h.

10.156.4 Member Function Documentation

10.156.4.1 void coco::prep_islp_eval::calculate (const expression_node & *__data*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from 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.

10.156.4.2 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::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 130 of file islp_evaluator.h.

10.156.4.3 virtual void coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::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 804 of file evaluator.h.

10.156.4.4 int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::collect (const node_data_type & *__data*, const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 750 of file evaluator.h.

10.156.4.5 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::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 119 of file islp_evaluator.h.

10.156.4.6 int coco::prep_islp_eval::initialize (const expression_node & *__data*) [inline, virtual]

Initialize method for normal nodes

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 109 of file islp_evaluator.h.

10.156.4.7 bool coco::prep_islp_eval::is_cached (const expression_node & *__data*) [inline, virtual]

Check whether this node has already been visited

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 103 of file islp_evaluator.h.

10.156.4.8 void coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::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. It is translated into calls to calculate and cleanup.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 744 of file evaluator.h.

10.156.4.9 int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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. It checks whether the result of this node is cached and calls either retrieve_from_cache and stops the downwards walk or calls initialize.

Reimplemented from coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 732 of file evaluator.h.

10.156.4.10 void coco::prep_islp_eval::retrieve_from_cache (const expression_node & *__data*) [inline, virtual]

The retrieve_from_cache, initialize, calculate, calculate_value, and update methods are basically NOP versions.

Reimplemented from coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool, expression_const_walker >.

Definition at line 121 of file islp_evaluator.h.

10.156.4.11 virtual const_walker coco::cached_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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 450 of file evaluator.h.

10.156.4.12 virtual int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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 826 of file evaluator.h.

10.156.4.13 virtual int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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 814 of file evaluator.h.

10.156.4.14 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 127 of file islp_evaluator.h.

10.156.4.15 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 125 of file islp_evaluator.h.

10.156.4.16 return_value coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > *, expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `false`.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 762 of file evaluator.h.

10.156.4.17 int coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::vcollect (const return_value & *__rval*) [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. It is translated to a call to update.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 756 of file evaluator.h.

10.156.4.18 void coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::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 771 of file evaluator.h.

10.156.4.19 return_value coco::cached_forward_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::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. It is translated to a call to calculate_value with parameter `true`.

Reimplemented from coco::_evaluator_base< std::vector< std::vector< interval > > *, expression_node, bool,expression_const_walker >.

Definition at line 767 of file evaluator.h.

10.156.5 Member Data Documentation

10.156.5.1 std::vector< std::vector< interval > > * coco::_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::eval_data [protected, inherited]

The internal data of the evaluator

Definition at line 316 of file evaluator.h.

10.156.5.2 const variable_indicator* coco::cached_evaluator_base< std::vector< std::vector< interval > > * , expression_node , bool , expression_const_walker >::v_ind [protected, inherited]

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:

- islp_evaluator.h

10.157 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)

10.157.1 Detailed Description

Definition at line 36 of file prointerval.h.

10.157.2 Constructor & Destructor Documentation

10.157.2.1 coco::proj_rational::proj_rational (int *n* = 0, unsigned int *d* = 1) [inline]

Definition at line 49 of file prointerval.h.

10.157.2.2 coco::proj_rational::proj_rational (const proj_rational & *R*) [inline]

Definition at line 51 of file prointerval.h.

10.157.2.3 coco::proj_rational::~proj_rational () [inline]

Definition at line 53 of file prointerval.h.

10.157.3 Member Function Documentation

10.157.3.1 int coco::proj_rational::denominator () const [inline]

Definition at line 107 of file prointerval.h.

10.157.3.2 interval coco::proj_rational::ival () const [inline]

Definition at line 74 of file prointerval.h.

10.157.3.3 int coco::proj_rational::nominator () const [inline]

Definition at line 106 of file prointerval.h.

10.157.3.4 bool coco::proj_rational::operator!= (int *n*) const [inline]

Definition at line 96 of file prointerval.h.

10.157.3.5 bool coco::proj_rational::operator!= (const proj_rational & *R*) const [inline]

Definition at line 79 of file prointerval.h.

10.157.3.6 proj_rational coco::proj_rational::operator∗ (const proj_rational & *R*) const [inline]

Definition at line 96 of file prointerval.hpp.

10.157.3.7 proj_rational & coco::proj_rational::operator∗= (const proj_rational & *R*) [inline]

Definition at line 148 of file prointerval.hpp.

10.157.3.8 proj_rational coco::proj_rational::operator+ (const proj_rational & *R*) const [inline]

Definition at line 74 of file prointerval.hpp.

10.157.3.9 proj_rational & coco::proj_rational::operator+= (const proj_rational & *R*) [inline]

Definition at line 126 of file prointerval.hpp.

10.157.3.10 proj_rational coco::proj_rational::operator- (const proj_rational & *R*) const [inline]

Definition at line 85 of file prointerval.hpp.

10.157.3.11 proj_rational coco::proj_rational::operator- () const [inline]

Definition at line 61 of file prointerval.h.

10.157.3.12 proj_rational & coco::proj_rational::operator-= (const proj_rational & *R*) [inline]

Definition at line 137 of file prointerval.hpp.

10.157.3.13 proj_rational coco::proj_rational::operator/ (const proj_rational & *R*) const [inline]

Definition at line 109 of file prointerval.hpp.

10.157.3.14 proj_rational & coco::proj_rational::operator/= (const proj_rational & *R*) [inline]

Definition at line 161 of file prointerval.hpp.

10.157.3.15 bool coco::proj_rational::operator< (int *n*) const [inline]

Definition at line 98 of file prointerval.h.

10.157.3.16 bool coco::proj_rational::operator< (const proj_rational & *R*) const [inline]

Definition at line 82 of file prointerval.h.

10.157.3.17 bool coco::proj_rational::operator<= (int *n*) const [inline]

Definition at line 102 of file prointerval.h.

10.157.3.18 bool coco::proj_rational::operator<= (const proj_rational & *R*) const [inline]

Definition at line 88 of file prointerval.h.

10.157.3.19 proj_rational& coco::proj_rational::operator= (int *n*) [inline]

Definition at line 58 of file prointerval.h.

10.157.3.20 proj_rational& coco::proj_rational::operator= (const proj_rational & *R*) [inline]

Definition at line 55 of file prointerval.h.

10.157.3.21 bool coco::proj_rational::operator== (int *n*) const [inline]

Definition at line 94 of file prointerval.h.

10.157.3.22 bool coco::proj_rational::operator== (const proj_rational & *R*) const [inline]

Definition at line 76 of file prointerval.h.

10.157.3.23 bool coco::proj_rational::operator> (int *n*) const [inline]

Definition at line 100 of file prointerval.h.

10.157.3.24 bool coco::proj_rational::operator> (const proj_rational & *R*) const [inline]

Definition at line 85 of file prointerval.h.

10.157.3.25 bool coco::proj_rational::operator>= (int *n*) const [inline]

Definition at line 104 of file prointerval.h.

10.157.3.26 bool coco::proj_rational::operator>= (const proj_rational & *R*) const [inline]

Definition at line 91 of file prointerval.h.

10.157.3.27 proj_rational& coco::proj_rational::set (int *n*, unsigned int *d*) [inline]

Definition at line 108 of file prointerval.h.

10.157.4 Friends And Related Function Documentation

10.157.4.1 proj_rational abs (const proj_rational & *r*) [friend]

Definition at line 120 of file prointerval.h.

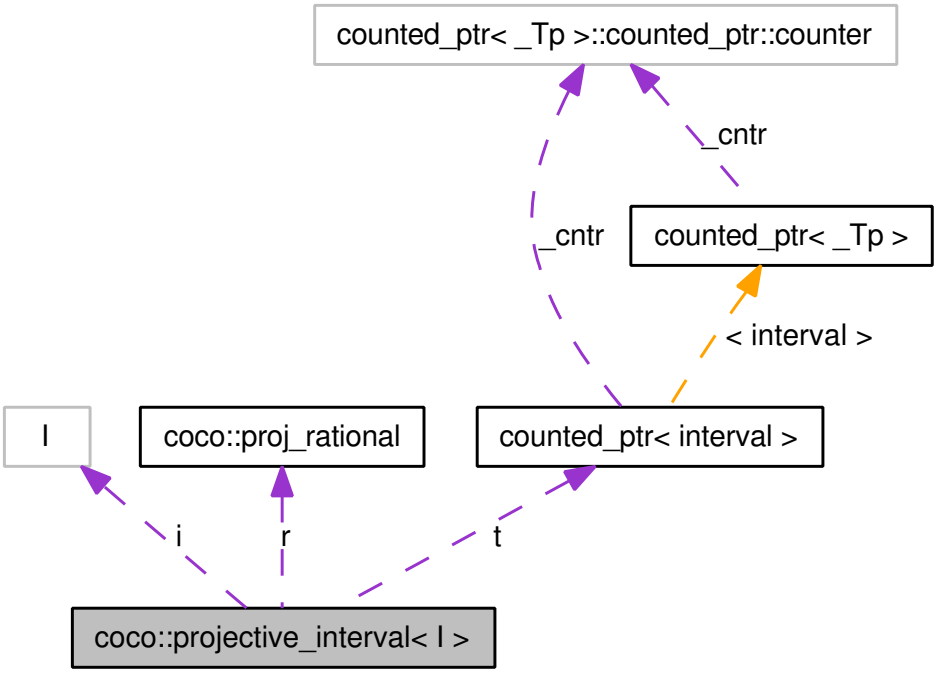
The documentation for this class was generated from the following files:

- prointerval.h
- prointerval.hpp

10.158 coco::projective_interval< I > Class Template Reference

`#include <prointerval.h>`

Collaboration diagram for coco::projective_interval< I >:



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, proj_rational _r, const interval &_t)
- projective_interval (const interval &_i, int _m, const proj_interval &_p)
- projective_interval (const interval &_i, proj_rational _r, const proj_interval &_p)
- projective_interval (double lo, double up, const proj_interval &_p)
- projective_interval (double d, 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)
- projective_interval (const interval &x, 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_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)
- 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

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 > 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)

10.158.1 Detailed Description

template<class I> class coco::projective_interval< I >

Definition at line 124 of file prointerval.h.

10.158.2 Constructor & Destructor Documentation

10.158.2.1 template<class I> coco::projective_interval< I >::projective_interval () [inline]

empty unusable projective interval

Definition at line 143 of file prointerval.h.

10.158.2.2 template<class I> coco::projective_interval< I >::projective_interval (const proj_interval & _p) [inline]

standard copy constructor

Definition at line 145 of file prointerval.h.

10.158.2.3 template<class I> coco::projective_interval< I >::projective_interval (const interval & _t) [inline]

constructors for the first projective interval

Definition at line 148 of file prointerval.h.

10.158.2.4 template<class I> coco::projective_interval< I >::projective_interval (const interval & _i, const interval & _t) [inline]

Definition at line 150 of file prointerval.h.

10.158.2.5 template<class I> coco::projective_interval< I >::projective_interval (const interval & _i, int _m, const interval & _t) [inline]

Definition at line 152 of file prointerval.h.

10.158.2.6 template<class I> coco::projective_interval< I >::projective_interval (const interval & _i, proj_rational _r, const interval & _t) [inline]

Definition at line 154 of file prointerval.h.

10.158.2.7 template<class I> coco::projective_interval< I >::projective_interval (const interval & _i, int _m, const proj_interval & _p) [inline]

constructors for dependant projective intervals

Definition at line 158 of file prointerval.h.

10.158.2.8 template<class I> coco::projective_interval< I >::projective_interval (const interval & *_i*, proj_rational *_r*, const proj_interval & *_p*) [inline]

Definition at line 160 of file prointerval.h.

10.158.2.9 template<class I> coco::projective_interval< I >::projective_interval (double *lo*, double *up*, const proj_interval & *_p*) [inline]

Definition at line 162 of file prointerval.h.

10.158.2.10 template<class I> coco::projective_interval< I >::projective_interval (double *d*, const proj_interval & *_p*) [inline]

Definition at line 164 of file prointerval.h.

10.158.2.11 template<class I> coco::projective_interval< I >::projective_interval (int *d*, const proj_interval & *_p*) [inline]

Definition at line 166 of file prointerval.h.

10.158.2.12 template<class I> coco::projective_interval< I >::projective_interval (unsigned *d*, const proj_interval & *_p*) [inline]

Definition at line 168 of file prointerval.h.

10.158.2.13 template<class I> coco::projective_interval< I >::projective_interval (long *d*, const proj_interval & *_p*) [inline]

Definition at line 170 of file prointerval.h.

10.158.2.14 template<class I> coco::projective_interval< I >::projective_interval (unsigned long *d*, const proj_interval & *_p*) [inline]

Definition at line 172 of file prointerval.h.

10.158.2.15 template<class I> coco::projective_interval< I >::projective_interval (const interval & *x*, const proj_interval & *_p*) [inline]

Definition at line 174 of file prointerval.h.

10.158.2.16 template<class I> virtual coco::projective_interval< I >::~projective_interval () [inline, virtual]

standard destructor

Definition at line 178 of file prointerval.h.

10.158.3 Member Function Documentation

10.158.3.1 template<class I > bool coco::projective_interval< I >::certain_proper_subset (const proj_interval & *x*) const [inline]

Definition at line 314 of file prointerval.hpp.

10.158.3.2 template<class I> bool coco::projective_interval< I >::certain_proper_superset (const proj_interval & *x*) const [inline]

Definition at line 238 of file prointerval.h.

10.158.3.3 template<class I > bool coco::projective_interval< I >::certain_subset (const proj_interval & *x*) const [inline]

Definition at line 296 of file prointerval.hpp.

10.158.3.4 template<class I> bool coco::projective_interval< I >::certain_superset (const proj_interval & *x*) const [inline]

Definition at line 224 of file prointerval.h.

10.158.3.5 template<class I> bool coco::projective_interval< I >::certainly_contains (const interval & *x*) const [inline]

Definition at line 254 of file prointerval.h.

10.158.3.6 template<class I > double coco::projective_interval< I >::diam () const [inline]

Definition at line 652 of file prointerval.hpp.

10.158.3.7 template<class I > bool coco::projective_interval< I >::disjoint (const proj_interval & *x*) const [inline]

Definition at line 332 of file prointerval.hpp.

10.158.3.8 template<class I > double coco::projective_interval< I >::dist (const proj_interval & *x*) const [inline]

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 689 of file prointerval.hpp.

10.158.3.9 template<class I> bool coco::projective_interval< I >::empty () const [inline]

Definition at line 205 of file prointerval.h.

10.158.3.10 template<class I> static proj_interval coco::projective_interval< I >::EMPTY (const proj_interval & *p*) [inline, static]

Definition at line 202 of file prointerval.h.

10.158.3.11 template<class I > projective_interval< I > & coco::projective_interval< I >::hull (const proj_interval & *x*) [inline]

Definition at line 373 of file prointerval.hpp.

10.158.3.12 template<class I > projective_interval< I > & coco::projective_interval< I >::hulldiff (const proj_interval & *x*) [inline]

Definition at line 391 of file prointerval.hpp.

10.158.3.13 template<class I> double coco::projective_interval< I >::i_inf () const [inline]

Definition at line 194 of file prointerval.h.

10.158.3.14 template<class I> bool coco::projective_interval< I >::i_proper_subset (const interval & *x*) const [inline]

Definition at line 228 of file prointerval.h.

10.158.3.15 template<class I> bool coco::projective_interval< I >::i_proper_superset (const proj_interval & *x*) const [inline]

Definition at line 234 of file prointerval.h.

10.158.3.16 template<class I> bool coco::projective_interval< I >::i_subset (const proj_interval & *x*) const [inline]

Definition at line 216 of file prointerval.h.

10.158.3.17 template<class I> double coco::projective_interval< I >::i_sup () const [inline]

Definition at line 195 of file prointerval.h.

10.158.3.18 template<class I> bool coco::projective_interval< I >::i_superset (const proj_interval & *x*) const [inline]

Definition at line 220 of file prointerval.h.

10.158.3.19 template<class I > projective_interval< I > & coco::projective_interval< I >::imax (const proj_interval & *x*) [inline]

The method changes this proj_interval to the maximum of itself and x.

Definition at line 432 of file prointerval.hpp.

10.158.3.20 template<class I > projective_interval< I > & coco::projective_interval< I >::imin (const proj_interval & *x*) [inline]

The method changes this proj_interval to the minimum of itself and x.

Definition at line 413 of file prointerval.hpp.

10.158.3.21 template<class I > projective_interval< I > & coco::projective_interval< I >::intersect_div (const proj_interval & *__i*, const proj_interval & *__j*) [inline]

Definition at line 460 of file prointerval.hpp.

10.158.3.22 template<class I > projective_interval< I > & coco::projective_interval< I >::intersect_invcos_wc (const proj_interval & *__i*, double *__l*, double *__d*) [inline]

Definition at line 517 of file prointerval.hpp.

10.158.3.23 template<class I > projective_interval< I > & coco::projective_interval< I >::intersect_invgauss_wc (const proj_interval & *__i*, double *__l*, double *__m*, double *__s*) [inline]

Definition at line 531 of file prointerval.hpp.

10.158.3.24 template<class I > projective_interval< I > & coco::projective_interval< I >::intersect_invpower_wc (const proj_interval & *__i*, double *__l*, int *__n*) [inline]

Definition at line 492 of file prointerval.hpp.

10.158.3.25 template<class I > projective_interval< I > & coco::projective_interval< I >::intersect_invsin_wc (const proj_interval & *__i*, double *__l*, double *__d*) [inline]

Definition at line 501 of file prointerval.hpp.

10.158.3.26 template<class I > projective_interval< I > & coco::projective_interval< I >::intersect_invsqr_wc (const proj_interval & *__i*, double *__l*, double *__d*) [inline]

Definition at line 483 of file prointerval.hpp.

10.158.3.27 template<class I > projective_interval< I > & coco::projective_interval< I >::intersect_power (const proj_interval & *__i*, int *__n*) [inline]

Definition at line 470 of file prointerval.hpp.

10.158.3.28 template<class I > projective_interval< I > & coco::projective_interval< I >::intersectwith (const proj_interval & *x*) [inline]

Definition at line 354 of file prointerval.hpp.

10.158.3.29 template<class I > projective_interval< I > & coco::projective_interval< I >::ipow (int *n*) [inline]

The method changes this interval to the n-th power.

Definition at line 405 of file prointerval.hpp.

10.158.3.30 template<class I> bool coco::projective_interval< I >::is_bounded () const [inline]

Definition at line 214 of file prointerval.h.

10.158.3.31 template<class I> bool coco::projective_interval< I >::is_empty () const [inline]

Definition at line 206 of file prointerval.h.

10.158.3.32 template<class I> bool coco::projective_interval< I >::is_entire () const [inline]

Definition at line 213 of file prointerval.h.

10.158.3.33 template<class I> bool coco::projective_interval< I >::is_i_thin () const [inline]

Definition at line 207 of file prointerval.h.

10.158.3.34 template<class I> bool coco::projective_interval< I >::is_t_thin () const [inline]

Definition at line 208 of file prointerval.h.

10.158.3.35 template<class I> bool coco::projective_interval< I >::is_unbounded_above () const [inline]

Definition at line 211 of file prointerval.h.

10.158.3.36 template<class I> bool coco::projective_interval< I >::is_unbounded_below () const [inline]

Definition at line 209 of file prointerval.h.

10.158.3.37 template<class I > double coco::projective_interval< I >::mag () const [inline]

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 683 of file prointerval.hpp.

10.158.3.38 template<class I > double coco::projective_interval< I >::mid () const [inline]

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 646 of file prointerval.hpp.

10.158.3.39 template<class I > double coco::projective_interval< I >::mig () const [inline]

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 677 of file prointerval.hpp.

10.158.3.40 template<class I > projective_interval< I > & coco::projective_interval< I >::operator*= (double *a*) [inline]

Definition at line 639 of file prointerval.hpp.

10.158.3.41 template<class I > projective_interval< I > & coco::projective_interval< I >::operator*= (const proj_interval & *a*) [inline]

Definition at line 631 of file prointerval.hpp.

10.158.3.42 template<class I > projective_interval< I > & coco::projective_interval< I >::operator+= (double *a*) [inline]

Definition at line 577 of file prointerval.hpp.

10.158.3.43 template<class I > projective_interval< I > & coco::projective_interval< I >::operator+= (const proj_interval & *a*) [inline]

Definition at line 559 of file prointerval.hpp.

10.158.3.44 template<class I> proj_interval coco::projective_interval< I >::operator- () const [inline]

Definition at line 277 of file prointerval.h.

10.158.3.45 template<class I > projective_interval< I > & coco::projective_interval< I >::operator-= (double *a*) [inline]

Definition at line 613 of file prointerval.hpp.

10.158.3.46 template<class I > projective_interval< I > & coco::projective_interval< I >::operator-= (const proj_interval & *a*) [inline]

Definition at line 595 of file prointerval.hpp.

10.158.3.47 template<class I> proj_interval& coco::projective_interval< I >::operator= (unsigned long *d*) [inline]

Definition at line 273 of file prointerval.h.

10.158.3.48 template<class I> proj_interval& coco::projective_interval< I >::operator= (long *d*) [inline]

Definition at line 272 of file prointerval.h.

10.158.3.49 template<class I> proj_interval& coco::projective_interval< I >::operator= (unsigned *d*) [inline]

Definition at line 271 of file prointerval.h.

10.158.3.50 template<class I> proj_interval& coco::projective_interval< I >::operator= (int *d*) [inline]

Definition at line 270 of file prointerval.h.

10.158.3.51 template<class I> proj_interval& coco::projective_interval< I >::operator= (double *d*) [inline]

Definition at line 269 of file prointerval.h.

10.158.3.52 template<class I> proj_interval& coco::projective_interval< I >::operator= (const proj_interval & *x*) [inline]

Definition at line 268 of file prointerval.h.

10.158.3.53 template<class I> const interval& coco::projective_interval< I >::pi () const [inline]

Definition at line 190 of file prointerval.h.

10.158.3.54 template<class I > bool coco::projective_interval< I >::possible_proper_subset (const proj_interval & *x*) const [inline]

Definition at line 323 of file prointerval.hpp.

10.158.3.55 template<class I> bool coco::projective_interval< I >::possible_proper_superset (const proj_interval & *x*) const [inline]

Definition at line 240 of file prointerval.h.

10.158.3.56 template<class I > bool coco::projective_interval< I >::possible_subset (const proj_interval & *x*) const [inline]

Definition at line 305 of file prointerval.hpp.

10.158.3.57 template<class I> bool coco::projective_interval< I >::possible_superset (const proj_interval & *x*) const [inline]

Definition at line 226 of file prointerval.h.

10.158.3.58 template<class I> const proj_rational& coco::projective_interval< I >::pr () const [inline]

Definition at line 192 of file prointerval.h.

10.158.3.59 template<class I> bool coco::projective_interval< I >::probably_contains (const interval & *x*) const [inline]

Definition at line 259 of file prointerval.h.

10.158.3.60 template<class I > double coco::projective_interval< I >::project (double *__d*) const [inline]

Definition at line 545 of file prointerval.hpp.

10.158.3.61 template<class I> const interval& coco::projective_interval< I >::pt () const [inline]

Definition at line 191 of file prointerval.h.

10.158.3.62 template<class I > double coco::projective_interval< I >::rad () const [inline]

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 671 of file prointerval.hpp.

10.158.3.63 template<class I> double coco::projective_interval< I >::rel_width () const [inline]

Definition at line 244 of file prointerval.h.

10.158.3.64 template<class I > double coco::projective_interval< I >::relDiam () const [inline]

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 665 of file prointerval.hpp.

10.158.3.65 template<class I > projective_interval< I > & coco::projective_interval< I >::round_to_integer () [inline]

This method rounds the proj_interval inward to integer borders.

Definition at line 451 of file prointerval.hpp.

10.158.3.66 template<class I > double coco::projective_interval< I >::safeguarded_mid () const [inline]

Returns the safeguarded mid of this proj_interval, i.e.

Definition at line 697 of file prointerval.hpp.

10.158.3.67 template<class I > void coco::projective_interval< I >::set (double *lo*, double *up*, const proj_interval & *_p*) [inline]

Definition at line 246 of file prointerval.hpp.

10.158.3.68 template<class I> void coco::projective_interval< I >::set_bounds (double *a*, double *b*) [inline]

Definition at line 264 of file prointerval.h.

10.158.3.69 template<class I > void coco::projective_interval< I >::set_i (const interval & *j*) [inline]

Definition at line 262 of file prointerval.hpp.

10.158.3.70 template<class I > void coco::projective_interval< I >::set_i (double *lo*, double *up*) [inline]

Definition at line 255 of file prointerval.hpp.

10.158.3.71 template<class I> void coco::projective_interval< I >::set_lb (double *d*) [inline]

Definition at line 265 of file prointerval.h.

10.158.3.72 template<class I > void coco::projective_interval< I >::set_r (const proj_rational & *s*) [inline]

Definition at line 290 of file prointerval.hpp.

10.158.3.73 template<class I > void coco::projective_interval< I >::set_r (int *n*, unsigned int *d*) [inline]

Definition at line 283 of file prointerval.hpp.

10.158.3.74 template<class I > void coco::projective_interval< I >::set_t (const interval & *u*) [inline]

Definition at line 276 of file prointerval.hpp.

10.158.3.75 template<class I > void coco::projective_interval< I >::set_t (double *lo*, double *up*) [inline]

Definition at line 269 of file prointerval.hpp.

10.158.3.76 template<class I> void coco::projective_interval< I >::set_ub (double *d*) [inline]

Definition at line 266 of file prointerval.h.

10.158.3.77 template<class I> double coco::projective_interval< I >::t_exp () const [inline]

Definition at line 198 of file prointerval.h.

10.158.3.78 template<class I> double coco::projective_interval< I >::t_exp_d () const [inline]

Definition at line 200 of file prointerval.h.

10.158.3.79 template<class I> double coco::projective_interval< I >::t_exp_n () const [inline]

Definition at line 199 of file prointerval.h.

10.158.3.80 template<class I> double coco::projective_interval< I >::t_inf () const [inline]

Definition at line 196 of file prointerval.h.

10.158.3.81 template<class I> bool coco::projective_interval< I >::t_proper_subset (const interval & *x*) const [inline]

Definition at line 230 of file prointerval.h.

10.158.3.82 template<class I> bool coco::projective_interval< I >::t_proper_superset (const proj_interval & *x*) const [inline]

Definition at line 236 of file prointerval.h.

10.158.3.83 template<class I> bool coco::projective_interval< I >::t_subset (const proj_interval & *x*) const [inline]

Definition at line 217 of file prointerval.h.

10.158.3.84 template<class I> double coco::projective_interval< I >::t_sup () const [inline]

Definition at line 197 of file prointerval.h.

10.158.3.85 template<class I> bool coco::projective_interval< I >::t_superset (const proj_interval & *x*) const [inline]

Definition at line 222 of file prointerval.h.

10.158.3.86 template<class I > double coco::projective_interval< I >::width () const [inline]

Definition at line 659 of file prointerval.hpp.

10.158.4 Friends And Related Function Documentation

10.158.4.1 template<class I> template<class J > projective_interval<J> abs (const projective_interval< J > & *x*) [friend]

10.158.4.2 template<class I> template<class J > projective_interval<J> acos (const projective_interval< J > & *x*) [friend]

10.158.4.3 template<class I> template<class J > projective_interval<J> acosh (const projective_interval< J > & *x*) [friend]

10.158.4.4 template<class I> template<class J > projective_interval<J> acot (const projective_interval< J > & *x*) [friend]

10.158.4.5 template<class I> template<class J > projective_interval<J> acoth (const projective_interval< J > & *x*) [friend]

10.158.4.6 template<class I> template<class J > projective_interval<J> asin (const projective_interval< J > & *x*) [friend]

10.158.4.7 template<class I> template<class J > projective_interval<J> asinh (const projective_interval< J > & *x*) [friend]

10.158.4.8 template<class I> template<class J > projective_interval<J> atan (const projective_interval< J > & *x*) [friend]

10.158.4.9 template<class I> template<class J > projective_interval<J> atanh (const projective_interval< J > & *x*) [friend]

10.158.4.10 template<class I> template<class J > projective_interval<J> cancel (const projective_interval< J > & *a*, const projective_interval< J > & *b*) [friend]

10.158.4.11 template<class I> template<class J > projective_interval<J> cos (const projective_interval< J > & *x*) [friend]

10.158.4.12 template<class I> template<class J > projective_interval<J> cosh (const projective_interval< J > & *x*) [friend]

10.158.4.13 template<class I> template<class J > projective_interval<J> cot (const projective_interval< J > & *x*) [friend]

10.158.4.14 template<class I> template<class J > projective_interval<J> coth (const projective_interval< J > & *x*) [friend]

10.158.4.15 template<class I> template<class J > projective_interval<J> division_part1 (const projective_interval< J > & *x*, const projective_interval< J > & *y*, bool & *b*) [friend]

10.158.4.16 template<class I> template<class J > projective_interval<J> division_part2 (const projective_interval< J > & *x*, const projective_interval< J > & *y*, bool *b*) [friend]

10.158.4.17 template<class I> template<class J > projective_interval<J> exp (const projective_interval< J > & *x*) [friend]

10.158.4.18 template<class I> template<class J > projective_interval<J> exp10 (const projective_interval< J > & *x*) [friend]

10.158.4.19 template<class I> template<class J > projective_interval<J> exp2 (const projective_interval< J > & *x*) [friend]

10.158.4.20 template<class I> template<class J > projective_interval<J> expm1 (const projective_interval< J > & *x*) [friend]

10.158.4.21 template<class I> template<class J > projective_interval<J> imax (const projective_interval< J > & *x*, const projective_interval< J > & *y*) [friend]

10.158.4.22 template<class I> template<class J > projective_interval<J> imin (const projective_interval< J > & *x*, const projective_interval< J > & *y*) [friend]

10.158.4.23 template<class I> template<class J > projective_interval<J> log (const projective_interval< J > & *x*) [friend]

10.158.4.24 template<class I> template<class J > projective_interval<J> log10 (const projective_interval< J > & *x*) [friend]

10.158.4.25 template<class I> template<class J > projective_interval<J> log1p (const projective_interval< J > & *x*) [friend]

10.158.4.26 template<class I> template<class J > projective_interval<J> log2 (const projective_interval< J > & *x*) [friend]

10.158.4.27 template<class I> template<class J > projective_interval<J> operator∗ (double *b*, const projective_interval< J > & *a*) [friend]

10.158.4.28 template<class I> template<class J > projective_interval<J> operator∗ (const projective_interval< J > & *a*, double *b*) [friend]

10.158.4.29 template<class I> template<class J > projective_interval<J> operator∗ (const projective_interval< J > & *a*, const projective_interval< J > & *b*) [friend]

10.158.4.30 template<class I> template<class J > projective_interval<J> operator+ (double *b*, const projective_interval< J > & *a*) [friend]

10.158.4.31 template<class I> template<class J > projective_interval<J> operator+ (const projective_interval< J > & *a*, double *b*) [friend]

10.158.4.32 template<class I> template<class J > projective_interval<J> operator+ (const projective_interval< J > & *a*, const projective_interval< J > & *b*) [friend]

10.158.4.33 template<class I> template<class J > projective_interval<J> operator- (double *b*, const projective_interval< J > & *a*) [friend]

10.158.4.34 template<class I> template<class J > projective_interval<J> operator- (const projective_interval< J > & *a*, double *b*) [friend]

10.158.4.35 template<class I> template<class J > projective_interval<J> operator- (const projective_interval< J > & *a*, const projective_interval< J > & *b*) [friend]

10.158.4.36 template<class I> template<class J > projective_interval<J> operator/ (double *b*, const projective_interval< J > & *a*) [friend]

10.158.4.37 template<class I> template<class J > projective_interval<J> operator/ (const projective_interval< J > & *a*, double *b*) [friend]

10.158.4.38 template<class I> template<class J > projective_interval<J> operator/ (const projective_interval< J > & *a*, const projective_interval< J > & *b*) [friend]

10.158.4.39 template<class I> template<class J > projective_interval<J> pow (const projective_interval< J > & *x*, const projective_interval< J > & *y*) [friend]

10.158.4.40 template<class I> template<class J > projective_interval<J> power (const projective_interval< J > & *x*, int *n*) [friend]

10.158.4.41 template<class I> template<class J > projective_interval<J> sin (const projective_interval< J > & *x*) [friend]

10.158.4.42 template<class I> template<class J > projective_interval<J> sinh (const projective_interval< J > & *x*) [friend]

10.158.4.43 template<class I> template<class J > projective_interval<J> sqr (const projective_interval< J > & *x*) [friend]

10.158.4.44 template<class I> template<class J > projective_interval<J> sqrt (const projective_interval< J > & *x*) [friend]

10.158.4.45 template<class I> template<class J > projective_interval<J> tan (const projective_interval< J > & *x*) [friend]

10.158.4.46 template<class I> template<class J > projective_interval<J> tanh (const projective_interval< J > & *x*) [friend]

The documentation for this class was generated from the following files:

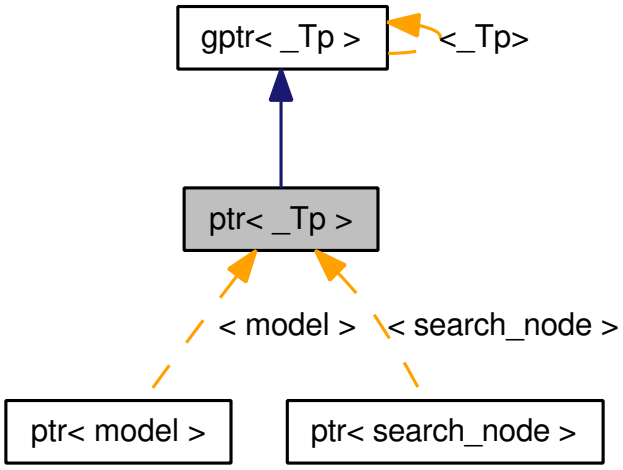
- prointerval.h
- prointerval.hpp

10.159 ptr< _Tp > Class Template Reference

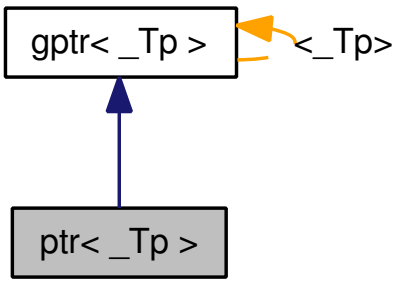
Local global pointer class.

`#include <gptr.h>`

Inheritance diagram for ptr< _Tp >:



Collaboration diagram for ptr< _Tp >:



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)

10.159.1 Detailed Description

template<class _Tp> class ptr< _Tp >

This class is a specialization of the global pointer class gptr, which points to a **local** object.

Definition at line 80 of file gptr.h.

10.159.2 Constructor & Destructor Documentation

10.159.2.1 template<class _Tp> ptr< _Tp >::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.

10.159.2.2 template<class _Tp> ptr< _Tp >::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.

10.159.2.3 template<class _Tp> ptr< _Tp >::ptr (const _Self & __p) [inline]

Standard Copy Constructor

Definition at line 104 of file gptr.h.

10.159.2.4 template<class _Tp> ptr< _Tp >::~ptr () [inline]

Standard Destructor

Definition at line 107 of file gptr.h.

10.159.3 Member Function Documentation

10.159.3.1 template<class _Tp> const_pointer ptr< _Tp >::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.

10.159.3.2 template<class _Tp> pointer ptr< _Tp >::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.

10.159.3.3 template<class _Tp> const_reference ptr< _Tp >::operator* () const [inline]

This is the const dereferentiation operator.

Definition at line 112 of file gptr.h.

10.159.3.4 template<class _Tp> reference ptr< _Tp >::operator* () [inline]

This is the dereferentiation operator.

Definition at line 110 of file gptr.h.

10.159.3.5 template<class _Tp> const_pointer ptr< _Tp >::operator → () const [inline]

This is the const pointer operator.

Definition at line 117 of file gptr.h.

10.159.3.6 template<class _Tp> pointer ptr< _Tp >::operator → () [inline]

This is the pointer operator.

Definition at line 115 of file gptr.h.

10.159.3.7 template<class _Tp> _Self& ptr< _Tp >::operator= (_Tp & *__p*) [inline]

This operator assigns a ptr object from a data structure ___p.

Definition at line 129 of file gptr.h.

10.159.3.8 template<class _Tp> _Self& ptr< _Tp >::operator= (const _Self & *__p*) [inline]

Standard Assignment Operator

Definition at line 127 of file gptr.h.

The documentation for this class was generated from the following file:

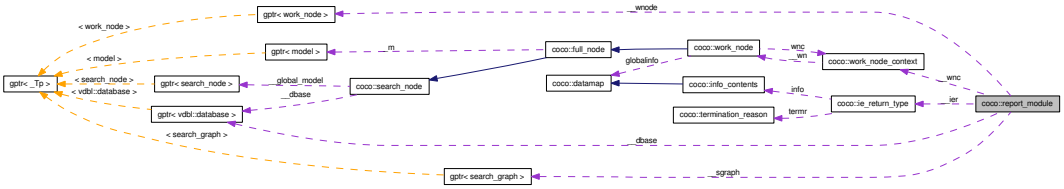
- gptr.h

10.160 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

10.160.1 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.

Definition at line 74 of file report_module.h.

10.160.2 Constructor & Destructor Documentation

10.160.2.1 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.

10.160.2.2 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.

10.160.2.3 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.

10.160.2.4 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.

10.160.2.5 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.

10.160.2.6 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.

10.160.2.7 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.

10.160.2.8 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.

10.160.2.9 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.

10.160.2.10 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.

10.160.2.11 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.

10.160.2.12 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.

10.160.2.13 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.

10.160.2.14 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.

10.160.2.15 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.

10.160.2.16 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.

10.160.2.17 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.

10.160.2.18 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.

10.160.2.19 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.

10.160.2.20 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.

10.160.2.21 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.

10.160.2.22 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.

10.160.2.23 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.

10.160.2.24 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.

10.160.2.25 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.

10.160.2.26 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.

10.160.2.27 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.

10.160.2.28 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.

10.160.2.29 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.

10.160.2.30 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.

10.160.2.31 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.

10.160.2.32 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.

10.160.2.33 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.

10.160.2.34 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.

10.160.2.35 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.

10.160.2.36 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.

10.160.2.37 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.

10.160.2.38 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.

10.160.2.39 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.

10.160.2.40 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.

10.160.2.41 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.

10.160.2.42 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.

10.160.2.43 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.

10.160.2.44 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.

10.160.2.45 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.

10.160.2.46 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.

10.160.2.47 virtual coco::report_module::~report_module () [inline, virtual]

Standard Destructor

Definition at line 718 of file report_module.h.

10.160.3 Member Function Documentation

10.160.3.1 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.

10.160.3.2 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.

10.160.3.3 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.

10.160.4 Member Data Documentation

10.160.4.1 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.

10.160.4.2 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.

10.160.4.3 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.

10.160.4.4 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.

10.160.4.5 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.

10.160.4.6 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.

10.160.4.7 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.

10.160.4.8 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:

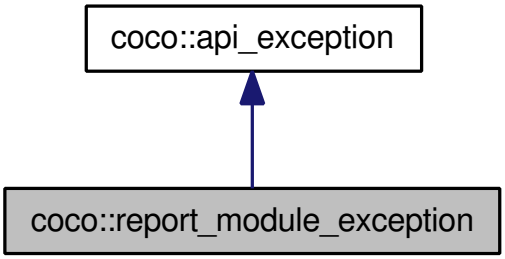
- report_module.h

10.161 coco::report_module_exception Class Reference

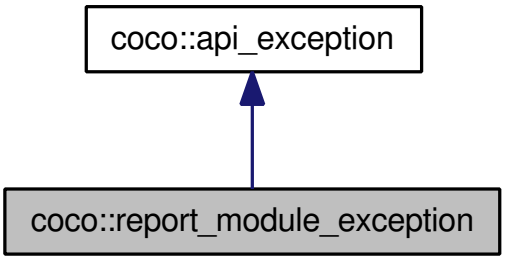
Report module exception class.

`#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 api_exception_type type () const throw ()
- virtual const char * type_str () const throw ()
- virtual std::string message () const throw ()

10.161.1 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.

Definition at line 54 of file report_module.h.

10.161.2 Constructor & Destructor Documentation

10.161.2.1 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.

10.161.2.2 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.

10.161.2.3 virtual coco::report_module_exception::~report_module_exception () throw () [inline, virtual]

Standard Destructor

Definition at line 65 of file report_module.h.

10.161.3 Member Function Documentation

10.161.3.1 virtual std::string coco::api_exception::message () const throw () [inline, virtual, inherited]

This method returns the message as C++-string.

Definition at line 104 of file api_exception.h.

10.161.3.2 virtual api_exception_type coco::api_exception::type () const throw () [inline, virtual, inherited]

This method returns the exception type as enum value.

Definition at line 95 of file api_exception.h.

10.161.3.3 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.

10.161.3.4 virtual char const* coco::api_exception::what () const throw () [inline, virtual, inherited]

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:

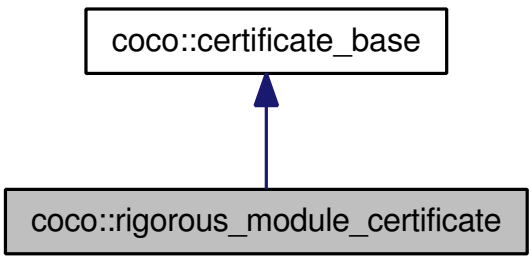
- report_module.h

10.162 coco::rigorous_module_certificate Class Reference

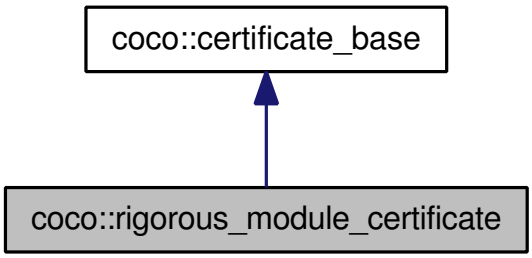
The certificate for deltas computed by rigorous inference engines.

`#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)

Protected Attributes

- std::string _contents

10.162.1 Detailed Description

The rigorous module certificate for inference engines, which do not produce certificates but rather always work in a rigorous mode!

Definition at line 262 of file api_cert.h.

10.162.2 Constructor & Destructor Documentation

10.162.2.1 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.

10.162.2.2 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.

10.162.2.3 virtual coco::rigorous_module_certificate::~rigorous_module_certificate () [inline, virtual]

Standard Destructor

Definition at line 282 of file api_cert.h.

10.162.3 Member Function Documentation

10.162.3.1 void coco::rigorous_module_certificate::destroy_copy (certificate_base * __c) const [inline]

Clone Destructor

Definition at line 293 of file api_cert.h.

10.162.3.2 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::certificate_base.

Definition at line 303 of file api_cert.h.

10.162.3.3 certificate 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 api_certbase.h.

10.162.3.4 rigorous_module_certificate* coco::rigorous_module_certificate::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::certificate_base.

Definition at line 285 of file api_cert.h.

10.162.3.5 bool coco::rigorous_module_certificate::operator!= (const rigorous_module_certificate & _c) const [inline]

Definition at line 317 of file api_cert.h.

10.162.3.6 bool coco::rigorous_module_certificate::operator!= (const certificate_base & _c) const [inline]

Definition at line 310 of file api_cert.h.

10.162.3.7 bool coco::rigorous_module_certificate::operator== (const rigorous_module_certificate & _c) const [inline]

Comparison operators

Definition at line 315 of file api_cert.h.

10.162.3.8 bool coco::rigorous_module_certificate::operator== (const certificate_base & _c) const [inline]

Comparison operators

Definition at line 307 of file api_cert.h.

10.162.3.9 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::certificate_base.

Definition at line 298 of file api_cert.h.

10.162.4 Member Data Documentation

10.162.4.1 std::string coco::certificate_base::_contents [protected, inherited]

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_cert.h

10.163 coco::search_graph Class Reference

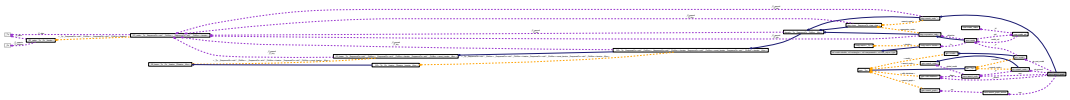
The search graph.

`#include <search_graph.h>`

Inheritance diagram for coco::search_graph:



Collaboration diagram for coco::search_graph:



Public Types

- typedef **_Base::walker walker**
- typedef **_Base::const_walker const_walker**
- typedef **_Base::children_iterator children_iterator**
- typedef **_Base::parents_iterator parents_iterator**
- typedef **_Base::children_const_iterator children_const_iterator**
- typedef **_Base::parents_const_iterator parents_const_iterator**
- typedef **_Base::erased_part erased_part**
- typedef _SequenceCtr< void *, _PtrAlloc > **container_type**
- typedef **_DG_iterator**< _Tp, _Tp &, _Tp *, **container_type**, **children_iterator**, **children_const_iterator** > **iterator**
- typedef **_DG_iterator**< _Tp, const _Tp &, const _Tp *, **container_type**, **children_iterator**, **children_const_iterator** > **const_iterator**

- typedef std::reverse_iterator< **const_iterator** > **const_reverse_iterator**
- typedef std::reverse_iterator< **iterator** > **reverse_iterator**
- typedef std::pair< **walker**, **walker** > **edge**
- typedef std::pair< **edge**, bool > **enhanced_edge**
- typedef _Tp **value_type**
- typedef _Node **node_type**
- typedef **value_type** * **pointer**
- typedef const **value_type** * **const_pointer**
- typedef **value_type** & **reference**
- typedef const **value_type** & **const_reference**
- typedef size_t **size_type**
- typedef ptrdiff_t **difference_type**

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
- bool register_hook (const work_node_comp_hook &wnch)
- bool unregister_hook (const std::string &name)
- void apply_hooks (const work_node &w, const search_node_id &id, vdbl::rowid &rid, std::list< delta > &add_ds, std::list< certificate > &add_cs)
- bool **check_acyclicity** (const **walker** &__parent, const **walker** &__child)
- void **clear** ()
- **walker between** (const **walker** &__parent, const **children_iterator** &__cit, const **walker** &__child, const **parents_iterator** &__pit, const _Tp &__x)
- **walker between** (const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children, const _Tp &__x)
- **walker between** (const **walker** &__parent, const **children_iterator** &__cit, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker between** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const **parents_iterator** &__pit, const _Tp &__x)
- **walker split** (const **walker** &__parent, const **children_iterator** &__ch_it, const **walker** &__child, const **parents_iterator** &__pa_it, const _Tp &__x)
- void **split** (const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children, const _Tp &__x)
- **walker split** (const **walker** &__parent, const **children_iterator** &__ch_it, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker split** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const **parents_iterator** &__pr_it, const _Tp &__x)
- **walker between_back** (const **walker** &__parent, const **walker** &__child, const _Tp &__x)
- **walker between_back** (const **walker** &__parent, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker between_back** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const _Tp &__x)

- **walker split_back** (const **walker** &__parent, const **walker** &__child, const _Tp &__x)
- **walker split_back** (const **walker** &__parent, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker split_back** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const _Tp &__x)
- **walker between_front** (const **walker** &__parent, const **walker** &__child, const _Tp &__x)
- **walker between_front** (const **walker** &__parent, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker between_front** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const _Tp &__x)
- **walker split_front** (const **walker** &__parent, const **walker** &__child, const _Tp &__x)
- **walker split_front** (const **walker** &__parent, const __SequenceCtr< **walker**, _Allocator > &__children, const _Tp &__x)
- **walker split_front** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const _Tp &__x)
- void **insert_subgraph** (_Self &__subgraph, const **walker** &__parent, const **children_iterator** &__ch_it, const **walker** &__child, const **parents_iterator** &__pa_it)
- void **insert_subgraph** (_Self &__subgraph, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **insert_subgraph** (_Self &__subgraph, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- void **insert_subgraph** (_Self &__subgraph, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **insert_subgraph** (_Self &__subgraph, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- void **insert_back_subgraph** (_Self &__subgraph, const **walker** &__parent, const **walker** &__child)
- void **insert_front_subgraph** (_Self &__subgraph, const **walker** &__parent, const **walker** &__child)
- void **add_edge** (const **walker** &__parent, const **children_iterator** &__ch_it, const **walker** &__child, const **parents_iterator** &__pa_it)
- void **add_edge** (const **edge** &__edge, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **add_edge** (const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **add_edge** (const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- void **add_edge_back** (const **walker** &__parent, const **walker** &__child)
- void **add_edge_front** (const **walker** &__parent, const **walker** &__child)
- **allocator_type get_allocator** () const
- **walker ground** ()
- **const_walker ground** () const
- **walker sky** ()
- **const_walker sky** () const
- **children_iterator root_begin** ()
- **children_const_iterator root_begin** () const
- **children_iterator root_end** ()
- **children_const_iterator root_end** () const
- **parents_iterator leaf_begin** ()
- **parents_const_iterator leaf_begin** () const
- **parents_iterator leaf_end** ()
- **parents_const_iterator leaf_end** () const

- bool **empty** () const
- **size_type size** () const
- **size_type max_size** () const
- void **swap** (_Self &__x)
- **walker insert_node_in_graph** (_Node *__n, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_node_in_graph** (_Node *__node, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_node_in_graph** (_Node *__node, const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_node_in_graph** (_Node *__node, const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_node_in_graph** (_Node *__n, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_node_in_graph** (_Node *__node, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_node_in_graph** (_Node *__node, const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_node_in_graph** (_Node *__node, const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_in_graph** (const _Tp &__x, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_in_graph** (const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_in_graph** (const _Tp &__x, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_in_graph** (const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_in_graph** (const _Tp &__x, const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_in_graph** (const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_in_graph** (const _Tp &__x, const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_in_graph** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_in_graph** (const _Tp &__x, const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_in_graph** (const **walker** &__parent, const **walker** &__child, const container_insert_arg &__Itc, const container_insert_arg &__Itp)
- **walker insert_in_graph** (const _Tp &__x, const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_in_graph** (const __SequenceCtr1< **walker**, _Allocator1 > &__parents, const __SequenceCtr2< **walker**, _Allocator2 > &__children)
- **walker insert_in_graph** (const _Tp &__x, const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_in_graph** (const **walker** &__parent, const container_insert_arg &__pref, const __SequenceCtr< **walker**, _Allocator > &__children)
- **walker insert_in_graph** (const _Tp &__x, const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)
- **walker insert_in_graph** (const __SequenceCtr< **walker**, _Allocator > &__parents, const **walker** &__child, const container_insert_arg &__cref)

- void **replace_edge_to_child** (const **walker** &__parent, const **walker** &__child_old, const **walker** &__child_new)
- void **replace_edge_to_child** (const **walker** &__parent, const **walker** &__child_old, const **walker** &__child_new)
- void **replace_edge_to_parent** (const **walker** &__parent_old, const **walker** &__parent_new, const **walker** &__child)
- void **replace_edge_to_parent** (const **walker** &__parent_old, const **walker** &__parent_new, const **walker** &__child)
- void **remove_edge** (const **edge** &__edge)
- void **remove_edge** (const **walker** &__parent, const **walker** &__child)
- void **remove_edge** (const **walker** &__parent, const **walker** &__child)
- void **remove_edge_and_deattach** (const **walker** &__parent, const **walker** &__child)
- void **remove_edge_and_deattach** (const **walker** &__parent, const **walker** &__child)
- void **sort_child_edges** (**walker** __position, **children_iterator** first, **children_iterator** last, Compare comp)
- void **sort_child_edges** (**walker** __position, Compare comp)
- void **sort_child_edges** (**walker** __position, **children_iterator** first, **children_iterator** last, Compare comp)
- void **sort_child_edges** (**walker** __position, Compare comp)
- void **sort_parent_edges** (**walker** __position, **parents_iterator** first, **parents_iterator** last, Compare comp)
- void **sort_parent_edges** (**walker** __position, Compare comp)
- void **sort_parent_edges** (**walker** __position, **parents_iterator** first, **parents_iterator** last, Compare comp)
- void **sort_parent_edges** (**walker** __position, Compare comp)
- **walker insert_node** (_Node *_node, const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (const _Tp &__x, const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (_Node *_node, const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (const _Tp &__x, const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node** (const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node_before** (_Node *_node, const **walker** &__position, const container_insert_arg &__It)
- void **insert_node_before** (const _Tp &__x, const **walker** &__position, const container_insert_arg &__It)
- void **insert_node_before** (const **walker** &__position, const container_insert_arg &__It)
- **walker insert_node_before** (_Node *_node, const **walker** &__position, const container_insert_arg &__It)
- void **insert_node_before** (const _Tp &__x, const **walker** &__position, const container_insert_arg &__It)
- void **insert_node_before** (const **walker** &__position, const container_insert_arg &__It)
- void **merge** (const **walker** &__position, const **walker** &__second, bool merge_parent_edges=true, bool merge_child_edges=true)
- void **merge** (const **walker** &__position, const **walker** &__second, bool merge_parent_edges=true, bool merge_child_edges=true)
- void **erase** (const **walker** &__position)
- void **erase** (const **walker** &__position)
- void **partial_erase_to_parent** (const **walker** &__position, const **walker** &__parent, unsigned int idx)
- void **partial_erase_to_parent** (const **walker** &__position, const **walker** &__parent, unsigned int idx)

- void **clear_erased_part** (**erased_part** &_ep)
- void **clear_erased_part** (**erased_part** &_ep)
- **erased_part erase_maximal_subgraph** (const **walker** &__position)
- **erased_part erase_maximal_subgraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_maximal_subgraph** (const **walker** &__position)
- **erased_part erase_maximal_subgraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_minimal_subgraph** (const **walker** &__position)
- **erased_part erase_minimal_subgraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_minimal_subgraph** (const **walker** &__position)
- **erased_part erase_minimal_subgraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_maximal_pregraph** (const **walker** &__position)
- **erased_part erase_maximal_pregraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_maximal_pregraph** (const **walker** &__position)
- **erased_part erase_maximal_pregraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_minimal_pregraph** (const **walker** &__position)
- **erased_part erase_minimal_pregraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- **erased_part erase_minimal_pregraph** (const **walker** &__position)
- **erased_part erase_minimal_pregraph** (const __SequenceCtr< **walker**, _Allocator > &__positions)
- bool **erase_child** (const **walker** &__position, const **children_iterator** &__It)
- bool **erase_child** (const **walker** &__position, const **children_iterator** &__It)
- bool **erase_parent** (const **walker** &__position, const **parents_iterator** &__It)
- bool **erase_parent** (const **walker** &__position, const **parents_iterator** &__It)

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 & 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

Protected Types

- typedef **_Base::allocator_type allocator_type**

Protected Member Functions

- _Node * **_C_create_node** (const _Tp &__x)
- _Node * **_C_create_node** ()
- void **_C_destroy_node** (_Node *__p)
- void **clear_graph** (**_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_node)
- void **clear_graph** (**_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_node)
- **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > * **_C_get_node** ()
- void **_C_put_node** (**_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *__p)
- void **_C_put_node** (**_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *__p)
- void **clear_children** ()
- void **clear_parents** ()
- void **add_all_children** (_Output_Iterator fi, **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_parent)

- void **add_all_children** (_Output_Iterator fi, **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_parent)
- void **add_all_parents** (_Output_Iterator fi, **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_child)
- void **add_all_parents** (_Output_Iterator fi, **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > *_child)

Protected Attributes

- **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > * **_C_ground**
- **_DG_node**< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::**iterator** > * **_C_sky**
- int **_C_mark**

10.163.1 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.

Definition at line 53 of file search_graph.h.

10.163.2 Member Typedef Documentation

10.163.2.1 typedef _Base::allocator_type dag< class, _SequenceCtr, class, class >::allocator_type [protected, inherited]

Reimplemented from **dgraph**< **_Tp, _SequenceCtr, _PtrAlloc, _Alloc** >.

10.163.2.2 typedef _Base::children_const_iterator dag< class, _SequenceCtr, class, class >::children_const_iterator [inherited]

Reimplemented from **dgraph**< **_Tp, _SequenceCtr, _PtrAlloc, _Alloc** >.

10.163.2.3 typedef _Base::children_iterator dag< class, _SequenceCtr, class, class >::children_iterator [inherited]

Reimplemented from **dgraph**< **_Tp, _SequenceCtr, _PtrAlloc, _Alloc** >.

10.163.2.4 typedef _DG_iterator< _Tp, const _Tp &, const _Tp *, container_type, children_iterator, children_const_iterator > dag< class, _SequenceCtr, class, class >::const_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.5 typedef const value_type * dag< class, _SequenceCtr, class, class >::const_pointer [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.6 typedef const value_type & dag< class, _SequenceCtr, class, class >::const_reference [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.7 typedef std::reverse_iterator< const_iterator > dag< class, _SequenceCtr, class, class >::const_reverse_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.8 typedef _Base::const_walker dag< class, _SequenceCtr, class, class >::const_walker [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.9 typedef _SequenceCtr< void *, _PtrAlloc > dag< class, _SequenceCtr, class, class >::container_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.10 typedef ptrdiff_t dag< class, _SequenceCtr, class, class >::difference_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.11 typedef std::pair< walker, walker > dag< class, _SequenceCtr, class, class >::edge [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.12 typedef std::pair< edge, bool > dag< class, _SequenceCtr, class, class >::enhanced_edge [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.13 typedef _Base::erased_part dag< class, _SequenceCtr, class, class >::erased_part [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.14 typedef _DG_iterator< _Tp, _Tp &, _Tp *, container_type, children_iterator, children_const_iterator > dag< class, _SequenceCtr, class, class >::iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.15 typedef _Node dag< class, _SequenceCtr, class, class >::node_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.16 typedef _Base::parents_const_iterator dag< class, _SequenceCtr, class, class >::parents_const_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.17 typedef _Base::parents_iterator dag< class, _SequenceCtr, class, class >::parents_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.18 typedef value_type * dag< class, _SequenceCtr, class, class >::pointer [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.19 typedef value_type & dag< class, _SequenceCtr, class, class >::reference [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.20 typedef std::reverse_iterator< iterator > dag< class, _SequenceCtr, class, class >::reverse_iterator [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.21 typedef size_t dag< class, _SequenceCtr, class, class >::size_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.22 typedef _Tp dag< class, _SequenceCtr, class, class >::value_type [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.2.23 typedef _Base::walker dag< class, _SequenceCtr, class, class >::walker [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.3 Constructor & Destructor Documentation

10.163.3.1 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.

10.163.3.2 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.

10.163.3.3 coco::search_graph::~search_graph () [inline]

Destructor

Definition at line 74 of file search_graph.hpp.

10.163.4 Member Function Documentation

10.163.4.1 _Node * dag< class, _SequenceCtr, class, class >::_C_create_node () [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.2 _Node * dag< class, _SequenceCtr, class, class >::_C_create_node (const _Tp & *__x*) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.3 void dag< class, _SequenceCtr, class, class >::_C_destroy_node (_Node * *__p*) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.4 _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * dag< class, _SequenceCtr, class, class >::_C_get_node () [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.5 void _DG_base< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _Alloc >::_C_put_node (_DG_node< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator > * *__p*) [protected, inherited]

10.163.4.6 void dag< class, _SequenceCtr, class, class >::_C_put_node (_DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * *__p*) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.7 void _DG_base< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _Alloc >::add_all_children (_Output_Iterator *fi*, _DG_node< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator > * *_parent*) [protected, inherited]

10.163.4.8 void dag< class, _SequenceCtr, class, class >::add_all_children (_Output_Iterator *fi*, _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * *_parent*) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.9 void _DG_base< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _Alloc >::add_all_parents (_Output_Iterator *fi*, _DG_node< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator > * *_child*) [protected, inherited]

10.163.4.10 void dag< class, _SequenceCtr, class, class >::add_all_parents (_Output_Iterator *fi*, _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * *_child*) [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.11 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 1064 of file search_graph.cc.

10.163.4.12 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::add_edge (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

10.163.4.13 void dag< class, _SequenceCtr, class, class >::add_edge (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.14 void dag< class, _SequenceCtr, class, class >::add_edge (const edge & *__edge*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.15 void dag< class, _SequenceCtr, class, class >::add_edge (const walker & *__parent*, const children_iterator & *__ch_it*, const walker & *__child*, const parents_iterator & *__pa_it*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.16 void dag< class, _SequenceCtr, class, class >::add_edge_back (const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.17 void dag< class, _SequenceCtr, class, class >::add_edge_front (const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.18 void coco::search_graph::apply_hooks (const work_node & *w*, const search_node_id & *id*, vdbl::rowid & *rid*, std::list< delta > & *add_ds*, std::list< certificate > & *add_cs*)

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 1134 of file search_graph.cc.

10.163.4.19 walker dag< class, _SequenceCtr, class, class >::between (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const parents_iterator & *__pit*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.20 walker dag< class, _SequenceCtr, class, class >::between (const walker & *__parent*, const children_iterator & *__cit*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.21 walker dag< class, _SequenceCtr, class, class >::between (const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.22 walker dag< class, _SequenceCtr, class, class >::between (const walker & *__parent*, const children_iterator & *__cit*, const walker & *__child*, const parents_iterator & *__pit*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.23 walker dag< class, _SequenceCtr, class, class >::between_back (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.24 walker dag< class, _SequenceCtr, class, class >::between_back (const walker & *__parent*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.25 walker dag< class, _SequenceCtr, class, class >::between_back (const walker & *__parent*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.26 walker dag< class, _SequenceCtr, class, class >::between_front (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.27 walker dag< class, _SequenceCtr, class, class >::between_front (const walker & *__parent*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.28 walker dag< class, _SequenceCtr, class, class >::between_front (const walker & *__parent*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.29 bool dag< class, _SequenceCtr, class, class >::check_acyclicity (const walker & *__parent*, const walker & *__child*) [inherited]

10.163.4.30 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.

10.163.4.31 void dag< class, _SequenceCtr, class, class >::clear () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.32 void dag< class, _SequenceCtr, class, class >::clear_children () [protected, inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.33 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::clear_erased_part (erased_part & *_ep*) [inherited]

10.163.4.34 void dag< class, _SequenceCtr, class, class >::clear_erased_part (erased_part & *_ep*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.35 void _DG_base< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _Alloc >::clear_graph (_DG_node< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator > * *_node*) [protected, inherited]

10.163.4.36 void dag< class, _SequenceCtr, class, class >::clear_graph (_DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * *_node*) [protected, inherited]

Reimplemented from **dgraph**< **_Tp, _SequenceCtr, _PtrAlloc, _Alloc** >.

10.163.4.37 void dag< class, _SequenceCtr, class, class >::clear_parents () [protected, inherited]

Reimplemented from **dgraph**< **_Tp, _SequenceCtr, _PtrAlloc, _Alloc** >.

10.163.4.38 void coco::search_graph::destroy_focus (const search_focus & *_sf*)

This method destroys the search focus `_sf`.

Definition at line 396 of file search_graph.cc.

10.163.4.39 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 437 of file search_graph.cc.

10.163.4.40 bool dag< class, _SequenceCtr, class, class >::empty () const [inherited]

Reimplemented from **dgraph**< **_Tp, _SequenceCtr, _PtrAlloc, _Alloc** >.

10.163.4.41 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase (const walker & *__position*) [inherited]

10.163.4.42 void dag< class, _SequenceCtr, class, class >::erase (const walker & *__position*) [inherited]

Reimplemented from **dgraph**< **_Tp, _SequenceCtr, _PtrAlloc, _Alloc** >.

10.163.4.43 bool __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_child (const walker & *__position*, const children_iterator & *__It*) [inherited]

10.163.4.44 bool dag< class, _SequenceCtr, class, class >::erase_child (const walker & *__position*, const children_iterator & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.45 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_maximal_pregraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

10.163.4.46 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_maximal_pregraph (const walker & *__position*) [inherited]

10.163.4.47 erased_part dag< class, _SequenceCtr, class, class >::erase_maximal_pregraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.48 erased_part dag< class, _SequenceCtr, class, class >::erase_maximal_pregraph (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.49 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_maximal_subgraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

10.163.4.50 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_maximal_subgraph (const walker & *__position*) [inherited]

10.163.4.51 erased_part dag< class, _SequenceCtr, class, class >::erase_maximal_subgraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.52 erased_part dag< class, _SequenceCtr, class, class >::erase_maximal_subgraph (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.53 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_minimal_pregraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

10.163.4.54 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_minimal_pregraph (const walker & *__position*) [inherited]

10.163.4.55 erased_part dag< class, _SequenceCtr, class, class >::erase_minimal_pregraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.56 erased_part dag< class, _SequenceCtr, class, class >::erase_minimal_pregraph (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.57 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_minimal_subgraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

10.163.4.58 erased_part __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_minimal_subgraph (const walker & *__position*) [inherited]

10.163.4.59 erased_part dag< class, _SequenceCtr, class, class >::erase_minimal_subgraph (const __SequenceCtr< walker, _Allocator > & *__positions*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.60 erased_part dag< class, _SequenceCtr, class, class >::erase_minimal_subgraph (const walker & *__position*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.61 bool __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::erase_parent (const walker & *__position*, const parents_iterator & *__It*) [inherited]

10.163.4.62 bool dag< class, _SequenceCtr, class, class >::erase_parent (const walker & *__position*, const parents_iterator & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.63 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 599 of file search_graph.cc.

10.163.4.64 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 587 of file search_graph.cc.

10.163.4.65 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 640 of file search_graph.cc.

10.163.4.66 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 611 of file search_graph.cc.

10.163.4.67 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.

10.163.4.68 allocator_type dag< class, _SequenceCtr, class, class >::get_allocator () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.69 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 375 of file search_graph.cc.

10.163.4.70 search_node_id coco::search_graph::get_node_id () [inline]

This method returns the next available search_node_id.

Definition at line 149 of file search_graph.h.

10.163.4.71 const search_graph_context* coco::search_graph::get_search_graph_context () const [inline]

This method returns the actual search graph context

Definition at line 155 of file search_graph.h.

10.163.4.72 search_graph_context* coco::search_graph::get_search_graph_context () [inline]

This method returns the actual search graph context

Definition at line 152 of file search_graph.h.

10.163.4.73 const vdbl::viewdbase& coco::search_graph::get_viewdbase () const [inline]

This method returns the actual view database.

Definition at line 161 of file search_graph.h.

10.163.4.74 vdbl::viewdbase& coco::search_graph::get_viewdbase () [inline]

This method returns the actual view database.

Definition at line 158 of file search_graph.h.

10.163.4.75 const_walker dag< class, _SequenceCtr, class, class >::ground () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.76 walker dag< class, _SequenceCtr, class, class >::ground () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.77 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 694 of file search_graph.cc.

10.163.4.78 void dag< class, _SequenceCtr, class, class >::insert_back_subgraph (_Self & *__subgraph*, const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.79 void dag< class, _SequenceCtr, class, class >::insert_front_subgraph (_Self & *__subgraph*, const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.80 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

10.163.4.81 **walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*)** [inherited]

10.163.4.82 **walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*)** [inherited]

10.163.4.83 **walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const _Tp & *__x*, const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*)** [inherited]

10.163.4.84 **walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*)** [inherited]

10.163.4.85 **walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*)** [inherited]

10.163.4.86 **walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*)** [inherited]

10.163.4.87 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_in_graph (const _Tp & *__x*, const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

10.163.4.88 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.89 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.90 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.91 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const _Tp & *__x*, const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.92 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.93 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const _Tp & *__x*, const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.94 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.95 walker dag< class, _SequenceCtr, class, class >::insert_in_graph (const _Tp & *__x*, const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.96 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node (const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.163.4.97 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node (const _Tp & *__x*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.163.4.98 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node (_Node * *_node*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.163.4.99 walker dag< class, _SequenceCtr, class, class >::insert_node (const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.100 walker dag< class, _SequenceCtr, class, class >::insert_node (const _Tp & *__x*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.101 walker dag< class, _SequenceCtr, class, class >::insert_node (_Node * *_node*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.102 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_before (const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.163.4.103 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_before (const _Tp & *__x*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.163.4.104 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_before (_Node * *_node*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

10.163.4.105 void dag< class, _SequenceCtr, class, class >::insert_node_before (const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.106 void dag< class, _SequenceCtr, class, class >::insert_node_before (const _Tp & *__x*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.107 walker dag< class, _SequenceCtr, class, class >::insert_node_before (_Node * *_node*, const walker & *__position*, const container_insert_arg & *__It*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.108 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_in_graph (_Node * *__node*, const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

10.163.4.109 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_in_graph (_Node * *__node*, const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inherited]

10.163.4.110 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_in_graph (_Node * *__node*, const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

10.163.4.111 walker __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_node_in_graph (_Node * *__n*, const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

10.163.4.112 walker dag< class, _SequenceCtr, class, class >::insert_node_in_graph (_Node * *__node*, const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const container_insert_arg & *__cref*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.113 walker dag< class, _SequenceCtr, class, class >::insert_node_in_graph (_Node * *__node*, const walker & *__parent*, const container_insert_arg & *__pref*, const __SequenceCtr< walker, _Allocator > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.114 walker dag< class, _SequenceCtr, class, class >::insert_node_in_graph (_Node * *__node*, const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.115 walker dag< class, _SequenceCtr, class, class >::insert_node_in_graph (_Node * *__n*, const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.116 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_subgraph (_Self & *__subgraph*, const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

10.163.4.117 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::insert_subgraph (_Self & *__subgraph*, const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

10.163.4.118 void dag< class, _SequenceCtr, class, class >::insert_subgraph (_Self & *__subgraph*, const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.119 void dag< class, _SequenceCtr, class, class >::insert_subgraph (_Self & *__subgraph*, const walker & *__parent*, const walker & *__child*, const container_insert_arg & *__Itc*, const container_insert_arg & *__Itp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.120 void dag< class, _SequenceCtr, class, class >::insert_subgraph (_Self & *__subgraph*, const walker & *__parent*, const children_iterator & *__ch_it*, const walker & *__child*, const parents_iterator & *__pa_it*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.121 parents_const_iterator dag< class, _SequenceCtr, class, class >::leaf_begin () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.122 parents_iterator dag< class, _SequenceCtr, class, class >::leaf_begin () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.123 parents_const_iterator dag< class, _SequenceCtr, class, class >::leaf_end () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.124 parents_iterator dag< class, _SequenceCtr, class, class >::leaf_end () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.125 size_type dag< class, _SequenceCtr, class, class >::max_size () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.126 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::merge (const walker & *__position*, const walker & *__second*, bool *merge_parent_edges* = true, bool *merge_child_edges* = true) [inherited]

10.163.4.127 void dag< class, _SequenceCtr, class, class >::merge (const walker & *__position*, const walker & *__second*, bool *merge_parent_edges* = true, bool *merge_child_edges* = true) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.128 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 363 of file search_graph.cc.

10.163.4.129 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 194 of file search_graph.h.

10.163.4.130 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 426 of file search_graph.cc.

10.163.4.131 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.

10.163.4.132 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::partial_erase_to_parent (const walker & *__position*, const walker & *__parent*, unsigned int *idx*) [inherited]

10.163.4.133 void dag< class, _SequenceCtr, class, class >::partial_erase_to_parent (const walker & *__position*, const walker & *__parent*, unsigned int *idx*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.134 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 319 of file search_graph.cc.

10.163.4.135 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 355 of file search_graph.cc.

10.163.4.136 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 878 of file search_graph.cc.

10.163.4.137 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 1080 of file search_graph.cc.

10.163.4.138 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 864 of file search_graph.cc.

10.163.4.139 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::remove_edge (const walker & *__parent*, const walker & *__child*) [inherited]

10.163.4.140 void dag< class, _SequenceCtr, class, class >::remove_edge (const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.141 void dag< class, _SequenceCtr, class, class >::remove_edge (const edge & *__edge*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.142 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::remove_edge_and_deattach (const walker & *__parent*, const walker & *__child*) [inherited]

10.163.4.143 void dag< class, _SequenceCtr, class, class >::remove_edge_and_deattach (const walker & *__parent*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.144 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 1013 of file search_graph.cc.

10.163.4.145 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 710 of file search_graph.cc.

10.163.4.146 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::replace_edge_to_child (const walker & *__parent*, const walker & *__child_old*, const walker & *__child_new*) [inherited]

10.163.4.147 void dag< class, _SequenceCtr, class, class >::replace_edge_to_child (const walker & *__parent*, const walker & *__child_old*, const walker & *__child_new*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.148 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::replace_edge_to_parent (const walker & *__parent_old*, const walker & *__parent_new*, const walker & *__child*) [inherited]

10.163.4.149 void dag< class, _SequenceCtr, class, class >::replace_edge_to_parent (const walker & *__parent_old*, const walker & *__parent_new*, const walker & *__child*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.150 children_const_iterator dag< class, _SequenceCtr, class, class >::root_begin () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.151 children_iterator dag< class, _SequenceCtr, class, class >::root_begin () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.152 children_const_iterator dag< class, _SequenceCtr, class, class >::root_end () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.153 children_iterator dag< class, _SequenceCtr, class, class >::root_end () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.154 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 405 of file search_graph.cc.

10.163.4.155 size_type dag< class, _SequenceCtr, class, class >::size () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.156 const_walker dag< class, _SequenceCtr, class, class >::sky () const [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.157 walker dag< class, _SequenceCtr, class, class >::sky () [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.158 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::sort_child_edges (walker *__position*, Compare *comp*) [inherited]

10.163.4.159 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::sort_child_edges (walker *__position*, children_iterator *first*, children_iterator *last*, Compare *comp*) [inherited]

10.163.4.160 void dag< class, _SequenceCtr, class, class >::sort_child_edges (walker *__position*, Compare *comp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.161 void dag< class, _SequenceCtr, class, class >::sort_child_edges (walker *__position*, children_iterator *first*, children_iterator *last*, Compare *comp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.162 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::sort_parent_edges (walker *__position*, Compare *comp*) [inherited]

10.163.4.163 void __DG< _Tp , _SequenceCtr< void *, _PtrAlloc > , _SequenceCtr< void *, _PtrAlloc >::iterator , _SequenceCtr< void *, _PtrAlloc >::const_iterator , _SequenceCtr< void *, _PtrAlloc >::iterator , _Alloc >::sort_parent_edges (walker *__position*, parents_iterator *first*, parents_iterator *last*, Compare *comp*) [inherited]

10.163.4.164 void dag< class, _SequenceCtr, class, class >::sort_parent_edges (walker *__position*, Compare *comp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.165 void dag< class, _SequenceCtr, class, class >::sort_parent_edges (walker *__position*, parents_iterator *first*, parents_iterator *last*, Compare *comp*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.166 walker dag< class, _SequenceCtr, class, class >::split (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const parents_iterator & *__pr_it*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >**.

10.163.4.167 walker dag< class, _SequenceCtr, class, class >::split (const walker & *__parent*, const children_iterator & *__ch_it*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.168 void dag< class, _SequenceCtr, class, class >::split (const __SequenceCtr1< walker, _Allocator1 > & *__parents*, const __SequenceCtr2< walker, _Allocator2 > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.169 walker dag< class, _SequenceCtr, class, class >::split (const walker & *__parent*, const children_iterator & *__ch_it*, const walker & *__child*, const parents_iterator & *__pa_it*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.170 walker dag< class, _SequenceCtr, class, class >::split_back (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.171 walker dag< class, _SequenceCtr, class, class >::split_back (const walker & *__parent*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.172 walker dag< class, _SequenceCtr, class, class >::split_back (const walker & *__parent*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.173 walker dag< class, _SequenceCtr, class, class >::split_front (const __SequenceCtr< walker, _Allocator > & *__parents*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.174 walker dag< class, _SequenceCtr, class, class >::split_front (const walker & *__parent*, const __SequenceCtr< walker, _Allocator > & *__children*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.175 walker dag< class, _SequenceCtr, class, class >::split_front (const walker & *__parent*, const walker & *__child*, const _Tp & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.176 void dag< class, _SequenceCtr, class, class >::swap (_Self & *__x*) [inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.4.177 bool coco::search_graph::unregister_hook (const std::string & *name*)

This method unregisters the work node computation hook with name `name`.

Definition at line 1114 of file search_graph.cc.

10.163.4.178 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 686 of file search_graph.cc.

10.163.4.179 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 669 of file search_graph.cc.

10.163.5 Member Data Documentation

10.163.5.1 gptr<vdbl::database>* coco::search_graph::__dbase

The __dbase variable holds a pointer to the global pointer to the search database.

Definition at line 88 of file search_graph.h.

10.163.5.2 vdbl::userid coco::search_graph::__dbuser

The __dbuser variable holds the database user id for accessing the search database.

Definition at line 91 of file search_graph.h.

10.163.5.3 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 97 of file search_graph.h.

10.163.5.4 _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * dag< class, _SequenceCtr, class, class >::_C_ground [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.5.5 int dag< class, _SequenceCtr, class, class >::_C_mark [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.5.6 _DG_node< _Tp, _SequenceCtr< void *, _PtrAlloc >, _SequenceCtr< void *, _PtrAlloc >::iterator > * dag< class, _SequenceCtr, class, class >::_C_sky [protected, inherited]

Reimplemented from **dgraph**< _Tp, _SequenceCtr, _PtrAlloc, _Alloc >.

10.163.5.7 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 85 of file search_graph.h.

10.163.5.8 std::list<walker> coco::search_graph::focus

This variable holds the list of currently active search focusses.

Definition at line 67 of file search_graph.h.

10.163.5.9 std::list<const_walker> coco::search_graph::inspector

This variable holds the list of currently registered search inspectors.

Definition at line 69 of file search_graph.h.

10.163.5.10 search_inspector coco::search_graph::inspector_for_root

This member holds a search_inspector pointing to the root node.

Definition at line 74 of file search_graph.h.

10.163.5.11 ptr<search_node> coco::search_graph::ptr_ground

This member points to the ground of the search graph.

Definition at line 76 of file search_graph.h.

10.163.5.12 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 79 of file search_graph.h.

10.163.5.13 search_node* coco::search_graph::root

This member points to the root of the search graph.

Definition at line 72 of file search_graph.h.

10.163.5.14 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 82 of file search_graph.h.

10.163.5.15 search_graph_context* coco::search_graph::sgc

This is the search graph context corresponding to the current work node.

Definition at line 94 of file search_graph.h.

The documentation for this class was generated from the following files:

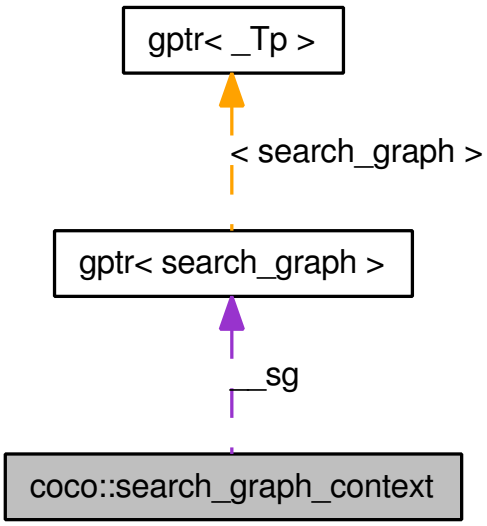
- search_graph.h
- search_graph.cc
- search_graph.hpp

10.164 coco::search_graph_context Class Reference

An evaluation context when retrieving from the search database.

`#include <sgraphctx.h>`

Collaboration diagram for coco::search_graph_context:



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

10.164.1 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).

Definition at line 44 of file sgraphctx.h.

10.164.2 Constructor & Destructor Documentation

10.164.2.1 coco::search_graph_context::search_graph_context () [inline]

Standard Constructor

Definition at line 54 of file sgraphctx.h.

10.164.2.2 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.

10.164.2.3 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.

10.164.2.4 coco::search_graph_context::search_graph_context (const search_graph_context & _s) [inline]

Standard Copy Constructor

Definition at line 61 of file sgraphctx.h.

10.164.2.5 virtual coco::search_graph_context::~search_graph_context () [inline, virtual]

Standard Destructor

Definition at line 64 of file sgraphctx.h.

10.164.3 Member Function Documentation

10.164.3.1 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.

10.164.3.2 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:

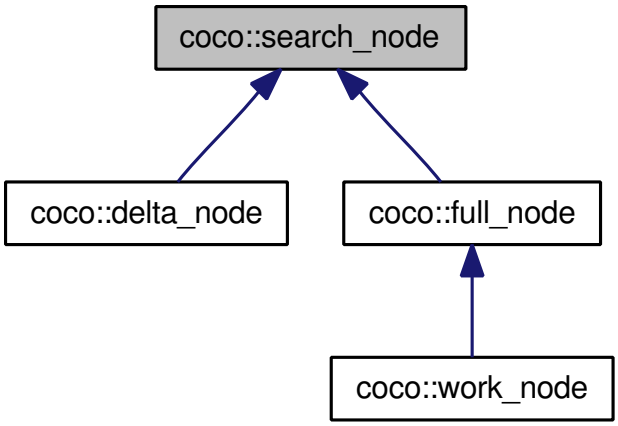
- sgraphctx.h

10.165 coco::search_node Class Reference

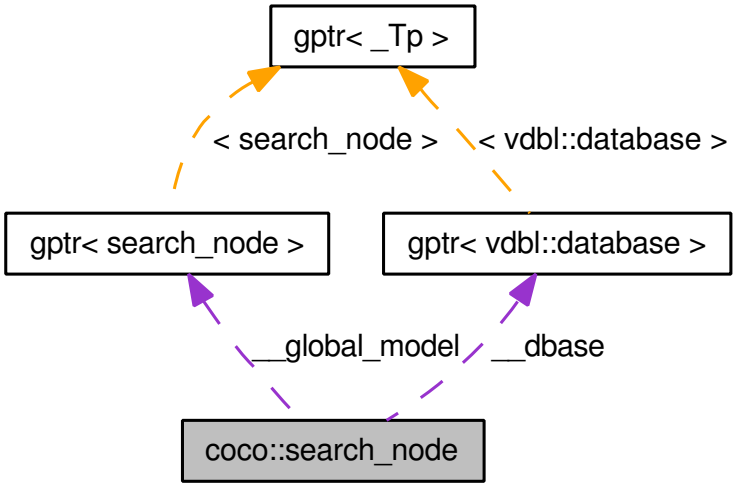
Base type of the nodes in the search graph.

`#include <search_node.h>`

Inheritance diagram for coco::search_node:



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

10.165.1 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.

Definition at line 88 of file search_node.h.

10.165.2 Constructor & Destructor Documentation

10.165.2.1 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.

10.165.2.2 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.

10.165.2.3 coco::search_node::search_node (const search_node & *__sn*) [inline]

Standard Copy Constructor

Definition at line 51 of file search_node.hpp.

10.165.2.4 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.

10.165.2.5 coco::search_node::~search_node () [inline, virtual]

Standard Destructor

Definition at line 66 of file search_node.hpp.

10.165.3 Member Function Documentation

10.165.3.1 gptr<vdbl::database>* coco::search_node::database () const [inline]

This is the accessor method for the search database.

Definition at line 164 of file search_node.h.

10.165.3.2 vdbl::userid coco::search_node::get_dbuserid () const [inline]

This is the accessor method for the database user id.

Definition at line 158 of file search_node.h.

10.165.3.3 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 167 of file search_node.h.

10.165.3.4 vdbl::rowid coco::search_node::get_rowid () const [inline]

This is the accessor method for the row id of this node.

Definition at line 173 of file search_node.h.

10.165.3.5 gptr<search_node>* coco::search_node::global_model () const [inline]

This is the accessor method for the global model.

Definition at line 161 of file search_node.h.

10.165.3.6 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.

Reimplemented in coco::delta_node, and coco::full_node.

Definition at line 122 of file search_node.h.

10.165.3.7 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 184 of file search_node.h.

10.165.3.8 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 180 of file search_node.h.

10.165.3.9 search_node & coco::search_node::operator= (const search_node & *__w*) [inline]

Standard Assignment Operator

Definition at line 174 of file search_node.hpp.

10.165.3.10 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 117 of file search_node.h.

10.165.3.11 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 170 of file search_node.h.

10.165.3.12 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 176 of file search_node.h.

10.165.3.13 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.

10.165.3.14 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.

10.165.4 Friends And Related Function Documentation

10.165.4.1 friend class search_graph [friend]

Definition at line 195 of file search_node.h.

10.165.5 Member Data Documentation

10.165.5.1 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 96 of file search_node.h.

10.165.5.2 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 93 of file search_node.h.

10.165.5.3 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 99 of file search_node.h.

10.165.5.4 search_node_id coco::search_node::_id [protected]

This is the unique identifier of this search node.

Definition at line 104 of file search_node.h.

10.165.5.5 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 108 of file search_node.h.

10.165.5.6 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 112 of file search_node.h.

10.165.5.7 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 102 of file search_node.h.

The documentation for this class was generated from the following files:

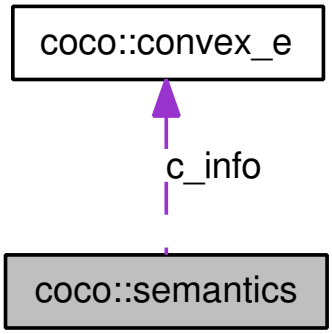
- search_node.h
- search_node.hpp

10.166 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.

10.166.1 Detailed Description

This class is used to describe additional semantics information associated to an expression_node in the expression DAG.

Definition at line 271 of file semantics.h.

10.166.2 Constructor & Destructor Documentation

10.166.2.1 coco::semantics::semantics () [inline]

Standard Constructor

Definition at line 441 of file semantics.h.

10.166.2.2 coco::semantics::semantics (const semantics & __s) [inline]

Standard Copy Constructor

Definition at line 318 of file semantics.h.

10.166.2.3 coco::semantics::~semantics () [inline]

Standard Destructor

Definition at line 326 of file semantics.h.

10.166.3 Member Function Documentation

10.166.3.1 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 387 of file semantics.h.

10.166.3.2 const activity_descr& coco::semantics::activity () const [inline]

This is the accessor for the activity description in the property_flags structure.

Definition at line 384 of file semantics.h.

10.166.3.3 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 373 of file semantics.h.

10.166.3.4 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 370 of file semantics.h.

10.166.3.5 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 408 of file semantics.h.

10.166.3.6 bool coco::semantics::hard_flag () const [inline]

This is the accessor for the hard (constraint) flag in the annotation_flags structure.

Definition at line 405 of file semantics.h.

10.166.3.7 bool coco::semantics::inactive () const [inline]

This method returns whether the constraint is known to be inactive.

Definition at line 436 of file semantics.h.

10.166.3.8 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 429 of file semantics.h.

10.166.3.9 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 433 of file semantics.h.

10.166.3.10 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 401 of file semantics.h.

10.166.3.11 bool coco::semantics::integer_flag () const [inline]

This is the accessor for the integer flag in the annotation_flags structure.

Definition at line 398 of file semantics.h.

10.166.3.12 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 415 of file semantics.h.

10.166.3.13 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 412 of file semantics.h.

10.166.3.14 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 394 of file semantics.h.

10.166.3.15 bool coco::semantics::kj_flag () const [inline]

This is the accessor for the KJ flag in the annotation_flags structure.

Definition at line 391 of file semantics.h.

10.166.3.16 void coco::semantics::merge (const semantics & __s) [inline]

This method merges this semantics structure with the structure __s.

Definition at line 500 of file semantics.h.

10.166.3.17 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 156 of file semantics.cc.

10.166.3.18 bool coco::semantics::redundancy () const [inline]

This method returns whether the constraint is known to be redundant.

Definition at line 425 of file semantics.h.

10.166.3.19 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 380 of file semantics.h.

10.166.3.20 const tristate& coco::semantics::separability () const [inline]

This is the accessor for the seperable entry in the property_flags structure.

Definition at line 377 of file semantics.h.

10.166.3.21 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 422 of file semantics.h.

10.166.3.22 const type_annotation& coco::semantics::type () const [inline]

This is the accessor for the type specifier in the annotation_flags structure.

Definition at line 419 of file semantics.h.

10.166.4 Friends And Related Function Documentation

10.166.4.1 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 251 of file semantics.cc.

10.166.5 Member Data Documentation

10.166.5.1 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 298 of file semantics.h.

10.166.5.2 activity_descr coco::semantics::act

The activity information for constraints

Definition at line 277 of file semantics.h.

10.166.5.3 int coco::semantics::addinfo

for KJ variables this numbe is the corresponding constraint number, -1 if kappa (for objective)

Definition at line 302 of file semantics.h.

10.166.5.4 struct { ... } coco::semantics::annotation_flags

The annotation_flags describe model specific properties

10.166.5.5 convex_e coco::semantics::c_info

The convexity structure for any node

Definition at line 276 of file semantics.h.

10.166.5.6 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 306 of file semantics.h.

10.166.5.7 int coco::semantics::dim

The dim member stores the number of variables involved in that expression (the apparent dimension).

Definition at line 309 of file semantics.h.

10.166.5.8 bool coco::semantics::hard

a hard or soft constraint?

Definition at line 288 of file semantics.h.

10.166.5.9 tristate coco::semantics::has_0chnbase

node has a 0-chain base?

Definition at line 293 of file semantics.h.

10.166.5.10 struct { ... } coco::semantics::info_flags

The info_flags describe algorithmic properties

10.166.5.11 bool coco::semantics::integer

integer (not real) value?

Definition at line 286 of file semantics.h.

10.166.5.12 bool coco::semantics::is_at_either_bound

Whether a constraint is known to be on either bound (unknown on which)

Definition at line 279 of file semantics.h.

10.166.5.13 bool coco::semantics::kj

from KJ conditions?

Definition at line 285 of file semantics.h.

10.166.5.14 struct { ... } coco::semantics::property_flags

The property_flags describe mathematical constraint properties

10.166.5.15 tristate coco::semantics::separable

The seperability information for any node.

Definition at line 278 of file semantics.h.

10.166.5.16 int coco::semantics::stage

This describes the stage to which this node belongs in a multistage problem.

Definition at line 312 of file semantics.h.

10.166.5.17 type_annotation coco::semantics::type

a exists, forall, free, stochastic type

Definition at line 287 of file semantics.h.

The documentation for this class was generated from the following files:

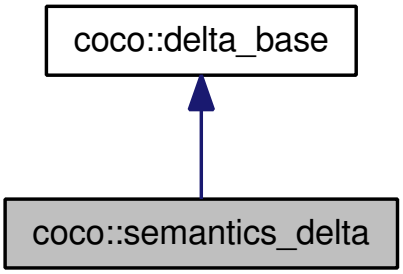
- semantics.h
- semantics.cc

10.167 coco::semantics_delta Class Reference

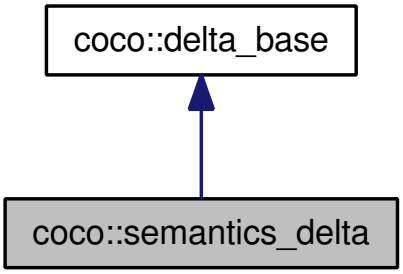
The semantics delta class for changing the node semantics within a model.

`#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) 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)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Protected Attributes

- std::string _action

Friends

- class semantics_undelta

10.167.1 Detailed Description

This class is used to specify information for changing the node semantics within a model.

Definition at line 91 of file semantics_delta.h.

10.167.2 Constructor & Destructor Documentation

10.167.2.1 coco::semantics_delta::semantics_delta () [inline]

Standard Constructor

Definition at line 272 of file semantics_delta.h.

10.167.2.2 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 34 of file semantics_delta.cc.

10.167.2.3 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.

10.167.2.4 coco::semantics_delta::semantics_delta (const semantics_delta & __d) [inline]

Standard Copy Constructor

Definition at line 285 of file semantics_delta.h.

10.167.3 Member Function Documentation

10.167.3.1 bool coco::semantics_delta::apply (work_node & _x, undelta_base *& _u, const delta_id & _did) const [virtual]

Apply the delta with delta_id _d to work node _x, hereby changing the bounds as requested.

Reimplemented from coco::delta_base.

Definition at line 222 of file semantics_delta.cc.

10.167.3.2 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *&& *_u*, const delta_id & *_d*) 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.

10.167.3.3 virtual void 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.

Reimplemented in coco::table_delta.

Definition at line 189 of file api_deltabase.h.

10.167.3.4 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.

10.167.3.5 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.

10.167.3.6 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.

10.167.3.7 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.

10.167.3.8 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.

10.167.3.9 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.

10.167.3.10 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.

10.167.3.11 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.

10.167.3.12 void coco::semantics_delta::destroy_copy (delta_base * *__d*) const [inline]

Clone Destructor

Definition at line 297 of file semantics_delta.h.

10.167.3.13 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.

10.167.3.14 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.

10.167.3.15 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.

10.167.3.16 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.

10.167.3.17 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.

10.167.3.18 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.

10.167.3.19 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.

10.167.3.20 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.

10.167.3.21 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.167.3.22 delta 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 api_deltabase.h.

10.167.3.23 semantics_delta* coco::semantics_delta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::delta_base.

Definition at line 295 of file semantics_delta.h.

10.167.3.24 bool coco::semantics_delta::operator!= (const semantics_delta & _c) const [inline]

Definition at line 405 of file semantics_delta.h.

10.167.3.25 bool coco::semantics_delta::operator!= (const delta_base & _c) const [inline]

Definition at line 397 of file semantics_delta.h.

10.167.3.26 bool coco::semantics_delta::operator== (const semantics_delta & _c) const [inline]

Comparison operators

Definition at line 402 of file semantics_delta.h.

10.167.3.27 bool coco::semantics_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Definition at line 394 of file semantics_delta.h.

10.167.3.28 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 126 of file semantics_delta.cc.

10.167.3.29 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 47 of file semantics_delta.cc.

10.167.3.30 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 57 of file semantics_delta.cc.

10.167.3.31 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 152 of file semantics_delta.cc.

10.167.3.32 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 76 of file semantics_delta.cc.

10.167.3.33 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 138 of file semantics_delta.cc.

10.167.3.34 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 66 of file semantics_delta.cc.

10.167.3.35 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 194 of file semantics_delta.cc.

10.167.3.36 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 106 of file semantics_delta.cc.

10.167.3.37 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 180 of file semantics_delta.cc.

10.167.3.38 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 96 of file semantics_delta.cc.

10.167.3.39 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.

10.167.3.40 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.

10.167.3.41 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 166 of file semantics_delta.cc.

10.167.3.42 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 86 of file semantics_delta.cc.

10.167.3.43 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 208 of file semantics_delta.cc.

10.167.3.44 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 116 of file semantics_delta.cc.

10.167.3.45 virtual void 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 api_deltabase.h.

10.167.4 Friends And Related Function Documentation

10.167.4.1 friend class semantics_undelta [friend]

Definition at line 409 of file semantics_delta.h.

10.167.5 Member Data Documentation

10.167.5.1 std::string coco::delta_base::_action [protected, inherited]

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 files:

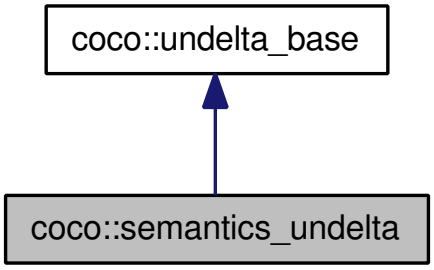
- semantics_delta.h
- semantics_delta.cc

10.168 coco::semantics_undelta Class Reference

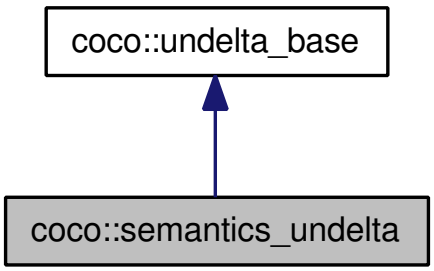
The semantics undelta class for undoing changes to the node semantics in a model.

`#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 ()
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

Friends

- class semantics_delta

10.168.1 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.

Definition at line 41 of file semantics_delta.h.

10.168.2 Constructor & Destructor Documentation

10.168.2.1 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.

10.168.2.2 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.

10.168.2.3 coco::semantics_undelta::semantics_undelta (const semantics_undelta & __d) [inline]

Standard Copy Constructor

Definition at line 65 of file semantics_delta.h.

10.168.3 Member Function Documentation

10.168.3.1 void coco::semantics_undelta::destroy_copy (undelta_base * __d) const [inline]

Clone Destructor

Definition at line 77 of file semantics_delta.h.

10.168.3.2 undelta coco::undelta_base::make_undelta () [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 283 of file api_deltabase.h.

10.168.3.3 semantics_undelta* coco::semantics_undelta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::undelta_base.

Definition at line 75 of file semantics_delta.h.

10.168.3.4 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::undelta_base.

Definition at line 253 of file semantics_delta.cc.

10.168.3.5 bool 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.

10.168.4 Friends And Related Function Documentation

10.168.4.1 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:

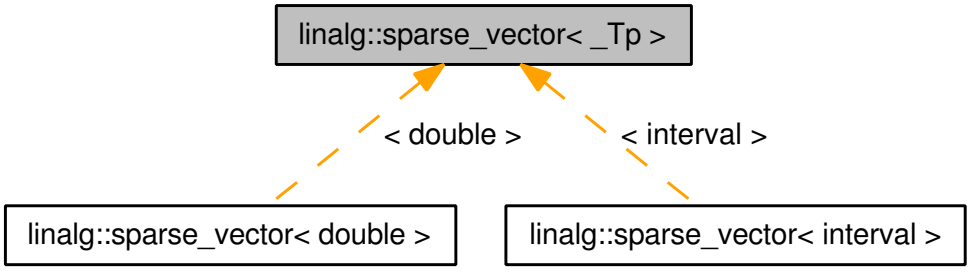
- semantics_delta.h
- semantics_delta.cc

10.169 linalg::sparse_vector< _Tp > Class Template Reference

Sparse Vector Template Class.

`#include <linalg.h>`

Inheritance diagram for linalg::sparse_vector< _Tp >:



Public Member Functions

- sparse_vector ()
- sparse_vector (const _Self &__m)
- sparse_vector (const size_t &n)
- void setOutputFormat (const linalg_output_format &f)
- const linalg_output_format & getOutputFormat () const
- _Self & operator= (const _Self &x)
- const_reference operator[ ] (size_type i) const MTL_THROW_ASSERTION
- reference operator[ ] (size_type i)
- _Self & operator+= (const _Self &a)
- _Self & operator*= (const _Tp &a)
- _Tp & operator*= (const _Self &a)

Protected Attributes

- linalg_output_format output_format

Friends

- template<typename _Tq >
  std::ostream & std::operator<< (std::ostream &s, const sparse_vector< _Tq > &A)

10.169.1 Detailed Description

template<class _Tp> class linalg::sparse_vector< _Tp >

This class is the base class for all sparse vector storage/computation in the COCONUT environment. It is based on the MTL, but wraps the MTL classes to make changing the matrix library possible (and hopefully not too awkward. The matrix is general rectangular column-major.

Definition at line 611 of file linalg.h.

10.169.2 Constructor & Destructor Documentation

10.169.2.1 template<class _Tp> linalg::sparse_vector< _Tp >::sparse_vector () [inline]

The Standard Constructor

Definition at line 635 of file linalg.h.

10.169.2.2 template<class _Tp> linalg::sparse_vector< _Tp >::sparse_vector (const _Self & __m) [inline]

The Standard Copy Constructor

Definition at line 637 of file linalg.h.

10.169.2.3 template<class _Tp> linalg::sparse_vector< _Tp >::sparse_vector (const size_t & *n*) [inline]

Constructor, which builds a vector of size n.

Definition at line 644 of file linalg.h.

10.169.3 Member Function Documentation

10.169.3.1 template<class _Tp> const linalg_output_format& linalg::sparse_vector< _Tp >::getOutputFormat () const [inline]

This method returns the output format of the sparse vector.

Definition at line 650 of file linalg.h.

10.169.3.2 template<class _Tp> _Tp& linalg::sparse_vector< _Tp >::operator*= (const _Self & *a*) [inline]

This operator computes the scalar product of this vector and a.

Definition at line 696 of file linalg.h.

10.169.3.3 template<class _Tp> _Self& linalg::sparse_vector< _Tp >::operator*= (const _Tp & *a*) [inline]

This operator multiplies this vector by a.

Definition at line 688 of file linalg.h.

10.169.3.4 template<class _Tp> _Self& linalg::sparse_vector< _Tp >::operator+= (const _Self & *a*) [inline]

This operator adds a to this vector.

Definition at line 681 of file linalg.h.

10.169.3.5 template<class _Tp> _Self& linalg::sparse_vector< _Tp >::operator= (const _Self & *x*) [inline]

The Assignment Operator, which copies the memory structure rather than copying the pointers.

Definition at line 655 of file linalg.h.

10.169.3.6 template<class _Tp> reference linalg::sparse_vector< _Tp >::operator[ ] (size_type *i*) [inline]

This operator returns the i-th element of the vector

Definition at line 672 of file linalg.h.

10.169.3.7 template<class _Tp> const_reference linalg::sparse_vector< _Tp >::operator[ ] (size_type *i*) const [inline]

This operator returns the i-th element of the vector

Definition at line 666 of file linalg.h.

10.169.3.8 template<class _Tp> void linalg::sparse_vector< _Tp >::setOutputFormat (const linalg_output_format & *f*) [inline]

This method sets the output format of the sparse vector.

Definition at line 647 of file linalg.h.

10.169.4 Friends And Related Function Documentation

10.169.4.1 template<class _Tp> template<typename _Tq > std::ostream& std::operator<< (std::ostream & *s*, const sparse_vector< _Tq > & *A*) [friend]

10.169.5 Member Data Documentation

10.169.5.1 template<class _Tp> linalg_output_format linalg::sparse_vector< _Tp >::output_format [protected]

This variable describes the output format for matrix output. Its possible values are linalg_output_sparse and linalg_output_full.

Definition at line 631 of file linalg.h.

The documentation for this class was generated from the following file:

- linalg.h

10.170 coco::sparsity_visitor Class Reference

Preorder visitor which calculates sparsity structures.

Inheritance diagram for coco::sparsity_visitor:



Collaboration diagram for coco::sparsity_visitor:



Public Types

- typedef _Ret **return_value**

Public Member Functions

- sparsity_visitor (splist &__o, int __n, bool __b=false)
- sparsity_visitor (const sparsity_visitor &__c)
- virtual void **vinit** ()
- virtual bool **preorder** (const _Node &__n)
- virtual void **collect** (const _Node &__n, collect_value __r)

  - bool preorder (const expression_node &r)
  - const variable_indicator & vvalue ()
  - const variable_indicator & value ()
  - void collect (const expression_node &r, const variable_indicator &__v)

10.170.1 Detailed Description

This preorder visitor is used to calculate the sparsity structure of Hessians and Jacobi matrices.

Definition at line 62 of file sparsity.cc.

10.170.2 Member Typedef Documentation

10.170.2.1 typedef _Ret preorder_visitor< class, class, class >::return_value [inherited]

10.170.3 Constructor & Destructor Documentation

10.170.3.1 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.

10.170.3.2 coco::sparsity_visitor::sparsity_visitor (const sparsity_visitor & __c) [inline]

Standard Copy Constructor

Definition at line 121 of file sparsity.cc.

10.170.4 Member Function Documentation

10.170.4.1 virtual void preorder_visitor< class, class, class >::collect (const _Node & __n, collect_value __r) [virtual, inherited]

10.170.4.2 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 249 of file sparsity.cc.

10.170.4.3 virtual bool preorder_visitor< class, class, class >::preorder (const _Node & __n) [virtual, inherited]

10.170.4.4 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.

10.170.4.5 const variable_indicator& coco::sparsity_visitor::value () [inline]

This is a method as needed by a preorder visitor.

Definition at line 247 of file sparsity.cc.

10.170.4.6 virtual void preorder_visitor< class, class, class >::vinit () [virtual, inherited]

10.170.4.7 const variable_indicator& coco::sparsity_visitor::vvalue () [inline, virtual]

This is a method as needed by a preorder visitor.

Reimplemented from **preorder_visitor< expression_node, const variable_indicator &, const variable_indicator & >**.

Definition at line 246 of file sparsity.cc.

The documentation for this class was generated from the following file:

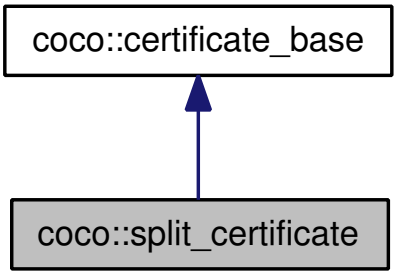
- sparsity.cc

10.171 coco::split_certificate Class Reference

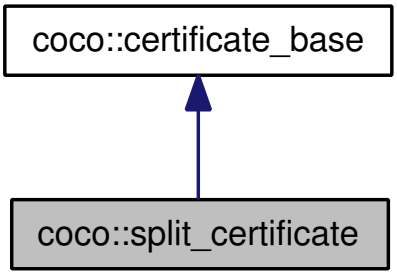
The certificate for deltas formed by splits.

`#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)

Protected Attributes

- std::string _contents

10.171.1 Detailed Description

The split certificate for generated splits.

Definition at line 134 of file api_cert.h.

10.171.2 Constructor & Destructor Documentation

10.171.2.1 coco::split_certificate::split_certificate () [inline]

Standard Constructor

Definition at line 141 of file api_cert.h.

10.171.2.2 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.

10.171.2.3 coco::split_certificate::split_certificate (const split_certificate & *__c*) [inline]

Standard Copy Constructor

Definition at line 147 of file api_cert.h.

10.171.2.4 virtual coco::split_certificate::~split_certificate () [inline, virtual]

Standard Destructor

Definition at line 156 of file api_cert.h.

10.171.3 Member Function Documentation

10.171.3.1 void coco::split_certificate::destroy_copy (certificate_base * *__c*) const [inline]

Clone Destructor

Definition at line 167 of file api_cert.h.

10.171.3.2 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::certificate_base.

Definition at line 177 of file api_cert.h.

10.171.3.3 certificate 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 api_certbase.h.

10.171.3.4 split_certificate* coco::split_certificate::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::certificate_base.

Definition at line 159 of file api_cert.h.

10.171.3.5 bool coco::split_certificate::operator!= (const split_certificate & _c) const [inline]

Definition at line 191 of file api_cert.h.

10.171.3.6 bool coco::split_certificate::operator!= (const certificate_base & _c) const [inline]

Definition at line 184 of file api_cert.h.

10.171.3.7 bool coco::split_certificate::operator== (const split_certificate & _c) const [inline]

Comparison operators

Definition at line 189 of file api_cert.h.

10.171.3.8 bool coco::split_certificate::operator== (const certificate_base & _c) const [inline]

Comparison operators

Definition at line 181 of file api_cert.h.

10.171.3.9 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::certificate_base.

Definition at line 172 of file api_cert.h.

10.171.4 Member Data Documentation

10.171.4.1 std::string coco::certificate_base::_contents [protected, inherited]

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:

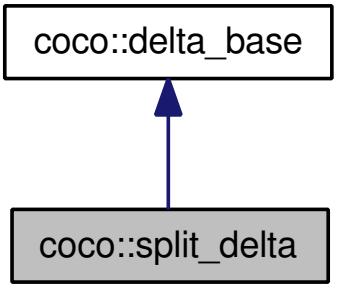
- api_cert.h

10.172 coco::split_delta Class Reference

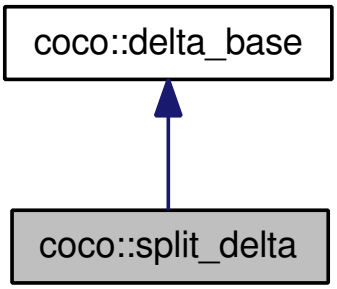
The split delta class for proposing useful splits.

`#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) 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)
- const std::string & get_action () const
- virtual void convert (work_node &_x, delta_base *&_d)
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Public Attributes

- std::list< std::vector< delta > > splits

Protected Attributes

- std::string _action

10.172.1 Detailed Description

This class is used to propose useful splits and store them in the proposed_splits structure in the work_node.

Definition at line 79 of file split_delta.h.

10.172.2 Constructor & Destructor Documentation

10.172.2.1 coco::split_delta::split_delta () [inline]

Standard Constructor

Definition at line 91 of file split_delta.h.

10.172.2.2 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.

10.172.2.3 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.

10.172.2.4 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.

10.172.2.5 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.

10.172.2.6 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.

10.172.2.7 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.

10.172.2.8 coco::split_delta::~split_delta () [inline]

Standard Destructor

Definition at line 162 of file split_delta.h.

10.172.2.9 coco::split_delta::split_delta (const split_delta & *__s*) [inline]

Standard Copy Constructor

Definition at line 165 of file split_delta.h.

10.172.3 Member Function Documentation

10.172.3.1 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.

10.172.3.2 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.

10.172.3.3 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 319 of file split_delta.h.

10.172.3.4 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 285 of file split_delta.h.

10.172.3.5 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.

10.172.3.6 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.

10.172.3.7 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.

10.172.3.8 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.

10.172.3.9 bool coco::split_delta::apply (work_node & *x*, undelta_base *& *_u*, const delta_id & *_did*) const [inline, virtual]

Apply the delta with delta_id _d to work node _x, hereby changing the bounds as requested.

Reimplemented from coco::delta_base.

Definition at line 271 of file split_delta.h.

10.172.3.10 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*) 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.

10.172.3.11 virtual void 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.

Reimplemented in coco::table_delta.

Definition at line 189 of file api_deltabase.h.

10.172.3.12 void coco::split_delta::destroy_copy (delta_base * __d) const [inline]

Clone Destructor

Definition at line 170 of file split_delta.h.

10.172.3.13 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.172.3.14 delta 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 api_deltabase.h.

10.172.3.15 split_delta* coco::split_delta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::delta_base.

Definition at line 168 of file split_delta.h.

10.172.3.16 bool coco::split_delta::operator!= (const split_delta & _c) const [inline]

Definition at line 252 of file split_delta.h.

10.172.3.17 bool coco::split_delta::operator!= (const delta_base & _c) const [inline]

Definition at line 245 of file split_delta.h.

10.172.3.18 bool coco::split_delta::operator== (const split_delta & _c) const [inline]

Comparison operators

Definition at line 250 of file split_delta.h.

10.172.3.19 bool coco::split_delta::operator== (const delta_base & _c) const [inline]

Comparison operators

Definition at line 242 of file split_delta.h.

10.172.3.20 virtual void 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 api_deltabase.h.

10.172.4 Member Data Documentation

10.172.4.1 std::string coco::delta_base::_action [protected, inherited]

The action (type) of this delta

Definition at line 154 of file api_deltabase.h.

10.172.4.2 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:

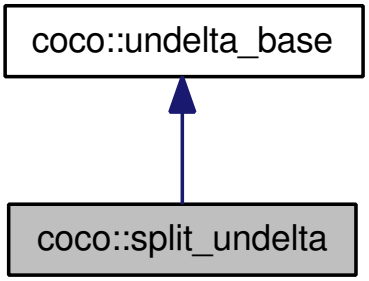
- split_delta.h

10.173 coco::split_undelta Class Reference

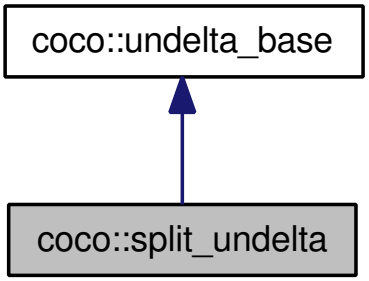
The split undelta class for removing proposed splits from a work_node.

`#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 ()
- virtual bool unapply3 (work_node &_x, const work_node &_y, const delta_id &_d) const

10.173.1 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.

Definition at line 42 of file split_delta.h.

10.173.2 Constructor & Destructor Documentation

10.173.2.1 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.

10.173.2.2 coco::split_undelta::split_undelta (const split_undelta & *_su*) [inline]

Standard Copy Constructor

Definition at line 54 of file split_delta.h.

10.173.2.3 virtual coco::split_undelta::~split_undelta () [inline, virtual]

Standard Destructor

Definition at line 62 of file split_delta.h.

10.173.3 Member Function Documentation

10.173.3.1 void coco::split_undelta::destroy_copy (undelta_base * *__d*) const [inline]

Clone Destructor

Definition at line 67 of file split_delta.h.

10.173.3.2 undelta coco::undelta_base::make_undelta () [inline, inherited]

Construct an undelta from this undelta_base.

Definition at line 283 of file api_deltabase.h.

10.173.3.3 split_undelta* coco::split_undelta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::undelta_base.

Definition at line 65 of file split_delta.h.

10.173.3.4 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::undelta_base.

Definition at line 256 of file split_delta.h.

10.173.3.5 bool 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

10.174 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

10.174.1 Detailed Description

This is the base class of all statistcs classes used to keep the statistics information of all the inference engines.

Definition at line 39 of file ie_statistic.h.

10.174.2 Constructor & Destructor Documentation

10.174.2.1 coco::statistic_info::statistic_info () [inline]

Standard Constructor

Definition at line 49 of file ie_statistic.h.

10.174.2.2 virtual coco::statistic_info::~statistic_info () [inline, virtual]

Standard Destructor

Definition at line 51 of file ie_statistic.h.

10.174.3 Member Function Documentation

10.174.3.1 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.

10.174.4 Member Data Documentation

10.174.4.1 double coco::statistic_info::effectiveness

This is the effectiveness of the engine

Definition at line 43 of file ie_statistic.h.

10.174.4.2 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:

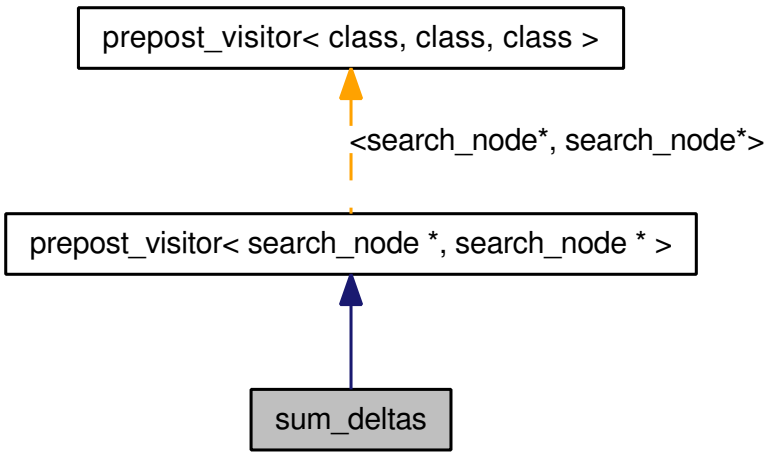
- ie_statistic.h

10.175 sum_deltas Class Reference

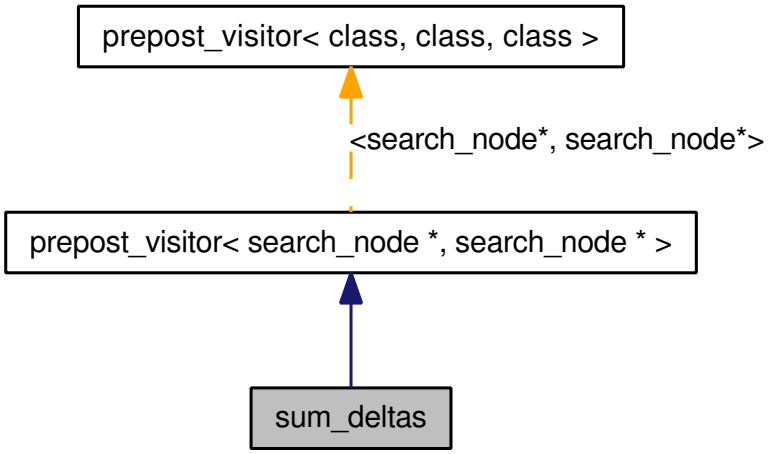
Pre-post visitor for summing up all the deltas during work node extraction.

`#include <sum_deltas.h>`

Inheritance diagram for sum_deltas:



Collaboration diagram for sum_deltas:



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)
- virtual bool **preorder** (const _Node &__n)
- virtual bool **postorder** (const _Node &__n)
- virtual void **collect** (const _Node &__n, collect_value __r)

- 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

10.175.1 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.

Definition at line 60 of file sum_deltas.h.

10.175.2 Constructor & Destructor Documentation

10.175.2.1 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.

10.175.2.2 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.

10.175.2.3 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.

10.175.3 Member Function Documentation

10.175.3.1 virtual void prepost_visitor< class, class, class >::collect (const _Node & *__n*, collect_value *__r*) [virtual, inherited]

10.175.3.2 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.

10.175.3.3 virtual bool prepost_visitor< class, class, class >::postorder (const _Node & *__n*) [virtual, inherited]

10.175.3.4 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.

10.175.3.5 virtual bool prepost_visitor< class, class, class >::preorder (const _Node & *__n*) [virtual, inherited]

10.175.3.6 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.

10.175.3.7 return_value sum_deltas::value () [inline]

This method is required by a **prepost_visitor**.

Definition at line 170 of file sum_deltas.h.

10.175.3.8 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.

10.175.3.9 void sum_deltas::vinit () [inline, virtual]

This method is required by a **prepost_visitor**.

Reimplemented from **prepost_visitor< search_node *, search_node * >**.

Definition at line 107 of file sum_deltas.h.

10.175.3.10 return_value sum_deltas::vvalue () [inline, virtual]

This method is required by a **prepost_visitor**.

Reimplemented from **prepost_visitor**< **search_node** ∗, **search_node** ∗ >.

Definition at line 119 of file sum_deltas.h.

10.175.4 Member Data Documentation

10.175.4.1 std::vector<return_value> sum_deltas::ancestors

The return values of the ancestors

Definition at line 64 of file sum_deltas.h.

10.175.4.2 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.

10.175.4.3 std::list<delta_id>∗ sum_deltas::dels

The list of deltas to be summed

Definition at line 68 of file sum_deltas.h.

10.175.4.4 std::set<search_node_id>∗ sum_deltas::nds

The list of search_node parents

Definition at line 70 of file sum_deltas.h.

10.175.4.5 const search_node_id∗ sum_deltas::psnid

The search_node parent until which we sum

Definition at line 74 of file sum_deltas.h.

10.175.4.6 return_value sum_deltas::result

The return value of this visit

Definition at line 66 of file sum_deltas.h.

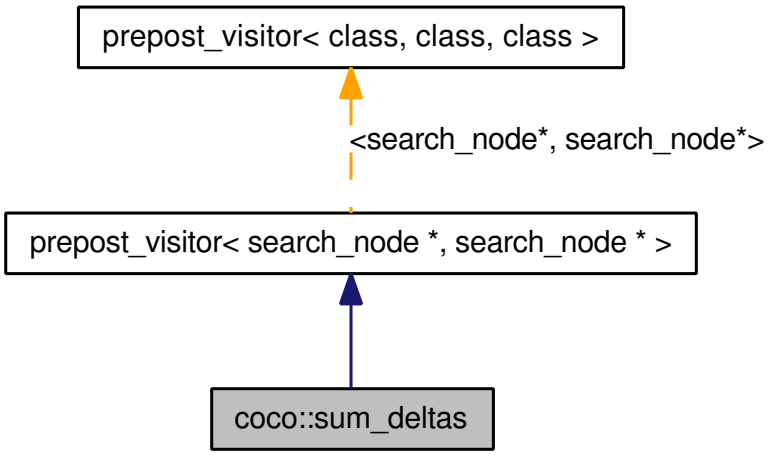
The documentation for this class was generated from the following file:

- sum_deltas.h

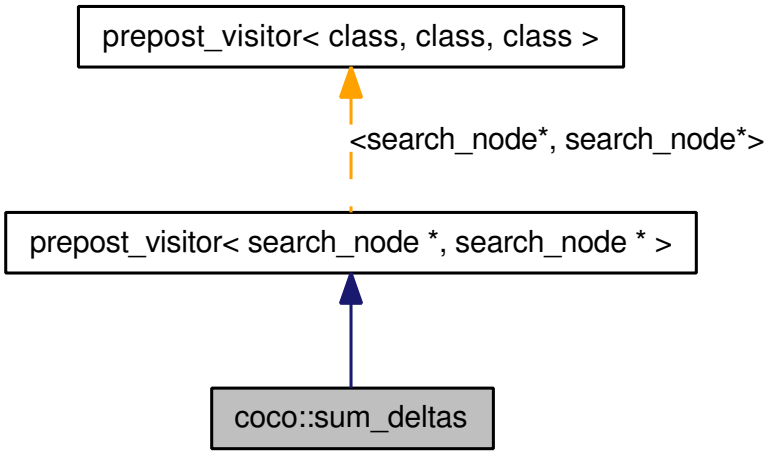
10.176 coco::sum_deltas Class Reference

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

Inheritance diagram for coco::sum_deltas:



Collaboration diagram for coco::sum_deltas:



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)
- virtual bool **preorder** (const _Node &__n)
- virtual bool **postorder** (const _Node &__n)
- virtual void **collect** (const _Node &__n, collect_value __r)

- 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

10.176.1 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.

Definition at line 142 of file search_graph.cc.

10.176.2 Constructor & Destructor Documentation

10.176.2.1 coco::sum_deltas::sum_deltas (std::list< delta_id > & *_dels*, std::set< search_node_id > & *_nds*) [inline]

Constructor for uncompared search

Definition at line 159 of file search_graph.cc.

10.176.2.2 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 168 of file search_graph.cc.

10.176.2.3 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 178 of file search_graph.cc.

10.176.3 Member Function Documentation

10.176.3.1 virtual void prepost_visitor< class, class, class >::collect (const _Node & *__n*, collect_value *__r*) [virtual, inherited]

10.176.3.2 void coco::sum_deltas::collect (search_node *const &, return_value const & *ancestor*) [inline]

This method is required by a **prepost_visitor**.

Definition at line 223 of file search_graph.cc.

10.176.3.3 virtual bool prepost_visitor< class, class, class >::postorder (const _Node & *__n*) [virtual, inherited]

10.176.3.4 bool coco::sum_deltas::postorder (search_node *const & *r*) [inline]

This method is required by a **prepost_visitor**.

Definition at line 229 of file search_graph.cc.

10.176.3.5 virtual bool prepost_visitor< class, class, class >::preorder (const _Node & __n) [virtual, inherited]

10.176.3.6 bool coco::sum_deltas::preorder (search_node *const & r) [inline]

This method is required by a **prepost_visitor**.

Definition at line 207 of file search_graph.cc.

10.176.3.7 return_value coco::sum_deltas::value () [inline]

This method is required by a **prepost_visitor**.

Definition at line 252 of file search_graph.cc.

10.176.3.8 void coco::sum_deltas::vcollect (return_value const &) [inline]

This method is required by a **prepost_visitor**.

Definition at line 194 of file search_graph.cc.

10.176.3.9 void coco::sum_deltas::vinit () [inline, virtual]

This method is required by a **prepost_visitor**.

Reimplemented from **prepost_visitor< search_node *, search_node * >**.

Definition at line 189 of file search_graph.cc.

10.176.3.10 return_value coco::sum_deltas::vvalue () [inline, virtual]

This method is required by a **prepost_visitor**.

Reimplemented from **prepost_visitor< search_node *, search_node * >**.

Definition at line 201 of file search_graph.cc.

10.176.4 Member Data Documentation

10.176.4.1 std::vector<return_value> coco::sum_deltas::ancestors

The return values of the ancestors

Definition at line 146 of file search_graph.cc.

10.176.4.2 const std::set<search_node_id>* coco::sum_deltas::c_nds

The list of search_node parents of the compared node

Definition at line 154 of file search_graph.cc.

10.176.4.3 std::list<delta_id>* coco::sum_deltas::dels

The list of deltas to be summed

Definition at line 150 of file search_graph.cc.

10.176.4.4 std::set<search_node_id>* coco::sum_deltas::nds

The list of search_node parents

Definition at line 152 of file search_graph.cc.

10.176.4.5 const search_node_id* coco::sum_deltas::psnid

The search_node parent until which we sum

Definition at line 156 of file search_graph.cc.

10.176.4.6 return_value coco::sum_deltas::result

The return value of this visit

Definition at line 148 of file search_graph.cc.

The documentation for this class was generated from the following file:

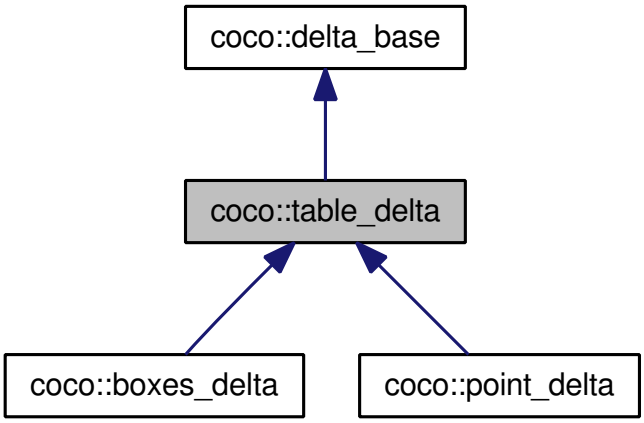
- search_graph.cc

10.177 coco::table_delta Class Reference

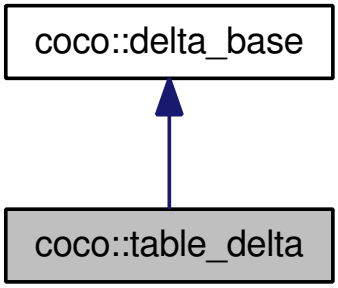
The base class for all deltas adding information to the search database.

`#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) 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)
- const std::string & get_action () const
- virtual void unkeep ()
- virtual bool apply3 (work_node &_x, const work_node &_y, undelta_base *&_u, const delta_id &_d) const

Protected Attributes

- std::string _action

10.177.1 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.

Definition at line 43 of file table_delta.h.

10.177.2 Member Typedef Documentation

10.177.2.1 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.

10.177.2.2 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.

10.177.3 Constructor & Destructor Documentation

10.177.3.1 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.

10.177.3.2 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.

10.177.3.3 coco::table_delta::table_delta (const table_delta & *__d*) [inline]

Standard Copy Constructor

Definition at line 69 of file table_delta.h.

10.177.3.4 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.

10.177.3.5 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.

10.177.4 Member Function Documentation

10.177.4.1 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.

10.177.4.2 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.

10.177.4.3 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.

10.177.4.4 bool coco::table_delta::apply (work_node & *_x*, undelta_base *& *_u*, const delta_id & *_did*) 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::delta_base.

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 35 of file table_delta.cc.

10.177.4.5 bool coco::delta_base::apply3 (work_node & *_x*, const work_node & *_y*, undelta_base *& *_u*, const delta_id & *_d*) 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.

10.177.4.6 void coco::table_delta::convert (work_node & *_x*, delta_base *& *_u*) [virtual]

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.

Reimplemented from coco::delta_base.

Definition at line 42 of file table_delta.cc.

10.177.4.7 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.

10.177.4.8 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.

10.177.4.9 const std::string& coco::delta_base::get_action () const [inline, inherited]

Retrieve the action information (the delta type) for this delta.

Definition at line 184 of file api_deltabase.h.

10.177.4.10 delta 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 api_deltabase.h.

10.177.4.11 virtual table_delta* coco::table_delta::new_copy () const [inline, virtual]

Clone Operation

Reimplemented from coco::delta_base.

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 111 of file table_delta.h.

10.177.4.12 bool coco::table_delta::operator!= (const table_delta & _c) const [inline]

Definition at line 143 of file table_delta.h.

10.177.4.13 bool coco::table_delta::operator!= (const delta_base & _c) const [inline]

Reimplemented in coco::boxes_delta, and coco::point_delta.

Definition at line 137 of file table_delta.h.

10.177.4.14 bool coco::table_delta::operator== (const table_delta & _c) const [inline]

Comparison operators

Definition at line 142 of file table_delta.h.

10.177.4.15 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 134 of file table_delta.h.

10.177.4.16 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.

10.177.4.17 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.

10.177.4.18 virtual void 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 api_deltabase.h.

10.177.5 Member Data Documentation

10.177.5.1 std::string coco::delta_base::_action [protected, inherited]

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 files:

- table_delta.h
- table_delta.cc

10.178 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.

10.178.1 Detailed Description

This class is used to store the reason of termination of inference and management modules.

Definition at line 45 of file termreason.h.

10.178.2 Constructor & Destructor Documentation

10.178.2.1 coco::termination_reason::termination_reason () [inline]

This is a map for message translation. Standard Constructor

Definition at line 58 of file termreason.h.

10.178.2.2 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.

10.178.2.3 coco::termination_reason::termination_reason (const termination_reason & *__t*) [inline]

Standard Copy Constructor

Definition at line 66 of file termreason.h.

10.178.3 Member Function Documentation

10.178.3.1 int coco::termination_reason::get_code () const [inline]

This method return the termination code.

Definition at line 81 of file termreason.h.

10.178.3.2 const std::string& coco::termination_reason::get_message () const [inline]

This method return the termination message.

Definition at line 78 of file termreason.h.

10.178.3.3 termination_reason& coco::termination_reason::operator= (const termination_reason & *__t*) [inline]

Standard Assignment Operator

Definition at line 70 of file termreason.h.

10.178.4 Friends And Related Function Documentation

10.178.4.1 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

10.179 std::triple< _T1, _T2, _T3 > Struct Template 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

10.179.1 Detailed Description

template<class _T1, class _T2, class _T3> struct std::triple< _T1, _T2, _T3 >

This class is used to hold three objects of arbitrary types. It is a slight generalization of `std::pair`.

Definition at line 42 of file stlp_triple.h.

10.179.2 Member Typedef Documentation

10.179.2.1 template<class _T1, class _T2, class _T3> typedef _T1 std::triple< _T1, _T2, _T3 >::first_type

`first_type` is the first bound type

Definition at line 44 of file stlp_triple.h.

10.179.2.2 template<class _T1, class _T2, class _T3> typedef _T2 std::triple< _T1, _T2, _T3 >::second_type

`second_type` is the second bound type

Definition at line 46 of file stlp_triple.h.

10.179.2.3 template<class _T1, class _T2, class _T3> typedef _T3 std::triple< _T1, _T2, _T3 >::third_type

`third_type` is the third bound type

Definition at line 48 of file stlp_triple.h.

10.179.3 Constructor & Destructor Documentation

10.179.3.1 template<class _T1, class _T2, class _T3> std::triple< _T1, _T2, _T3 >::triple () [inline]

Definition at line 64 of file stlp_triple.h.

10.179.3.2 template<class _T1, class _T2, class _T3> std::triple< _T1, _T2, _T3 >::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.

10.179.3.3 template<class _T1, class _T2, class _T3> template<class _U1 , class _U2 , class _U3 > std::triple< _T1, _T2, _T3 >::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.

10.179.4 Member Data Documentation

10.179.4.1 template<class _T1, class _T2, class _T3> _T1 std::triple< _T1, _T2, _T3 >::first

`first` is a copy of the first object

Definition at line 51 of file stlp_triple.h.

10.179.4.2 template<class _T1, class _T2, class _T3> _T2 std::triple< _T1, _T2, _T3 >::second

`second` is a copy of the second object

Definition at line 53 of file stlp_triple.h.

10.179.4.3 template<class _T1, class _T2, class _T3> _T3 std::triple< _T1, _T2, _T3 >::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:

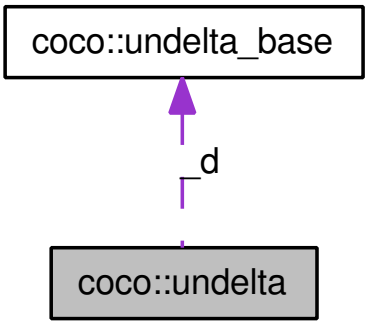
- stlp_triple.h

10.180 coco::undelta Class Reference

The undelta class (undo of updates to work nodes).

`#include <api_deltabase.h>`

Collaboration diagram for coco::undelta:



Public Member Functions

- undelta ()
- undelta (undelta_base *__d)
- 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)

10.180.1 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 223 of file api_deltabase.h.

10.180.2 Constructor & Destructor Documentation

10.180.2.1 coco::undelta::undelta () [inline]

Standard Constructor

Definition at line 231 of file api_deltabase.h.

10.180.2.2 coco::undelta::undelta (undelta_base * *__d*) [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 235 of file api_deltabase.h.

10.180.2.3 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 197 of file api_delta.h.

10.180.2.4 coco::undelta::~undelta () [inline]

Destructor, which frees the `_d` using `delete`

Definition at line 203 of file api_delta.h.

10.180.3 Member Function Documentation

10.180.3.1 const undelta_base * coco::undelta::get_base () const [inline]

Return the undelta_base stored in this wrapper

Definition at line 218 of file api_delta.h.

10.180.3.2 undelta & 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 209 of file api_delta.h.

10.180.3.3 bool 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 220 of file api_delta.h.

10.180.3.4 bool 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 223 of file api_delta.h.

The documentation for this class was generated from the following files:

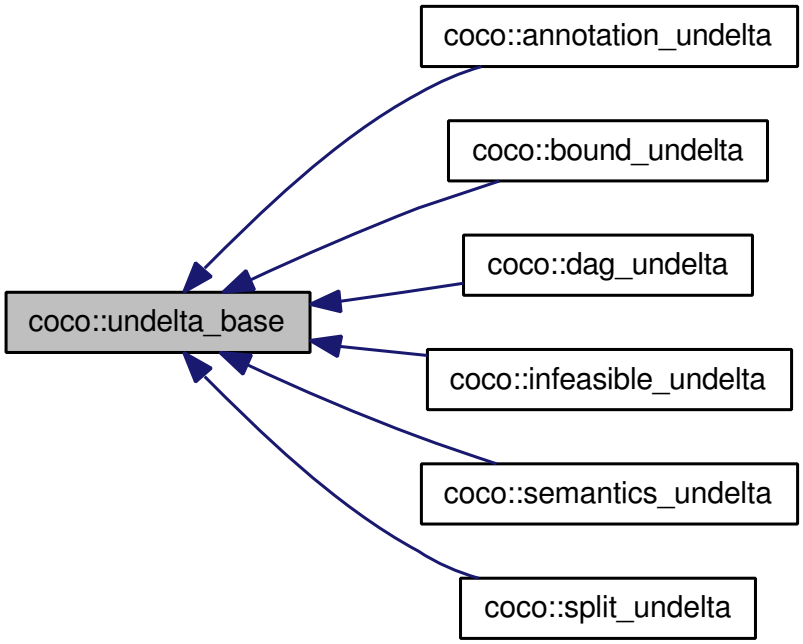
- api_deltabase.h
- api_delta.h

10.181 coco::undelta_base Class Reference

Base class for the undeltas.

`#include <api_deltabase.h>`

Inheritance diagram for 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 ()
- 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

10.181.1 Detailed Description

Base class for the undeltas, which is wrapped by the undelta class since copy constructors cannot directly be overloaded.

Definition at line 266 of file api_deltabase.h.

10.181.2 Constructor & Destructor Documentation

10.181.2.1 coco::undelta_base::undelta_base () [inline]

Standard Constructor

Definition at line 270 of file api_deltabase.h.

10.181.2.2 coco::undelta_base::undelta_base (const undelta_base & *__d*) [inline]

Standard Copy Constructor, which is usually not used.

Definition at line 272 of file api_deltabase.h.

10.181.3 Member Function Documentation

10.181.3.1 undelta coco::undelta_base::make_undelta () [inline]

Construct an undelta from this undelta_base.

Definition at line 283 of file api_deltabase.h.

10.181.3.2 virtual undelta_base* coco::undelta_base::new_copy () const [inline, virtual]

Clone Operation, which can be overloaded Clone Destructor Standard Destructor

Reimplemented in coco::annotation_undelta, coco::bound_undelta, coco::dag_undelta, coco::infeasible_undelta, coco::semantics_undelta, and coco::split_undelta.

Definition at line 275 of file api_deltabase.h.

10.181.3.3 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

Reimplemented in coco::annotation_undelta, coco::bound_undelta, coco::dag_undelta, coco::infeasible_undelta, coco::semantics_undelta, and coco::split_undelta.

Definition at line 290 of file api_deltabase.h.

10.181.3.4 bool 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.

The documentation for this class was generated from the following files:

- api_deltabase.h
- api_delta.h

10.182 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)

10.182.1 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.

Definition at line 73 of file evaluator.h.

10.182.2 Constructor & Destructor Documentation

10.182.2.1 coco::variable_indicator::variable_indicator () [inline]

Standard Constructor

Definition at line 80 of file evaluator.h.

10.182.2.2 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.

10.182.2.3 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.

10.182.2.4 coco::variable_indicator::variable_indicator (const variable_indicator & *__x*) [inline]

Standard Copy Constructor

Definition at line 95 of file evaluator.h.

10.182.2.5 virtual coco::variable_indicator::~variable_indicator () [inline, virtual]

Standard Destructor

Definition at line 98 of file evaluator.h.

10.182.3 Member Function Documentation

10.182.3.1 void coco::variable_indicator::clear () [inline]

The clear method unsets all variables in the variable_indicator

Definition at line 160 of file evaluator.h.

10.182.3.2 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.

10.182.3.3 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.

10.182.3.4 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.

10.182.3.5 variable_indicator& coco::variable_indicator::operator= (const variable_indicator & *__v*) [inline]

Standard Assignment Operator

Definition at line 257 of file evaluator.h.

10.182.3.6 bool coco::variable_indicator::operator== (const variable_indicator & *__v*) [inline]

Standard Comparison Operator

Definition at line 264 of file evaluator.h.

10.182.3.7 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.

10.182.3.8 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.

10.182.3.9 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.

10.182.3.10 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.

10.182.3.11 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.

10.182.3.12 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.

10.182.3.13 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.

10.182.3.14 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.

10.182.3.15 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.

10.182.3.16 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.

The documentation for this class was generated from the following file:

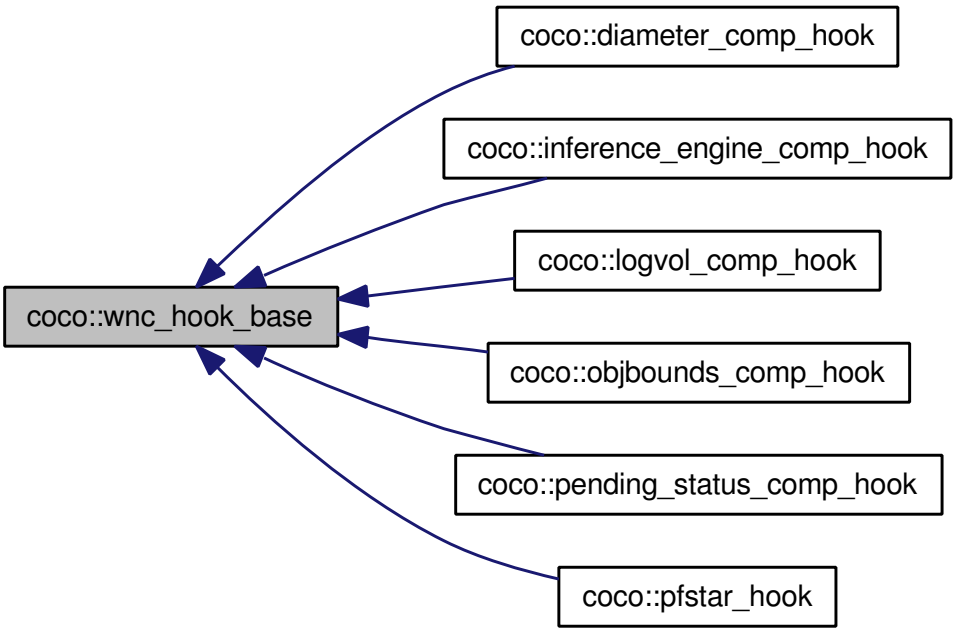
- evaluator.h

10.183 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

10.183.1 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.

Definition at line 111 of file comp_hook.h.

10.183.2 Constructor & Destructor Documentation

10.183.2.1 coco::wnc_hook_base::wnc_hook_base () [inline]

Standard Constructor

Definition at line 153 of file comp_hook.h.

10.183.2.2 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.

10.183.2.3 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.

10.183.2.4 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.

10.183.2.5 virtual coco::wnc_hook_base::~wnc_hook_base () [inline, virtual]

Standard Destructor

Definition at line 163 of file comp_hook.h.

10.183.3 Member Function Documentation

10.183.3.1 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.

10.183.3.2 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.

10.183.3.3 template<class _CI > bool coco::wnc_hook_base::_init_column (vdbl::standard_table & *stable*, const std::string & *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 238 of file comp_hook.h.

10.183.3.4 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::diameter_comp_hook, coco::inference_engine_comp_hook, coco::logvol_comp_hook, coco::objbounds_comp_hook, coco::pending_status_comp_hook, and coco::pfstar_hook.

Definition at line 187 of file comp_hook.h.

10.183.3.5 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::diameter_comp_hook, coco::inference_engine_comp_hook, coco::logvol_comp_hook, coco::objbounds_comp_hook, coco::pending_status_comp_hook, and coco::pfstar_hook.

Definition at line 182 of file comp_hook.h.

10.183.3.6 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.

10.183.3.7 virtual wnc_hook_base* coco::wnc_hook_base::new_copy () const [pure virtual]

Clone Operation, which must be overloaded

Implemented in coco::diameter_comp_hook, coco::inference_engine_comp_hook, coco::logvol_comp_hook, coco::objbounds_comp_hook, coco::pending_status_comp_hook, and coco::pfstar_hook.

10.183.3.8 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.

10.183.3.9 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::diameter_comp_hook, coco::inference_engine_comp_hook, coco::logvol_comp_hook, coco::objbounds_comp_hook, coco::pending_status_comp_hook, and coco::pfstar_hook.

10.183.3.10 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:

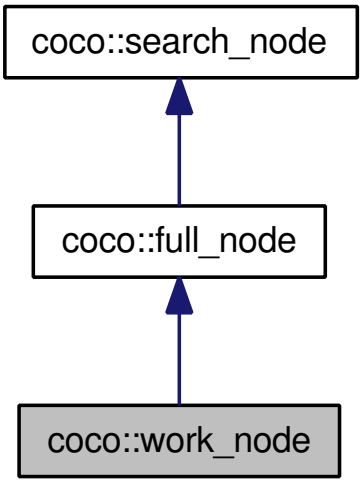
- comp_hook.h
- comp_hook.cc

10.184 coco::work_node Class Reference

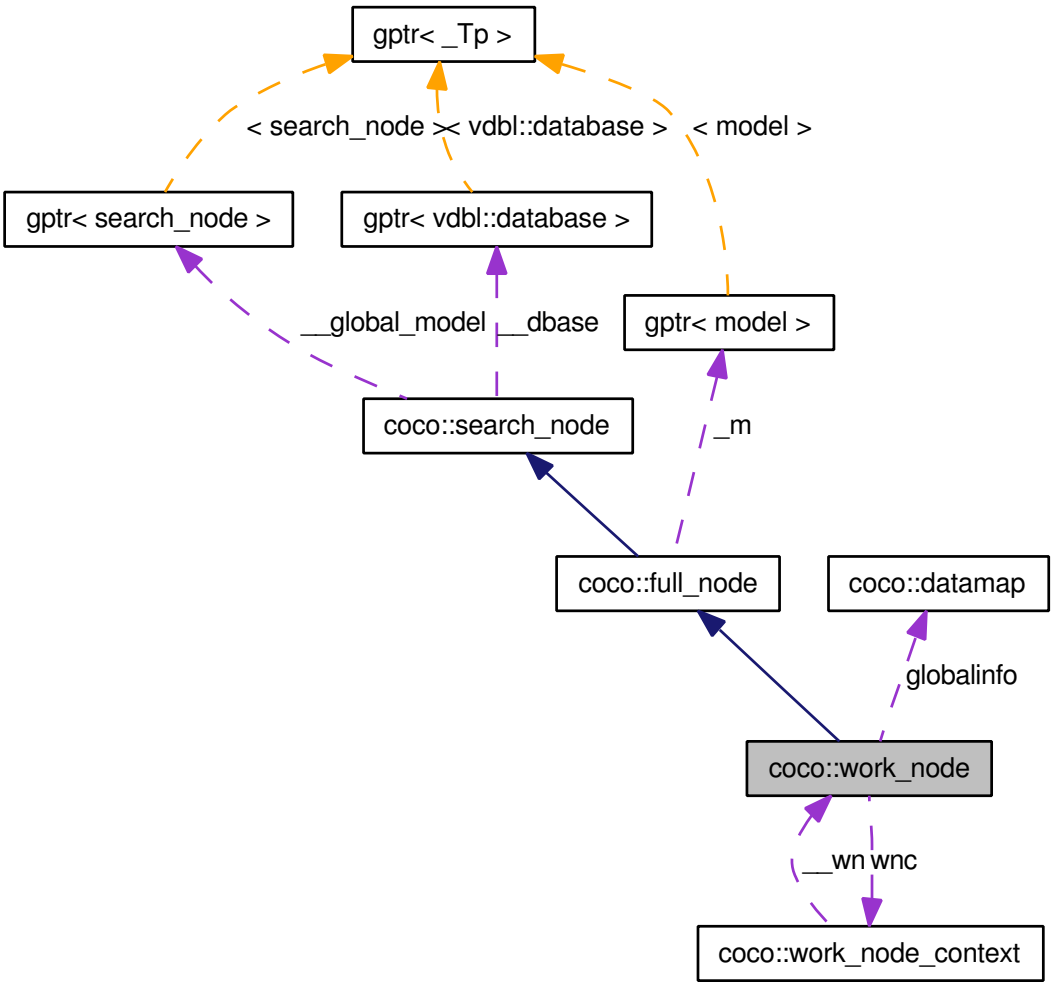
Work node, which is passed to the inference engines.

`#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
- 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::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

10.184.1 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.

Definition at line 417 of file search_node.h.

10.184.2 Member Typedef Documentation

10.184.2.1 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 425 of file search_node.h.

10.184.2.2 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 439 of file search_node.h.

10.184.2.3 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 511 of file search_node.h.

10.184.3 Constructor & Destructor Documentation

10.184.3.1 coco::work_node::work_node (const work_node & *__w*) [inline]

Standard Copy Constructor

Definition at line 100 of file search_node.hpp.

10.184.3.2 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 192 of file search_node.cc.

10.184.3.3 coco::work_node::work_node (const search_node_id & *_i*, const vdbl::userid & *_dui*, gptr< model > & *__m*, gptr< vdbl::database > & *__d*, const std::list< 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 162 of file search_node.cc.

10.184.3.4 coco::work_node::~work_node () [inline, virtual]

Standard Destructor

Definition at line 135 of file search_node.hpp.

10.184.4 Member Function Documentation

10.184.4.1 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 271 of file search_node.cc.

10.184.4.2 gptr<vdbl::database>* coco::search_node::database () const [inline, inherited]

This is the accessor method for the search database.

Definition at line 164 of file search_node.h.

10.184.4.3 double coco::work_node::gain () const [inline]

This method returns the current gain_factor of the work_node.

Definition at line 755 of file search_node.h.

10.184.4.4 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 653 of file search_node.h.

10.184.4.5 const annotation& coco::full_node::get_annotation (unsigned int *i*) const [inline, inherited]

A call to this method returns a const reference to the `i`th annotation.

Definition at line 369 of file search_node.h.

10.184.4.6 const std::vector<annotation>& 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 374 of file search_node.h.

10.184.4.7 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 153 of file search_node.hpp.

10.184.4.8 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 138 of file search_node.hpp.

10.184.4.9 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 479 of file search_node.hpp.

10.184.4.10 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 455 of file search_node.hpp.

10.184.4.11 const vdbl::database* 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 385 of file search_node.h.

10.184.4.12 vdbl::database* 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 396 of file search_node.h.

10.184.4.13 vdbl::userid coco::search_node::get_dbuserid () const [inline, inherited]

This is the accessor method for the database user id.

Definition at line 158 of file search_node.h.

10.184.4.14 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 439 of file search_node.hpp.

10.184.4.15 delta coco::work_node::get_delta (const delta_id & *_id*) [inline]

This method returns the delta with delta_id _id.

Definition at line 423 of file search_node.hpp.

10.184.4.16 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 160 of file search_node.hpp.

10.184.4.17 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 146 of file search_node.hpp.

10.184.4.18 const vdbl::viewdbase& coco::work_node::get_fullviewdbase () const [inline]

This method returns the actual full view database.

Definition at line 633 of file search_node.h.

10.184.4.19 vdbl::viewdbase& coco::work_node::get_fullviewdbase () [inline]

This method returns the actual full view database.

Definition at line 630 of file search_node.h.

10.184.4.20 search_node_id coco::search_node::get_id () const [inline, inherited]

This is the accessor method for the search node id of this node.

Definition at line 167 of file search_node.h.

10.184.4.21 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 614 of file search_node.h.

10.184.4.22 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::full_node.

Definition at line 610 of file search_node.h.

10.184.4.23 model* 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 390 of file search_node.h.

10.184.4.24 vdbl::rowid coco::search_node::get_rowid () const [inline, inherited]

This is the accessor method for the row id of this node.

Definition at line 173 of file search_node.h.

10.184.4.25 transaction_number coco::work_node::get_transaction_number () [inline]

This method returns a unique transaction number for split deltas.

Definition at line 521 of file search_node.h.

10.184.4.26 const vdbl::viewdbase& coco::work_node::get_viewdbase () const [inline]

This method returns the actual view database.

Definition at line 627 of file search_node.h.

10.184.4.27 vdbl::viewdbase& coco::work_node::get_viewdbase () [inline]

This method returns the actual view database.

Definition at line 624 of file search_node.h.

10.184.4.28 const work_node_context* coco::work_node::get_work_node_context () const [inline]

This method returns the actual work node context

Definition at line 621 of file search_node.h.

10.184.4.29 work_node_context* coco::work_node::get_work_node_context () [inline]

This method returns the actual work node context

Definition at line 618 of file search_node.h.

10.184.4.30 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 772 of file search_node.h.

10.184.4.31 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 333 of file search_node.cc.

10.184.4.32 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 765 of file search_node.h.

10.184.4.33 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 325 of file search_node.cc.

10.184.4.34 const datamap& coco::work_node::global_info () const [inline]

This method returns the global data used by the algorithm. See `globalinfo`.

Definition at line 758 of file search_node.h.

10.184.4.35 gptr<search_node>* coco::search_node::global_model () const [inline, inherited]

This is the accessor method for the global model.

Definition at line 161 of file search_node.h.

10.184.4.36 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 220 of file search_node.cc.

10.184.4.37 bool 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::search_node.

Definition at line 361 of file search_node.h.

10.184.4.38 void 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 184 of file search_node.h.

10.184.4.39 void 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 180 of file search_node.h.

10.184.4.40 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 752 of file search_node.h.

10.184.4.41 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 263 of file search_node.cc.

10.184.4.42 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 242 of file search_node.cc.

10.184.4.43 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 513 of file search_node.hpp.

10.184.4.44 unsigned int coco::full_node::n_annotations () const [inline, inherited]

This method returns the number of annotations stored in this full node.

Definition at line 365 of file search_node.h.

10.184.4.45 work_node & coco::work_node::operator= (const full_node & *__f*) [inline]

Special Assignment Operator — be careful when using it!

Reimplemented from coco::full_node.

Definition at line 222 of file search_node.hpp.

10.184.4.46 work_node & coco::work_node::operator= (const work_node & __w) [inline]

Standard Assignment Operator

Definition at line 254 of file search_node.hpp.

10.184.4.47 search_node_relation 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 117 of file search_node.h.

10.184.4.48 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 777 of file search_node.h.

10.184.4.49 void coco::work_node::reset_node_ranges () [inline]

This method resets all unused node ranges to $[-\infty, \infty]$.

Definition at line 503 of file search_node.hpp.

10.184.4.50 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 299 of file search_node.cc.

10.184.4.51 void 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 170 of file search_node.h.

10.184.4.52 void 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 176 of file search_node.h.

10.184.4.53 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.

10.184.4.54 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.

10.184.5 Friends And Related Function Documentation

10.184.5.1 friend class certificate [friend]

Definition at line 806 of file search_node.h.

10.184.5.2 friend class certificate_base [friend, inherited]

Definition at line 400 of file search_node.h.

10.184.5.3 friend class dag_delta [friend, inherited]

Definition at line 401 of file search_node.h.

10.184.5.4 friend class dag_undelta [friend, inherited]

Definition at line 402 of file search_node.h.

10.184.5.5 friend class delta [friend]

Definition at line 804 of file search_node.h.

10.184.5.6 friend class delta_base [friend, inherited]

Definition at line 399 of file search_node.h.

10.184.5.7 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 324 of file search_node.hpp.

10.184.5.8 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 297 of file search_node.hpp.

10.184.5.9 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 340 of file search_node.hpp.

10.184.5.10 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 310 of file search_node.hpp.

10.184.5.11 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 364 of file search_node.hpp.

10.184.5.12 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 35 of file search_node.cc.

10.184.5.13 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 382 of file search_node.hpp.

10.184.5.14 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 76 of file search_node.cc.

10.184.5.15 friend class search_graph [friend, inherited]

Definition at line 195 of file search_node.h.

10.184.5.16 friend class undelta [friend]

Definition at line 805 of file search_node.h.

10.184.5.17 friend class wnc_hook_base [friend]

Definition at line 807 of file search_node.h.

10.184.6 Member Data Documentation

10.184.6.1 gptr<vdbl::database>* 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 96 of file search_node.h.

10.184.6.2 gptr<search_node>* 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 93 of file search_node.h.

10.184.6.3 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 474 of file search_node.h.

10.184.6.4 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 477 of file search_node.h.

10.184.6.5 std::vector<annotation> coco::full_node::_ann [inherited]

This is the vector of annotations active at this full node.

Definition at line 290 of file search_node.h.

10.184.6.6 vdbl::userid 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 99 of file search_node.h.

10.184.6.7 search_node_id coco::search_node::_id [protected, inherited]

This is the unique identifier of this search node.

Definition at line 104 of file search_node.h.

10.184.6.8 std::vector<annotation> 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 108 of file search_node.h.

10.184.6.9 gptr<model>* 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 286 of file search_node.h.

10.184.6.10 vdbl::rowid 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 112 of file search_node.h.

10.184.6.11 search_node_relation 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 102 of file search_node.h.

10.184.6.12 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 450 of file search_node.h.

10.184.6.13 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 447 of file search_node.h.

10.184.6.14 vdbl::standard_table* coco::work_node::dtable

This is a pointer to the "deltas" table in the search database.

Definition at line 459 of file search_node.h.

10.184.6.15 vdbl::tableid coco::work_node::dtable_id

This is the table identifier of the "deltas" table in the search database.

Definition at line 462 of file search_node.h.

10.184.6.16 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 502 of file search_node.h.

10.184.6.17 vdbl::standard_table* coco::work_node::gitable

This is a pointer to the "global info" table in the search database.

Definition at line 464 of file search_node.h.

10.184.6.18 vdbl::tableid coco::work_node::gitable_id

This is the table identifier of the "global info" table in the search database.

Definition at line 467 of file search_node.h.

10.184.6.19 datamap coco::work_node::globalinfo

This member keeps a cache of the global data used in the algorithm

Definition at line 504 of file search_node.h.

10.184.6.20 bool coco::work_node::infeasible

This boolean specifies, whether the problem represented by this work node is known to be infeasible.

Definition at line 495 of file search_node.h.

10.184.6.21 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 499 of file search_node.h.

10.184.6.22 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 492 of file search_node.h.

10.184.6.23 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 487 of file search_node.h.

10.184.6.24 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 453 of file search_node.h.

10.184.6.25 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 534 of file search_node.h.

10.184.6.26 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 537 of file search_node.h.

10.184.6.27 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 457 of file search_node.h.

10.184.6.28 work_node_context* coco::work_node::wnc

This is the work node context corresponding to the current work node.

Definition at line 471 of file search_node.h.

The documentation for this class was generated from the following files:

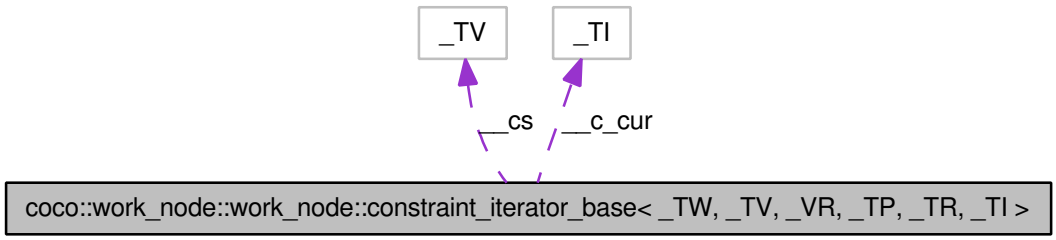
- search_node.h
- search_node.cc
- search_node.hpp

10.185 coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI > Class Template Reference

The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

`#include <search_node.h>`

Collaboration diagram for coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >:



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)

10.185.1 Detailed Description

template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> class coco::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >

This is the base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

See also:

work_node::get_begin

Definition at line 817 of file search_node.h.

10.185.2 Member Typedef Documentation

10.185.2.1 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> typedef ptrdiff_t coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::difference_type

These types are needed for an iterator.

Definition at line 847 of file search_node.h.

10.185.2.2 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> typedef std::bidirectional_iterator_tag coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::iterator_category

These types are needed for an iterator.

Definition at line 840 of file search_node.h.

10.185.2.3 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> typedef _TP coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::pointer

These types are needed for an iterator.

Definition at line 844 of file search_node.h.

10.185.2.4 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> typedef _TR coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::reference

These types are needed for an iterator.

Definition at line 843 of file search_node.h.

10.185.2.5 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> typedef size_t coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::size_type

These types are needed for an iterator.

Definition at line 846 of file search_node.h.

10.185.2.6 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> typedef _TW coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::value_type

These types are needed for an iterator.

Definition at line 842 of file search_node.h.

10.185.3 Constructor & Destructor Documentation

10.185.3.1 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::constraint_iterator_base () [inline]

Standard Constructor

Definition at line 852 of file search_node.h.

10.185.3.2 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::constraint_iterator_base (_Reference *cs*, unsigned int *tp*) [inline]

Constructor, which initializes the vector with `cs` and the type with `tp`.

Definition at line 855 of file search_node.h.

10.185.3.3 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::constraint_iterator_base (const _Self & *__c*) [inline]

Standard Copy Constructor

Definition at line 858 of file search_node.h.

10.185.3.4 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::~constraint_iterator_base () [inline]

Standard Destructor

Definition at line 862 of file search_node.h.

10.185.4 Member Function Documentation

10.185.4.1 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> bool coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator!= (const _Self & *_c*) [inline]

Disequality Comparison Operator

Definition at line 874 of file search_node.h.

10.185.4.2 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> reference coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator* () const [inline]

Dereference Operator

Definition at line 865 of file search_node.h.

10.185.4.3 template<class _TW , class _TV , class _VR , class _TP , class _TR , class _TI > work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator++ (int) [inline]

Increment operator: go to the next constraint of the specified type

Definition at line 594 of file search_node.hpp.

10.185.4.4 template<class _TW , class _TV , class _VR , class _TP , class _TR , class _TI > work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self & coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator++ () [inline]

Increment operator: go to the next constraint of the specified type

Definition at line 579 of file search_node.hpp.

10.185.4.5 template<class _TW , class _TV , class _VR , class _TP , class _TR , class _TI > work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator– (int) [inline]

Decrement operator: go to the previous constraint of the specified type

Definition at line 623 of file search_node.hpp.

10.185.4.6 template<class _TW , class _TV , class _VR , class _TP , class _TR , class _TI > work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self & coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator– () [inline]

Decrement operator: go to the previous constraint of the specified type

Definition at line 604 of file search_node.hpp.

10.185.4.7 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> pointer coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator → () const [inline]

Arrow Operator

Definition at line 867 of file search_node.h.

10.185.4.8 template<class _TW , class _TV , class _VR , class _TP , class _TR , class _TI > work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self & coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator= (const _Self & *_c*) [inline]

Standard Assignment Operator

Definition at line 633 of file search_node.hpp.

10.185.4.9 template<class _TW, class _TV, class _VR, class _TP, class _TR, class _TI> bool coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::operator== (const _Self & *_c*) [inline]

Equality Comparison Operator

Definition at line 870 of file search_node.h.

10.185.4.10 template<class _TW , class _TV , class _VR , class _TP , class _TR , class _TI > work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::_Self & coco::work_node::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >::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 565 of file search_node.hpp.

The documentation for this class was generated from the following files:

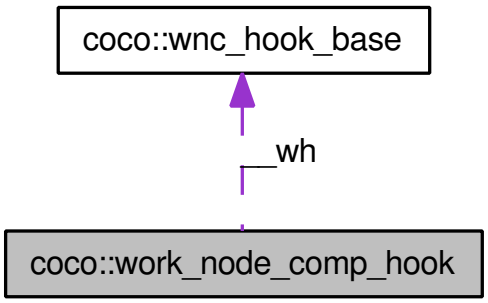
- search_node.h
- search_node.hpp

10.186 coco::work_node_comp_hook Class Reference

The work_node_comp_hook class (work node computation hook).

`#include <comp_hook.h>`

Collaboration diagram for coco::work_node_comp_hook:



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

10.186.1 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.

Definition at line 50 of file comp_hook.h.

10.186.2 Constructor & Destructor Documentation

10.186.2.1 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.

10.186.2.2 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.

10.186.2.3 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.

10.186.3 Member Function Documentation

10.186.3.1 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.

10.186.3.2 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.

10.186.3.3 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.

10.186.3.4 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.

10.186.3.5 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:

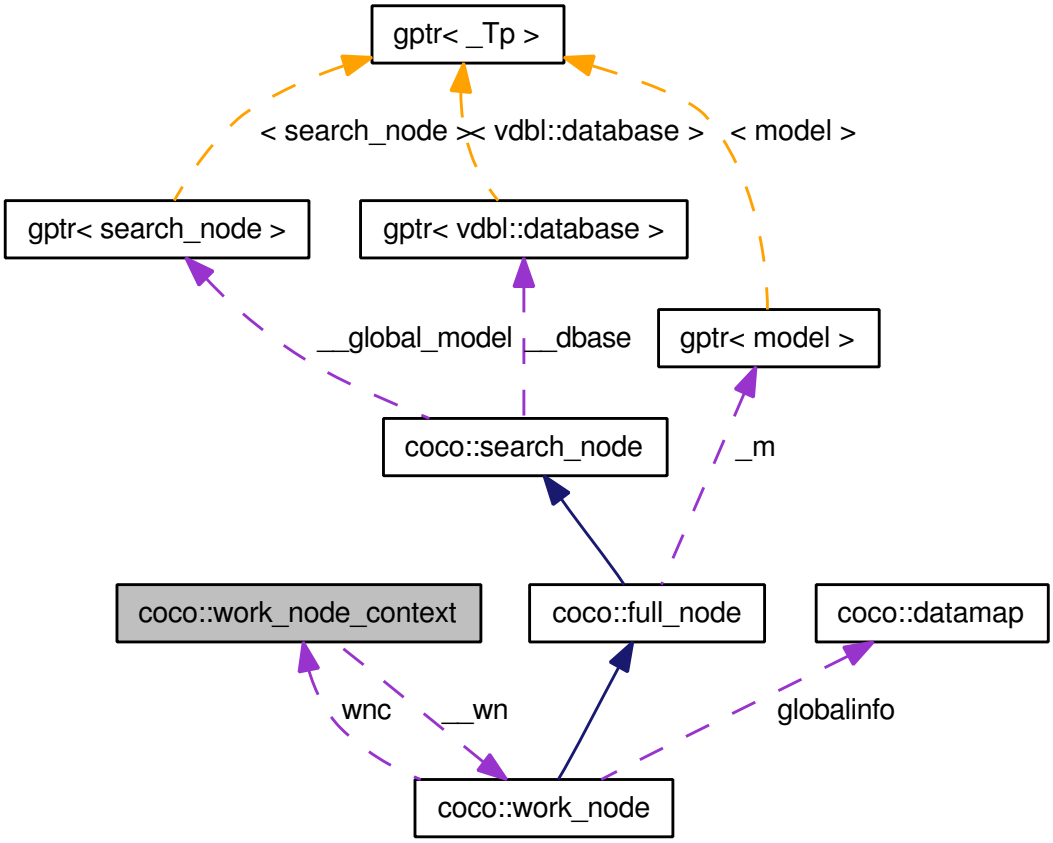
- comp_hook.h

10.187 coco::work_node_context Class Reference

The evaluation context when retrieving from the search database.

`#include <wnodectx.h>`

Collaboration diagram for coco::work_node_context:



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

10.187.1 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).

Definition at line 42 of file wnodectx.h.

10.187.2 Constructor & Destructor Documentation

10.187.2.1 coco::work_node_context::work_node_context () [inline]

Standard Constructor

Definition at line 50 of file wnodectx.h.

10.187.2.2 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.

10.187.2.3 coco::work_node_context::work_node_context (const work_node_context & _w) [inline]

Standard Copy Constructor

Definition at line 54 of file wnodectx.h.

10.187.2.4 virtual coco::work_node_context::~work_node_context () [inline, virtual]

Standard Destructor

Definition at line 56 of file wnodectx.h.

10.187.3 Member Function Documentation

10.187.3.1 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.

10.187.3.2 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

10.188 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

10.188.1 Detailed Description

Definition at line 202 of file dbcompare_expression_scanner.cc.

10.188.2 Member Data Documentation

10.188.2.1 int yy_buffer_state::yy_at_bol

Definition at line 237 of file dbcompare_expression_scanner.cc.

10.188.2.2 int yy_buffer_state::yy_bs_column

The column count.

Definition at line 240 of file dbcompare_expression_scanner.cc.

10.188.2.3 int yy_buffer_state::yy_bs_lineno

The line count.

Definition at line 239 of file dbcompare_expression_scanner.cc.

10.188.2.4 char * yy_buffer_state::yy_buf_pos

Definition at line 208 of file dbcompare_expression_scanner.cc.

10.188.2.5 yy_size_t yy_buffer_state::yy_buf_size

Definition at line 213 of file dbcompare_expression_scanner.cc.

10.188.2.6 int yy_buffer_state::yy_buffer_status

Definition at line 247 of file dbcompare_expression_scanner.cc.

10.188.2.7 char * yy_buffer_state::yy_ch_buf

Definition at line 207 of file dbcompare_expression_scanner.cc.

10.188.2.8 int yy_buffer_state::yy_fill_buffer

Definition at line 245 of file dbcompare_expression_scanner.cc.

10.188.2.9 std::istream * yy_buffer_state::yy_input_file

Definition at line 205 of file dbcompare_expression_scanner.cc.

10.188.2.10 int yy_buffer_state::yy_is_interactive

Definition at line 231 of file dbcompare_expression_scanner.cc.

10.188.2.11 int yy_buffer_state::yy_is_our_buffer

Definition at line 224 of file dbcompare_expression_scanner.cc.

10.188.2.12 int yy_buffer_state::yy_n_chars

Definition at line 218 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

10.189 yy_trans_info Struct Reference

Public Attributes

- flex_int32_t yy_verify
- flex_int32_t yy_nxt

10.189.1 Detailed Description

Definition at line 339 of file dbcompare_expression_scanner.cc.

10.189.2 Member Data Documentation

10.189.2.1 flex_int32_t yy_trans_info::yy_nxt

Definition at line 342 of file dbcompare_expression_scanner.cc.

10.189.2.2 flex_int32_t yy_trans_info::yy_verify

Definition at line 341 of file dbcompare_expression_scanner.cc.

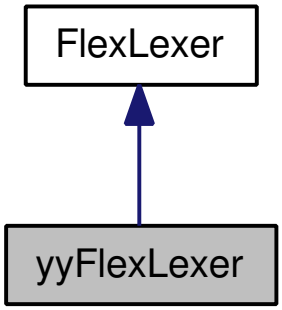
The documentation for this struct was generated from the following files:

- dbcompare_expression_scanner.cc
- dbcompare_sortorder_scanner.cc

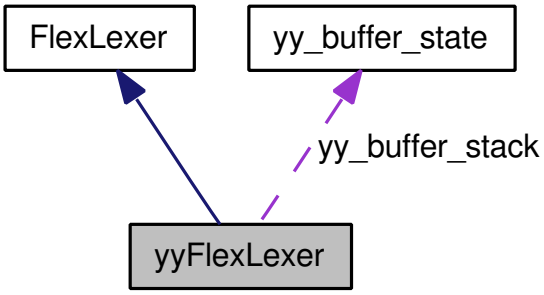
10.190 yyFlexLexer Class Reference

`#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

10.190.1 Detailed Description

Definition at line 112 of file FlexLexer.h.

10.190.2 Constructor & Destructor Documentation

10.190.2.1 yyFlexLexer::yyFlexLexer (FLEX_STD istream * *arg_yyin* = 0, FLEX_STD ostream * *arg_yyout* = 0)

10.190.2.2 yyFlexLexer::~yyFlexLexer () [virtual]

Definition at line 919 of file dbcompare_expression_scanner.cc.

10.190.3 Member Function Documentation

10.190.3.1 int FlexLexer::debug () const [inline, inherited]

Definition at line 91 of file FlexLexer.h.

10.190.3.2 virtual void yyFlexLexer::LexerError (const char * *msg*) [protected, virtual]

10.190.3.3 int yyFlexLexer::LexerInput (char * *buf*, int *max_size*) [protected, virtual]

Definition at line 941 of file dbcompare_expression_scanner.cc.

10.190.3.4 void yyFlexLexer::LexerOutput (const char * *buf*, int *size*) [protected, virtual]

Definition at line 968 of file dbcompare_expression_scanner.cc.

10.190.3.5 int FlexLexer::lineno () const [inline, inherited]

Definition at line 89 of file FlexLexer.h.

10.190.3.6 void FlexLexer::set_debug (int *flag*) [inline, inherited]

Definition at line 92 of file FlexLexer.h.

10.190.3.7 virtual void yyFlexLexer::switch_streams (FLEX_STD istream * *new_in*, FLEX_STD ostream * *new_out*) [virtual]

Implements FlexLexer.

10.190.3.8 struct yy_buffer_state* yyFlexLexer::yy_create_buffer (FLEX_STD istream * *s*, int *size*) [read, virtual]

Implements FlexLexer.

10.190.3.9 void yyFlexLexer::yy_delete_buffer (struct yy_buffer_state * *b*) [virtual]

Implements FlexLexer.

10.190.3.10 void yyFlexLexer::yy_flush_buffer (struct yy_buffer_state * *b*) [protected]

10.190.3.11 int yyFlexLexer::yy_get_next_buffer () [protected]

Definition at line 980 of file dbcompare_expression_scanner.cc.

10.190.3.12 yy_state_type yyFlexLexer::yy_get_previous_state () [protected]

Definition at line 1106 of file dbcompare_expression_scanner.cc.

10.190.3.13 void yyFlexLexer::yy_init_buffer (struct yy_buffer_state * *b*, FLEX_STD istream * *s*) [protected]

10.190.3.14 void yyFlexLexer::yy_load_buffer_state () [protected]

Definition at line 1318 of file dbcompare_expression_scanner.cc.

10.190.3.15 void yyFlexLexer::yy_pop_state () [protected]

Definition at line 1548 of file dbcompare_expression_scanner.cc.

10.190.3.16 void yyFlexLexer::yy_push_state (int *new_state*) [protected]

Definition at line 1523 of file dbcompare_expression_scanner.cc.

10.190.3.17 void yyFlexLexer::yy_switch_to_buffer (struct yy_buffer_state * *new_buffer*) [virtual]

Implements FlexLexer.

10.190.3.18 int yyFlexLexer::yy_top_state () [protected]

Definition at line 1556 of file dbcompare_expression_scanner.cc.

10.190.3.19 yy_state_type yyFlexLexer::yy_try_NUL_trans (yy_state_type *current_state*) [protected]

Definition at line 1138 of file dbcompare_expression_scanner.cc.

10.190.3.20 void yyFlexLexer::yyensure_buffer_stack (void) [protected]

Definition at line 1484 of file dbcompare_expression_scanner.cc.

10.190.3.21 int yyFlexLexer::yyinput () [protected]

Definition at line 1198 of file dbcompare_expression_scanner.cc.

10.190.3.22 int FlexLexer::YYLeng () [inline, inherited]

Definition at line 66 of file FlexLexer.h.

10.190.3.23 int FlexLexer::yylex (FLEX_STD istream * *new_in*, FLEX_STD ostream * *new_out* = 0) [inline, inherited]

Definition at line 78 of file FlexLexer.h.

10.190.3.24 int yyFlexLexer::yylex () [virtual]

Implements FlexLexer.

Definition at line 317 of file dbcompare_expression_scanner.cc.

10.190.3.25 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 1465 of file dbcompare_expression_scanner.cc.

10.190.3.26 void yyFlexLexer::yypush_buffer_state (struct yy_buffer_state * *new_buffer*)

10.190.3.27 void yyFlexLexer::yyrestart (FLEX_STD istream * *s*) [virtual]

Implements FlexLexer.

10.190.3.28 const char* FlexLexer::YYText () [inline, inherited]

Definition at line 65 of file FlexLexer.h.

10.190.3.29 void yyFlexLexer::yyunput (int *c*, char * *buf_ptr*) [protected]

10.190.4 Member Data Documentation

10.190.4.1 struct yy_buffer_state** yyFlexLexer::yy_buffer_stack [read, protected]

Stack as an array.

Definition at line 177 of file FlexLexer.h.

10.190.4.2 size_t yyFlexLexer::yy_buffer_stack_max [protected]

capacity of stack.

Definition at line 176 of file FlexLexer.h.

10.190.4.3 size_t yyFlexLexer::yy_buffer_stack_top [protected]

index of top of stack.

Definition at line 175 of file FlexLexer.h.

10.190.4.4 char* yyFlexLexer::yy_c_buf_p [protected]

Definition at line 165 of file FlexLexer.h.

10.190.4.5 int yyFlexLexer::yy_did_buffer_switch_on_eof [protected]

Definition at line 172 of file FlexLexer.h.

10.190.4.6 int FlexLexer::yy_flex_debug [protected, inherited]

Definition at line 98 of file FlexLexer.h.

10.190.4.7 int yyFlexLexer::yy_full_lp [protected]

Definition at line 191 of file FlexLexer.h.

10.190.4.8 char* yyFlexLexer::yy_full_match [protected]

Definition at line 189 of file FlexLexer.h.

10.190.4.9 int* yyFlexLexer::yy_full_state [protected]

Definition at line 190 of file FlexLexer.h.

10.190.4.10 char yyFlexLexer::yy_hold_char [protected]

Definition at line 159 of file FlexLexer.h.

10.190.4.11 int yyFlexLexer::yy_init [protected]

Definition at line 167 of file FlexLexer.h.

10.190.4.12 char* yyFlexLexer::yy_last_accepting_cpos [protected]

Definition at line 184 of file FlexLexer.h.

10.190.4.13 yy_state_type yyFlexLexer::yy_last_accepting_state [protected]

Definition at line 183 of file FlexLexer.h.

10.190.4.14 int yyFlexLexer::yy_looking_for_trail_begin [protected]

Definition at line 194 of file FlexLexer.h.

10.190.4.15 int yyFlexLexer::yy_lp [protected]

Definition at line 193 of file FlexLexer.h.

10.190.4.16 int yyFlexLexer::yy_more_flag [protected]

Definition at line 196 of file FlexLexer.h.

10.190.4.17 int yyFlexLexer::yy_more_len [protected]

Definition at line 197 of file FlexLexer.h.

10.190.4.18 int yyFlexLexer::yy_more_offset [protected]

Definition at line 198 of file FlexLexer.h.

10.190.4.19 int yyFlexLexer::yy_n_chars [protected]

Definition at line 162 of file FlexLexer.h.

10.190.4.20 int yyFlexLexer::yy_prev_more_offset [protected]

Definition at line 199 of file FlexLexer.h.

10.190.4.21 int yyFlexLexer::yy_start [protected]

Definition at line 168 of file FlexLexer.h.

10.190.4.22 int* yyFlexLexer::yy_start_stack [protected]

Definition at line 145 of file FlexLexer.h.

10.190.4.23 int yyFlexLexer::yy_start_stack_depth [protected]

Definition at line 144 of file FlexLexer.h.

10.190.4.24 int yyFlexLexer::yy_start_stack_ptr [protected]

Definition at line 143 of file FlexLexer.h.

10.190.4.25 yy_state_type* yyFlexLexer::yy_state_buf [protected]

Definition at line 186 of file FlexLexer.h.

10.190.4.26 yy_state_type* yyFlexLexer::yy_state_ptr [protected]

Definition at line 187 of file FlexLexer.h.

10.190.4.27 FLEX_STD istream* yyFlexLexer::yyin [protected]

Definition at line 155 of file FlexLexer.h.

10.190.4.28 int FlexLexer::yyleng [protected, inherited]

Definition at line 96 of file FlexLexer.h.

10.190.4.29 int FlexLexer::yylineno [protected, inherited]

Definition at line 97 of file FlexLexer.h.

10.190.4.30 FLEX_STD ostream* yyFlexLexer::yyout [protected]

Definition at line 156 of file FlexLexer.h.

10.190.4.31 char* FlexLexer::yytext [protected, inherited]

Definition at line 95 of file FlexLexer.h.

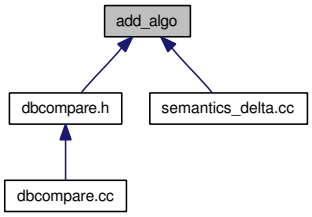
The documentation for this class was generated from the following files:

- FlexLexer.h
- dbcompare_expression_scanner.cc
- dbcompare_sortorder_scanner.cc

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

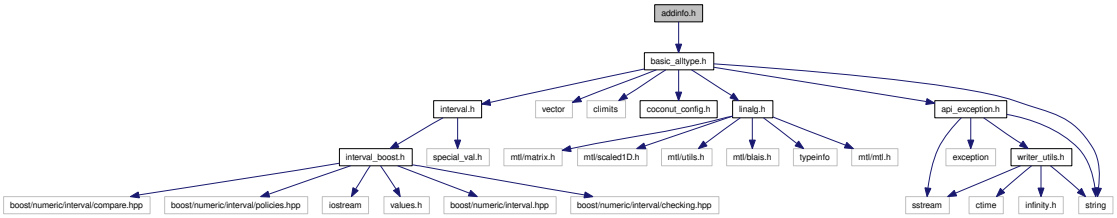
11.1.2.1 #define _CPP_ADD_ALGO 1

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

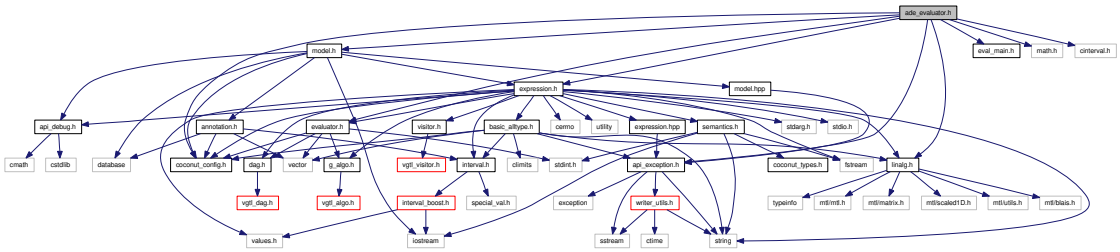
11.2.1 Detailed Description

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 <linalg.h>
#include <math.h>
#include <cinterval.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

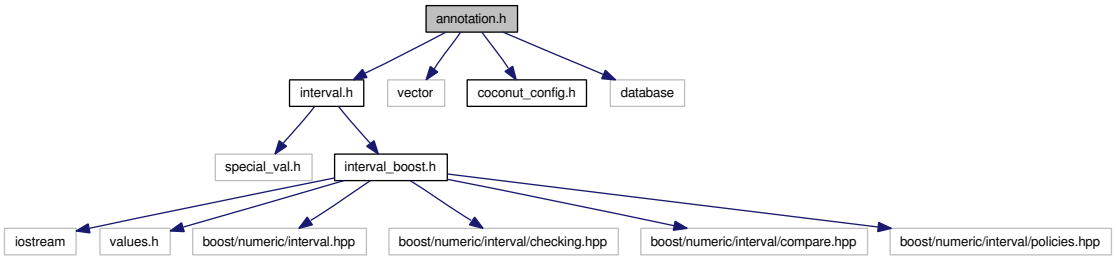
```
#include <interval.h>

#include <vector>

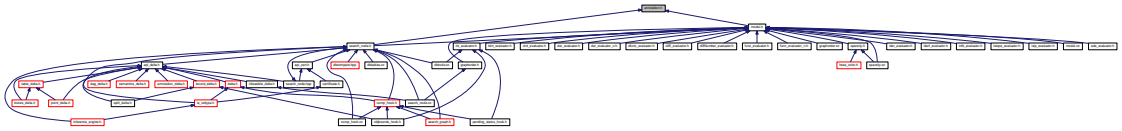
#include <coconut_config.h>

#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

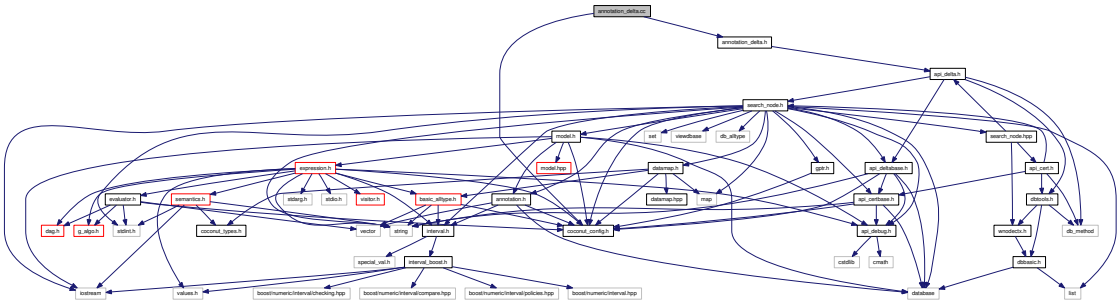
Definition in file annotation.h.

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

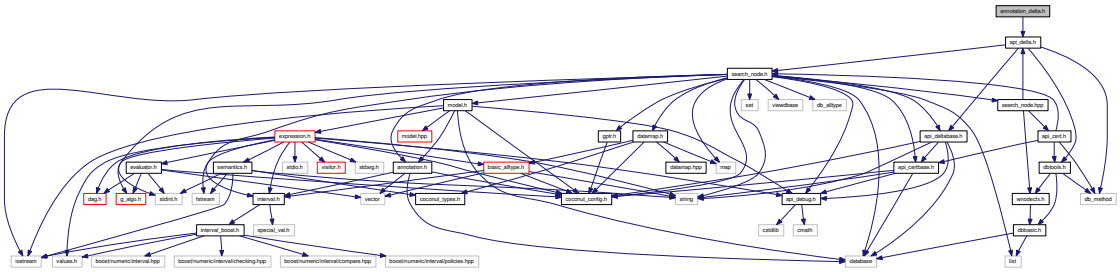
11.5.1 Detailed Description

Definition in file annotation_delta.cc.

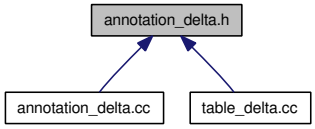
11.6 annotation_delta.h File Reference

```
#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

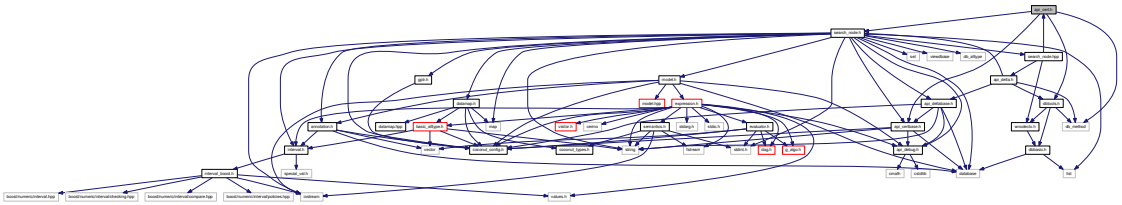
11.6.1 Detailed Description

Definition in file annotation_delta.h.

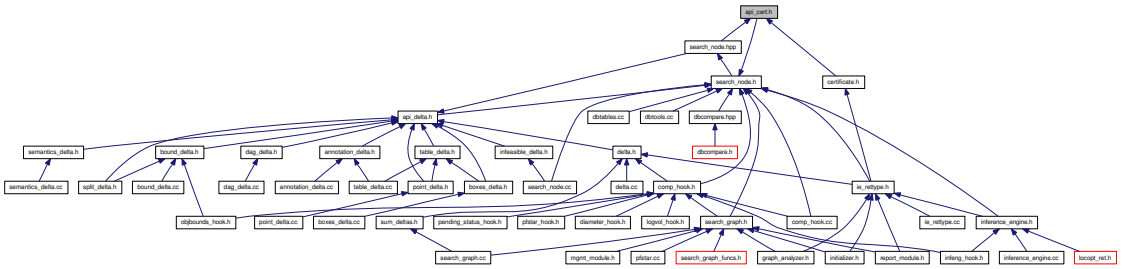
11.7 api_cert.h File Reference

```
#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

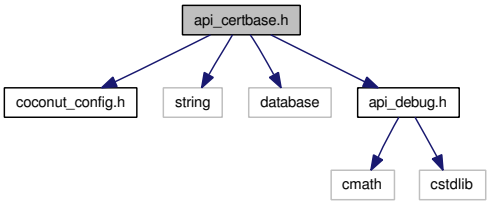
```
#include <coconut_config.h>

#include <string>

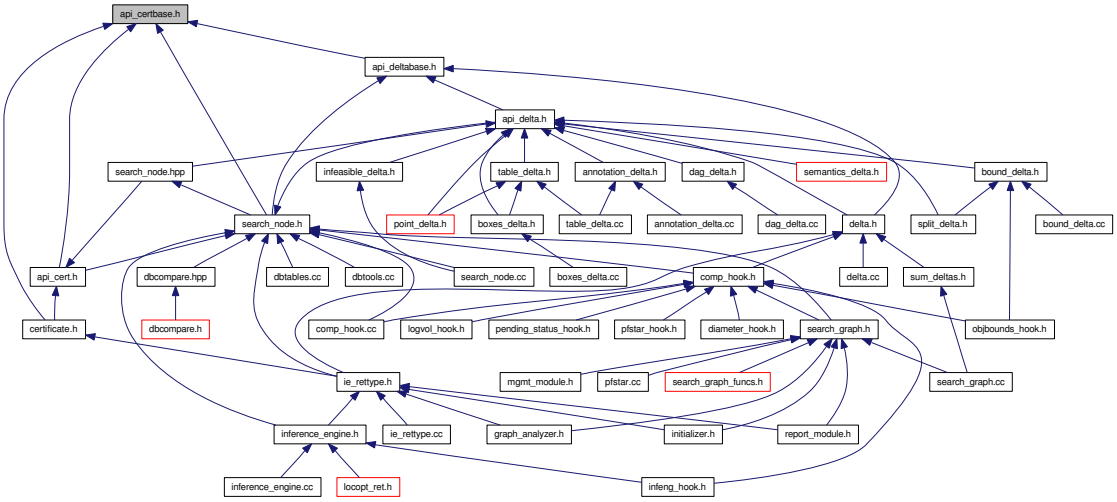
#include <database>

#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

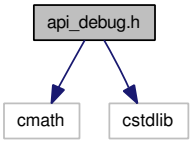
Definition in file api_certbase.h.

11.9 api_debug.h File Reference

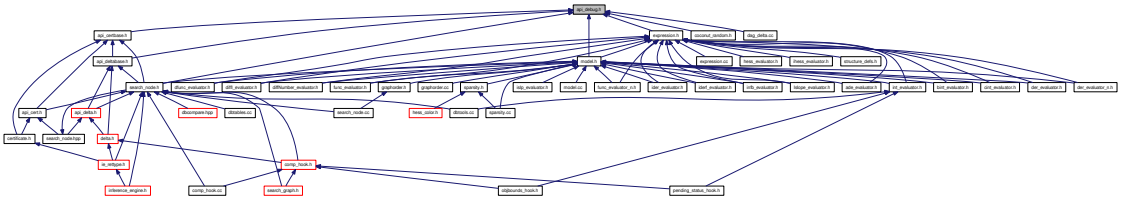
```
#include <cmath>
```

```
#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.

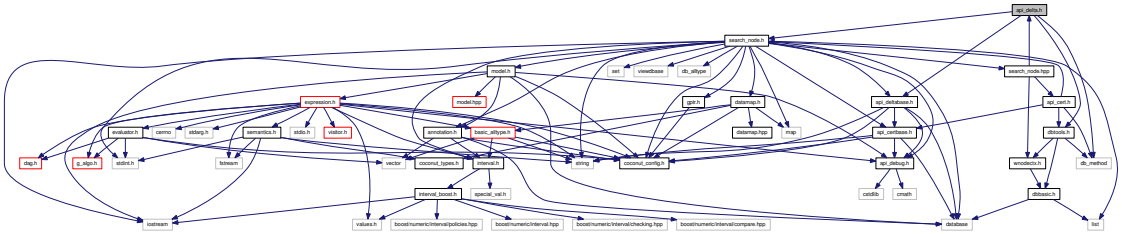
11.9.2.8 #define MODEL_INLINE_DEBUG_SIMPLIFY 0

Definition at line 65 of file api_debug.h.

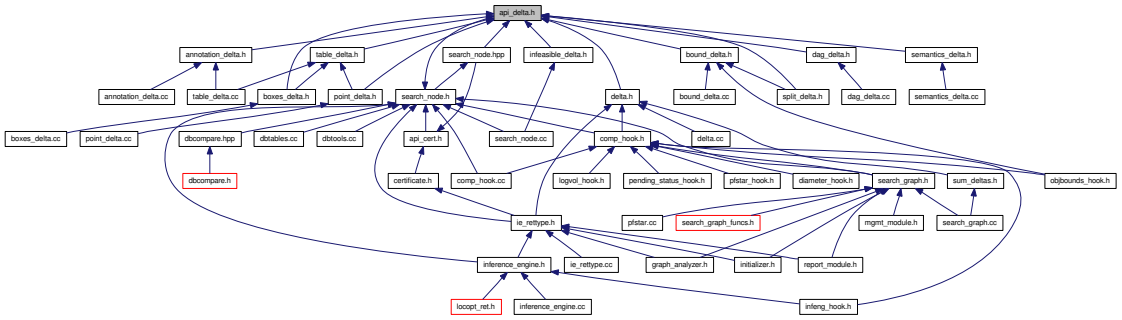
11.10 api_delta.h File Reference

```
#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

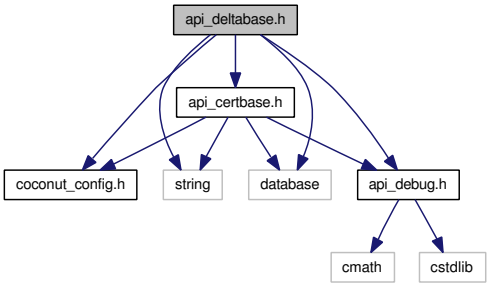
This is an internal header file, which is not intended for use outside the API.

Definition in file api_delta.h.

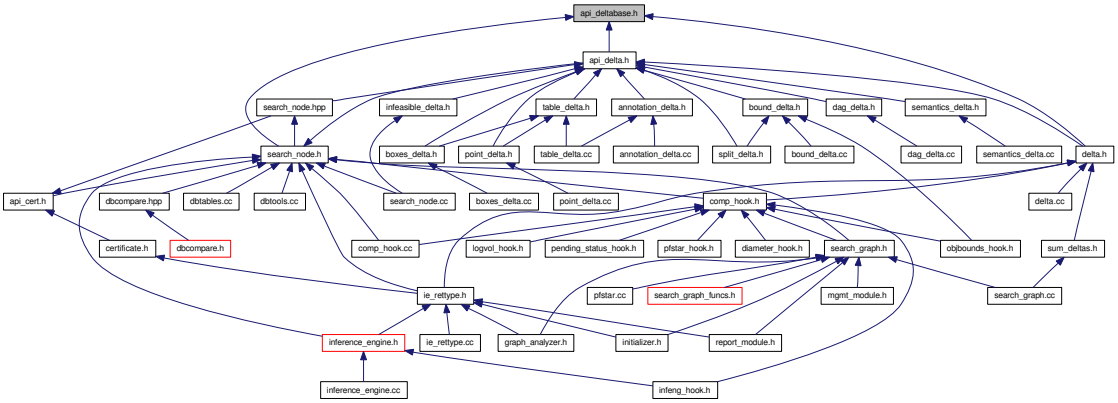
11.11 api_deltabase.h File Reference

```
#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

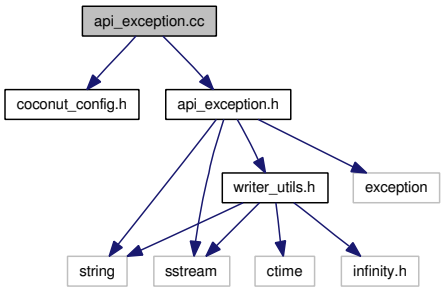
This is an internal header file not intended for use outside the API

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

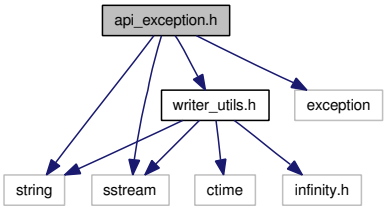
11.12.1 Detailed Description

Definition in file api_exception.cc.

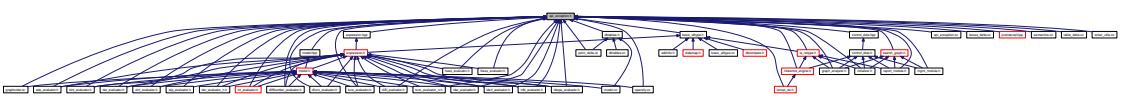
11.13 api_exception.h File Reference

```
#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

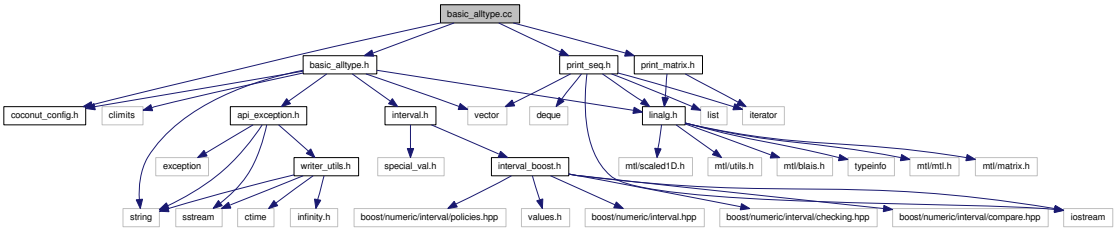
11.14.1 Detailed Description

Definition in file api_extradocu.h.

11.15 basic_alltype.cc File Reference

```
#include <coconut_config.h>
#include <basic_alltype.h>
#include <print_seq.h>
#include <print_matrix.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)

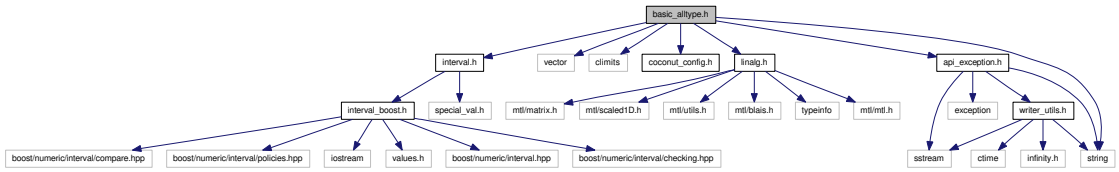
11.15.1 Detailed Description

Definition in file basic_alltype.cc.

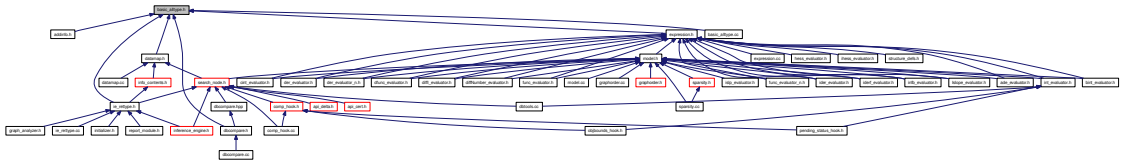
11.16 basic_alltype.h File Reference

```
#include <string>
#include <vector>
#include <climits>
#include <coconut_config.h>
#include <interval.h>
#include <linalg.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 -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_S 7
- #define ALLTYPE_ALLOCED_M 8
- #define ALLTYPE_ALLOCED_NM 9
- #define ALLTYPE_ALLOCED_IM 10

- #define ALLTYPE_ALLOCED_SM 11
- #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)

11.16.1 Detailed Description

Definition in file basic_alltype.h.

11.16.2 Define Documentation

11.16.2.1 #define ALLTYPE_ALLOCED_B 1

a vector<bool> is stored

Definition at line 59 of file basic_alltype.h.

11.16.2.2 #define ALLTYPE_ALLOCED_C 6

a vector<string> is stored

Definition at line 64 of file basic_alltype.h.

11.16.2.3 #define ALLTYPE_ALLOCED_D 4

a vector<double> is stored

Definition at line 62 of file basic_alltype.h.

11.16.2.4 #define ALLTYPE_ALLOCED_I 5

a vector<interval> is stored

Definition at line 63 of file basic_alltype.h.

11.16.2.5 #define ALLTYPE_ALLOCED_IM 10

a matrix<interval> is stored

Definition at line 70 of file basic_alltype.h.

11.16.2.6 #define ALLTYPE_ALLOCED_M 8

a matrix<double> is stored

Definition at line 68 of file basic_alltype.h.

11.16.2.7 #define ALLTYPE_ALLOCED_N 2

a vector<int> is stored

Definition at line 60 of file basic_alltype.h.

11.16.2.8 #define ALLTYPE_ALLOCED_NM 9

a matrix<int> is stored

Definition at line 69 of file basic_alltype.h.

11.16.2.9 #define ALLTYPE_ALLOCED_S 7

a string is stored

Definition at line 66 of file basic_alltype.h.

11.16.2.10 #define ALLTYPE_ALLOCED_SM 11

a matrix<string> is stored

Definition at line 71 of file basic_alltype.h.

11.16.2.11 #define ALLTYPE_ALLOCED_U 3

a vector<unsigned int> is stored

Definition at line 61 of file basic_alltype.h.

11.16.2.12 #define ALLTYPE_ANY 99

an unspecified type is stored

Definition at line 73 of file basic_alltype.h.

11.16.2.13 #define ALLTYPE_BOOL -1

a bool is stored

Definition at line 55 of file basic_alltype.h.

11.16.2.14 #define ALLTYPE_DOUBLE -4

a double is stored

Definition at line 52 of file basic_alltype.h.

11.16.2.15 #define ALLTYPE_EMPTY 0

the basic_alltype is empty

Definition at line 57 of file basic_alltype.h.

11.16.2.16 #define ALLTYPE_INT -2

an int is stored

Definition at line 54 of file basic_alltype.h.

11.16.2.17 #define ALLTYPE_INTERVAL -5

an interval is stored

Definition at line 51 of file basic_alltype.h.

11.16.2.18 #define ALLTYPE_UINT -3

an unsigned int is stored

Definition at line 53 of file basic_alltype.h.

11.16.2.19 #define BASIC_ALLTYPE_NAMES

Value:

```
{ \
      "interval", "double", "unsigned int", "int", "bool", "empty", \
      "bool vector", "int vector", "unsigned int vector", "double vector", \
      "interval vector", "string vector", "string", "double matrix", \
      "int matrix", "interval matrix", "string matrix" }
```

Definition at line 75 of file basic_alltype.h.

11.16.2.20 #define BASIC_ALLTYPE_THROW throw(api_exception)

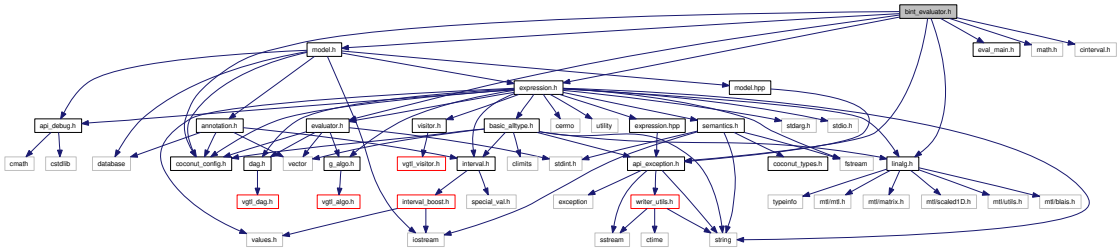
Definition at line 105 of file basic_alltype.h.

11.17 bint_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <model.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <cinterval.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)

11.17.1 Detailed Description

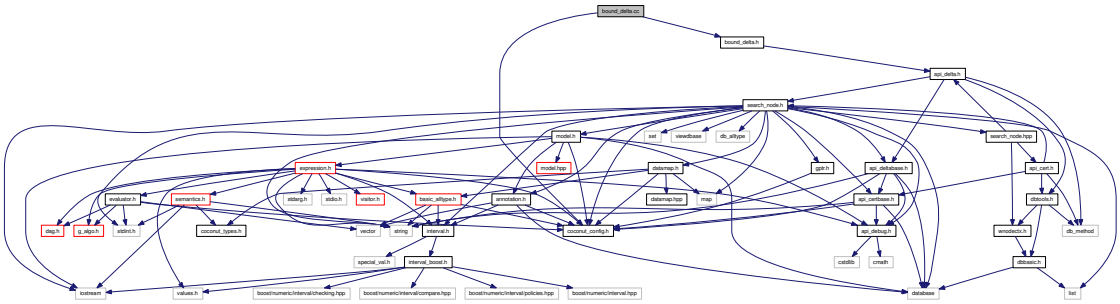
Definition in file bint_evaluator.h.

11.18 bound_delta.cc File Reference

```
#include <coconut_config.h>

#include <bound_delta.h>
```

Include dependency graph for bound_delta.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

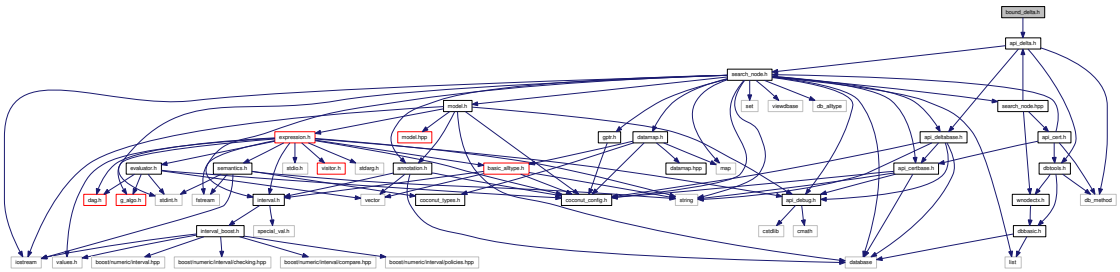
11.18.1 Detailed Description

Definition in file bound_delta.cc.

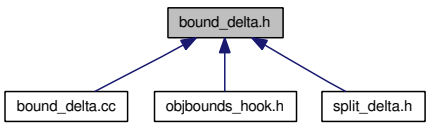
11.19 bound_delta.h File Reference

```
#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

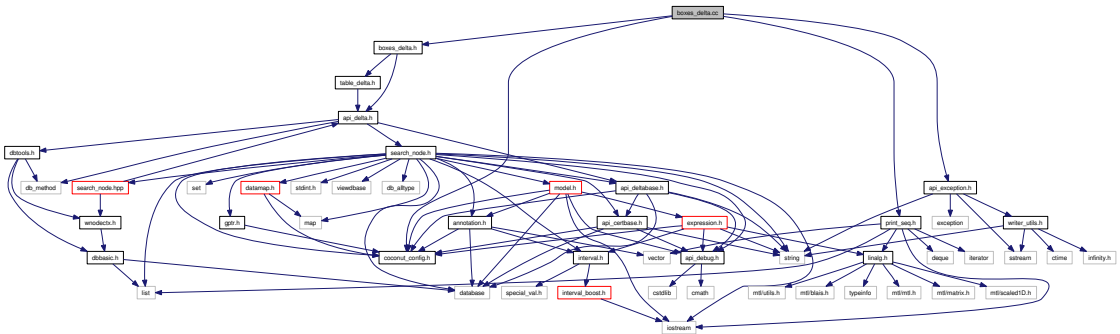
11.19.1 Detailed Description

Definition in file bound_delta.h.

11.20 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

11.20.1 Detailed Description

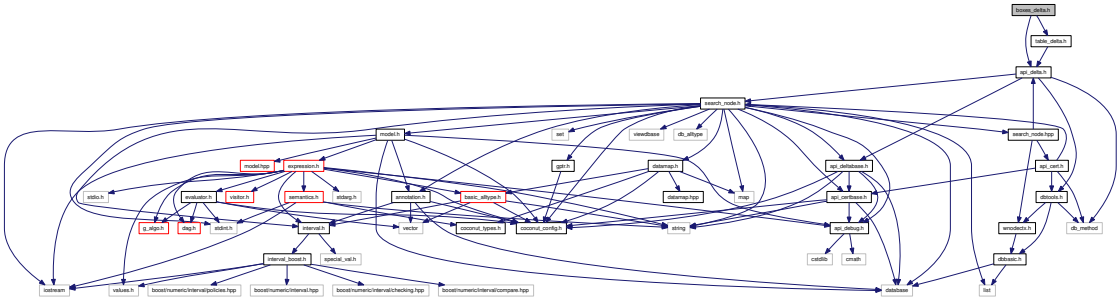
Definition in file boxes_delta.cc.

11.21 boxes_delta.h File Reference

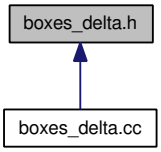
```
#include <api_delta.h>
```

```
#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

11.21.1 Detailed Description

Definition in file boxes_delta.h.

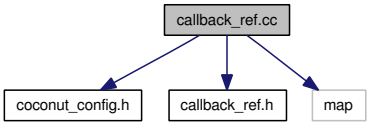
11.22 callback_ref.cc File Reference

```
#include <coconut_config.h>

#include <callback_ref.h>

#include <map>
```

Include dependency graph for callback_ref.cc:



Functions

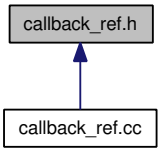
- cb_ref_handle _new_cb_ref (void *dest)
- int delete_cb_ref (cb_ref_handle ref)
- void * cb_ref (cb_ref_handle ref)

11.22.1 Detailed Description

Definition in file callback_ref.cc.

11.23 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)

11.23.1 Detailed Description

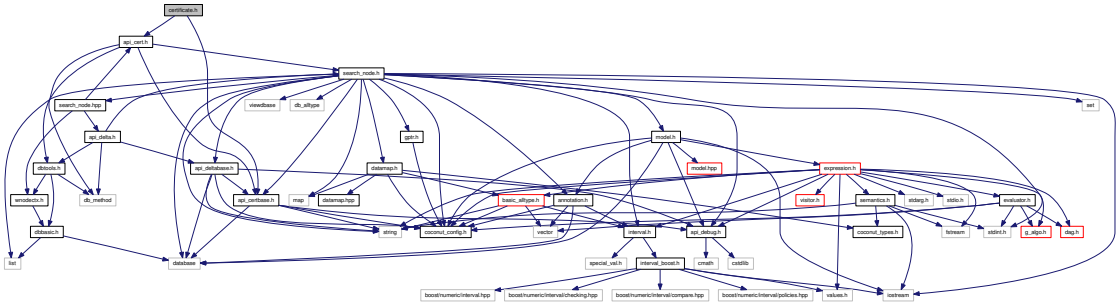
Definition in file callback_ref.h.

11.24 certificate.h File Reference

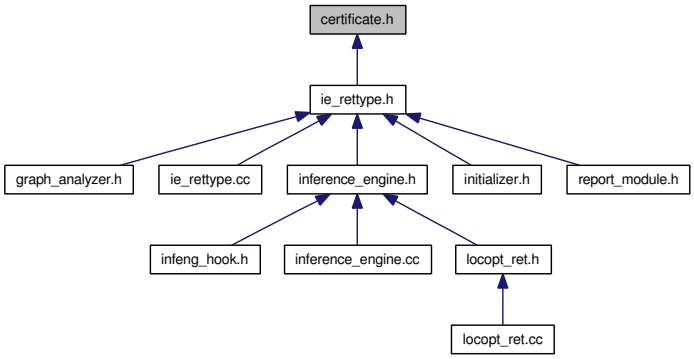
```
#include <api_certbase.h>
```

```
#include <api_cert.h>
```

Include dependency graph for certificate.h:



This graph shows which files directly or indirectly include this file:



11.24.1 Detailed Description

Definition in file certificate.h.

11.25 cint_evaluator.h File Reference

```
#include <coconut_config.h>
```

```
#include <evaluator.h>
```

```
#include <expression.h>
```

```
#include <model.h>

#include <eval_main.h>

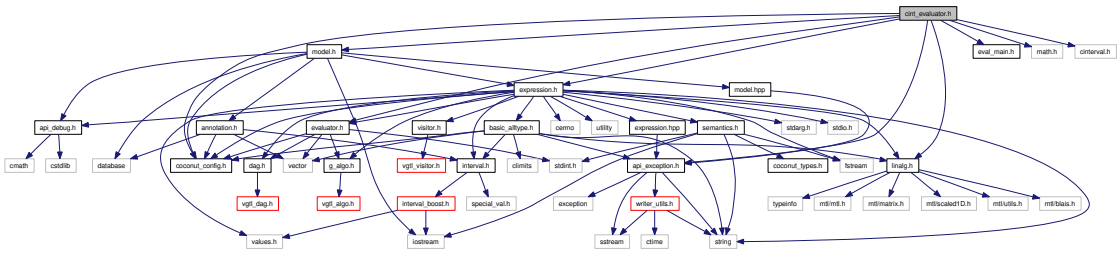
#include <linalg.h>

#include <math.h>

#include <cinterval.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)

11.25.1 Detailed Description

Definition in file cint_evaluator.h.

11.26 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__

11.26.1 Detailed Description

Definition in file coconut_config.h.

11.26.2 Define Documentation

11.26.2.1 #define __COCONUT_PRETTY_FUNCTION__ __PRETTY_FUNCTION__

Definition at line 56 of file coconut_config.h.

11.26.2.2 #define COCONUT_INTERVAL_LIBRARY INTERVAL_LIBRARY_FILIB

Definition at line 45 of file coconut_config.h.

11.26.2.3 #define COCONUT_LINUX 1

Definition at line 47 of file coconut_config.h.

11.26.2.4 #define INTERVAL_LIBRARY_BOOST 1

Definition at line 43 of file coconut_config.h.

11.26.2.5 #define INTERVAL_LIBRARY_FILIB 0

Definition at line 42 of file coconut_config.h.

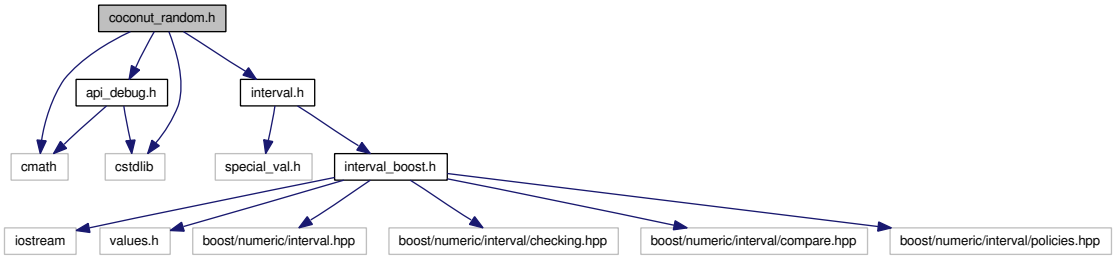
11.27 coconut_random.h File Reference

```
#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 ()

11.27.1 Detailed Description

Definition in file coconut_random.h.

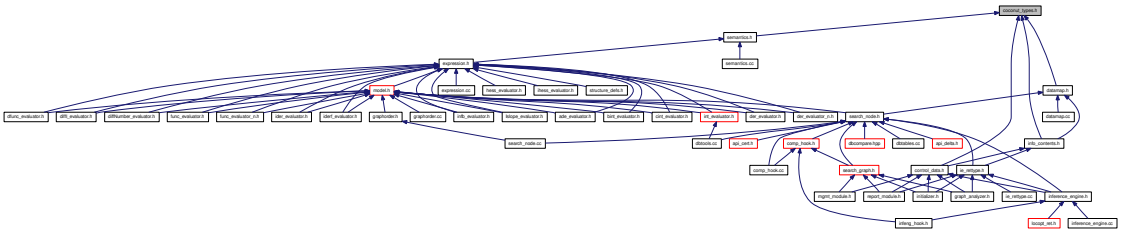
11.27.2 Define Documentation

11.27.2.1 #define INIT_SEED coconut_random()

Definition at line 79 of file coconut_random.h.

11.28 coconut_types.h File Reference

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

Enumerations

- enum coco::tristate_e { coco::t_true = 1, coco::t_false = -1, coco::t_maybe = 0 }

11.28.1 Detailed Description

Definition in file coconut_types.h.

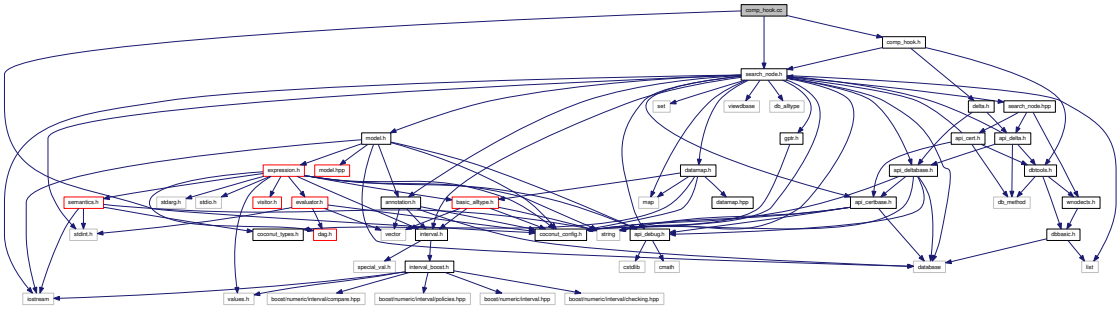
11.29 comp_hook.cc File Reference

```
#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

11.29.1 Detailed Description

Definition in file comp_hook.cc.

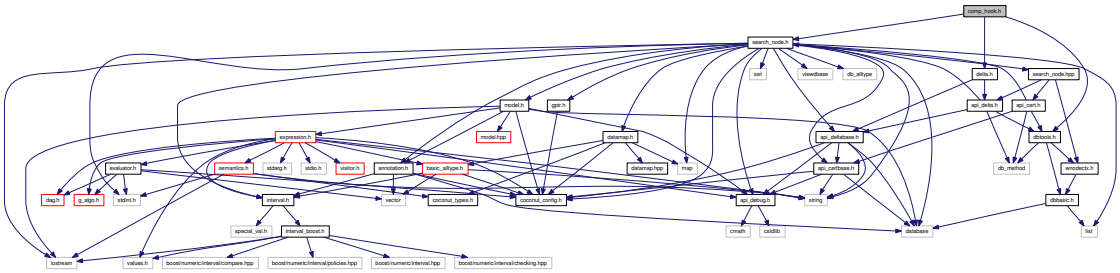
11.30 comp_hook.h File Reference

```
#include <dbtools.h>
```

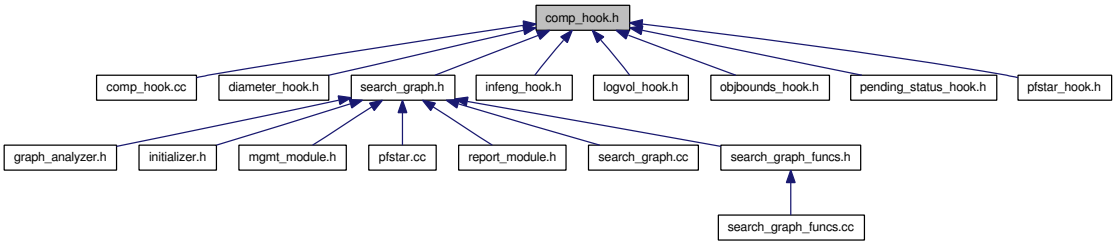
```
#include <search_node.h>
```

```
#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

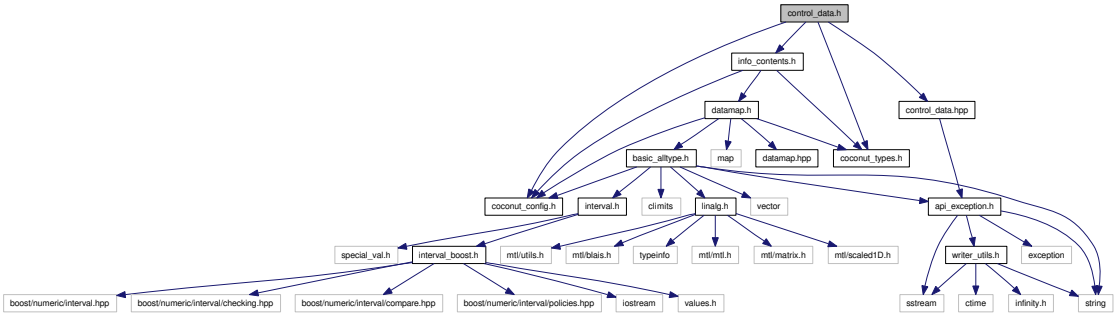
11.30.1 Detailed Description

Definition in file comp_hook.h.

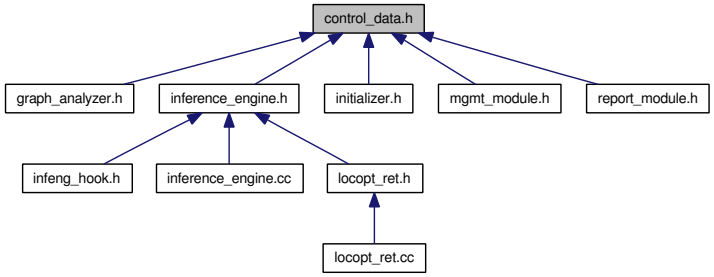
11.31 control_data.h File Reference

```
#include <coconut_config.h>
#include <coconut_types.h>
#include <info_contents.h>
#include <control_data.hpp>
```

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

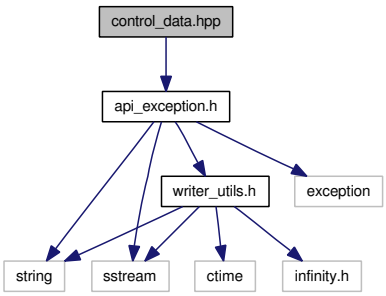
11.31.1 Detailed Description

Definition in file control_data.h.

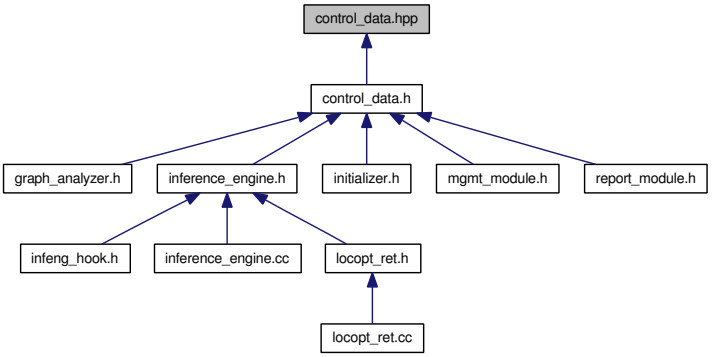
11.32 control_data.hpp File Reference

```
#include <api_exception.h>
```

Include dependency graph for control_data.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

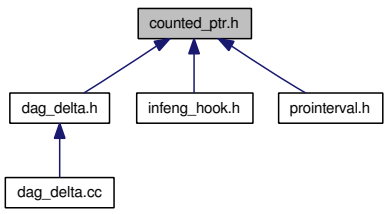
- namespace coco
  the main namespace of the COCONUT API

11.32.1 Detailed Description

Definition in file control_data.hpp.

11.33 counted_ptr.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class counted_ptr< _Tp >
- class **counted_ptr< _Tp >::counted_ptr::counter**

Defines

- #define DEBUG_COUNTED_PTR 0

11.33.1 Detailed Description

Definition in file counted_ptr.h.

11.33.2 Define Documentation

11.33.2.1 #define DEBUG_COUNTED_PTR 0

Definition at line 31 of file counted_ptr.h.

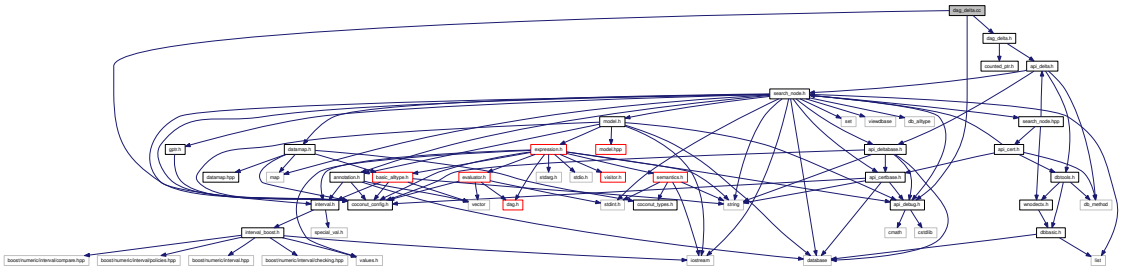
11.34 dag_delta.cc File Reference

```
#include <coconut_config.h>

#include <dag_delta.h>

#include <api_debug.h>
```

Include dependency graph for dag_delta.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

11.34.1 Detailed Description

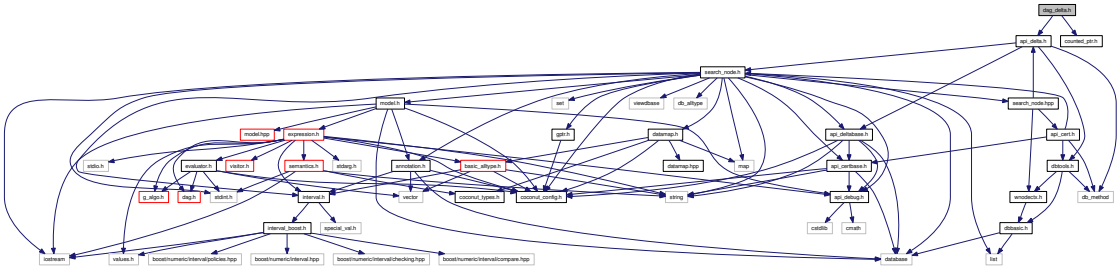
Definition in file dag_delta.cc.

11.35 dag_delta.h File Reference

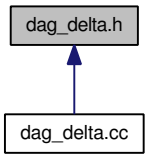
```
#include <api_delta.h>
```

```
#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::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::dag_delta::__docompare_nodes**
- class **coco::dag_delta::dag_delta::__check_nodes**

Namespaces

- namespace coco

  the main namespace of the COCONUT API

11.35.1 Detailed Description

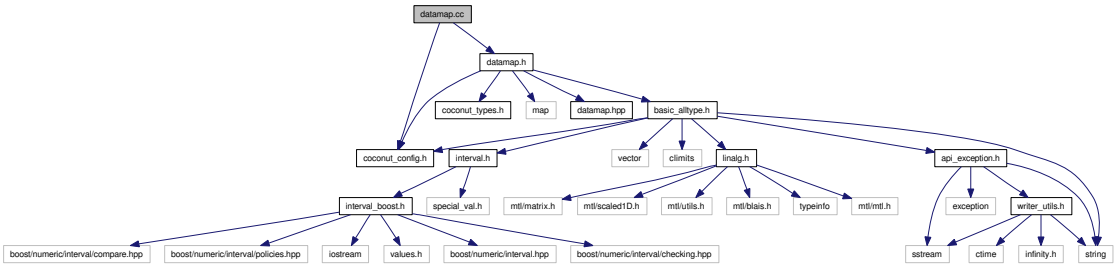
Definition in file dag_delta.h.

11.36 datamap.cc File Reference

```
#include <coconut_config.h>
```

```
#include <datamap.h>
```

Include dependency graph for datamap.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

11.36.1 Detailed Description

Definition in file datamap.cc.

11.37 datamap.h File Reference

```
#include <coconut_config.h>
```

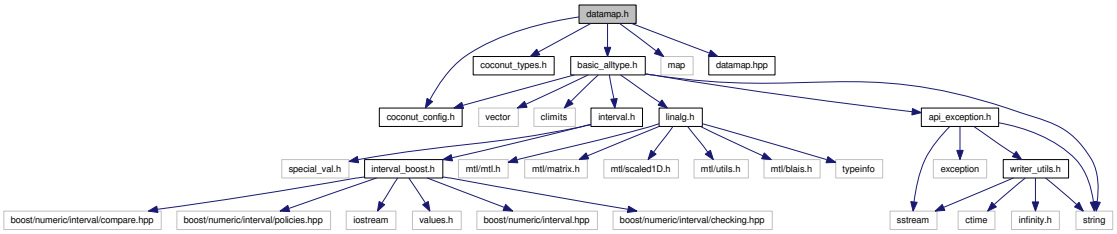
```
#include <coconut_types.h>
```

```
#include <basic_alltype.h>
```

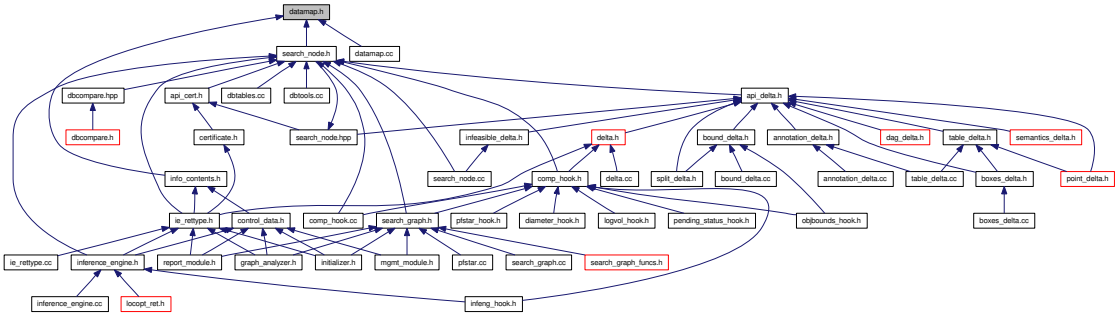
```
#include <map>
```

```
#include <datamap.hpp>
```

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

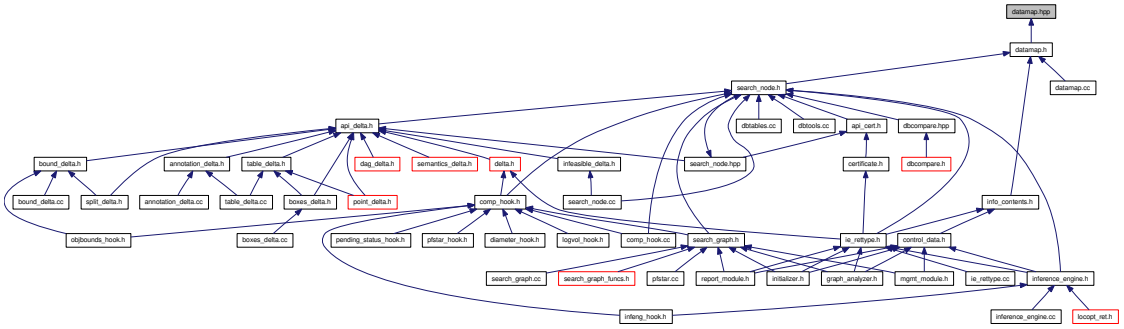
  the main namespace of the COCONUT API

11.37.1 Detailed Description

Definition in file datamap.h.

11.38 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"

11.38.1 Detailed Description

Definition in file datamap.hpp.

11.38.2 Define Documentation

11.38.2.1 #define __DATAMAP_IDXSPEC "IDX"

Definition at line 34 of file datamap.hpp.

11.38.2.2 #define __DATAMAP_SEPARATOR "%"

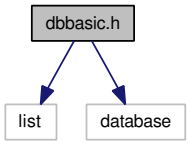
Definition at line 33 of file datamap.hpp.

11.39 dbbasic.h File Reference

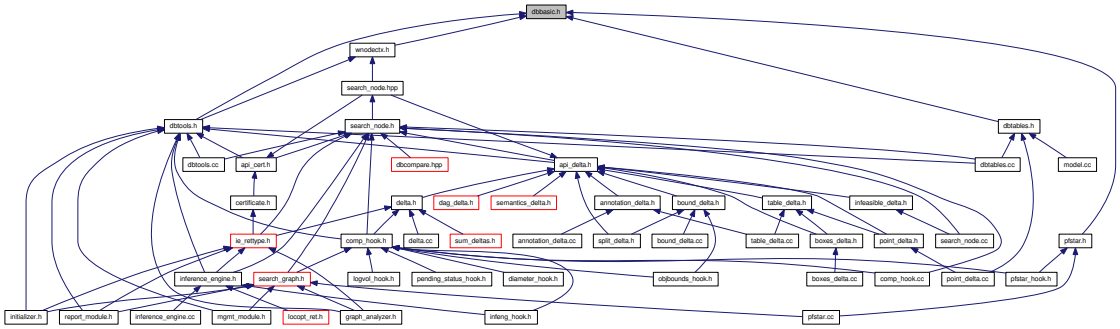
```
#include <list>
```

```
#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

11.39.1 Detailed Description

Definition in file dbbasic.h.

11.40 dbcompare.cc File Reference

```
#include <coconut_config.h>
```

```
#include <interval.h>

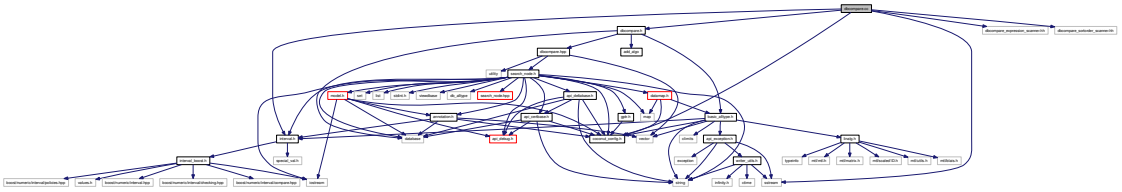
#include <dbcompare.h>

#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)

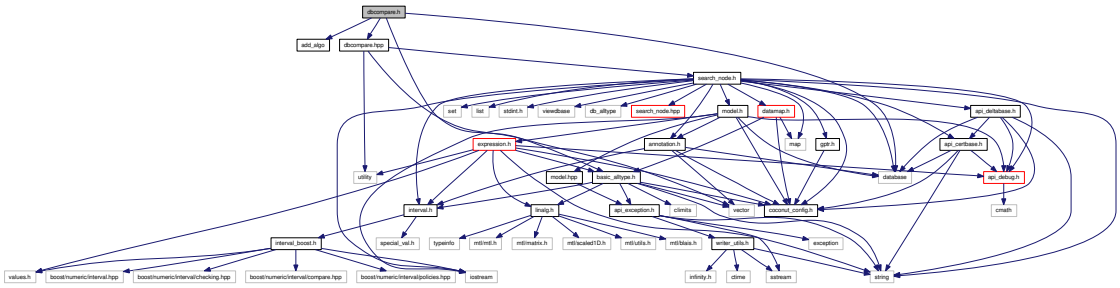
11.40.1 Detailed Description

Definition in file dbcompare.cc.

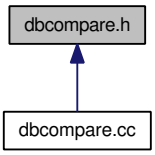
11.41 dbcompare.h File Reference

```
#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)

11.41.1 Detailed Description

Definition in file dbcompare.h.

11.41.2 Define Documentation

11.41.2.1 #define DB_COMPARE_CMP_ABSEQ 7

Definition at line 45 of file dbcompare.h.

11.41.2.2 #define DB_COMPARE_CMP_ABSGE 10

Definition at line 48 of file dbcompare.h.

11.41.2.3 #define DB_COMPARE_CMP_ABSGT 9

Definition at line 47 of file dbcompare.h.

11.41.2.4 #define DB_COMPARE_CMP_ABSIN 13

Definition at line 51 of file dbcompare.h.

11.41.2.5 #define DB_COMPARE_CMP_ABSLE 12

Definition at line 50 of file dbcompare.h.

11.41.2.6 #define DB_COMPARE_CMP_ABSLT 11

Definition at line 49 of file dbcompare.h.

11.41.2.7 #define DB_COMPARE_CMP_ABSNE 8

Definition at line 46 of file dbcompare.h.

11.41.2.8 #define DB_COMPARE_CMP_EQ 0

Definition at line 37 of file dbcompare.h.

11.41.2.9 #define DB_COMPARE_CMP_GE 3

Definition at line 40 of file dbcompare.h.

11.41.2.10 #define DB_COMPARE_CMP_GT 2

Definition at line 39 of file dbcompare.h.

11.41.2.11 #define DB_COMPARE_CMP_IN 6

Definition at line 43 of file dbcompare.h.

11.41.2.12 #define DB_COMPARE_CMP_LE 5

Definition at line 42 of file dbcompare.h.

11.41.2.13 #define DB_COMPARE_CMP_LT 4

Definition at line 41 of file dbcompare.h.

11.41.2.14 #define DB_COMPARE_CMP_NE 1

Definition at line 38 of file dbcompare.h.

11.41.2.15 #define DB_COMPARE_SORT_ABSMAX 17

Definition at line 60 of file dbcompare.h.

11.41.2.16 #define DB_COMPARE_SORT_ABSMIN 16

Definition at line 59 of file dbcompare.h.

11.41.2.17 #define DB_COMPARE_SORT_FALSE 19

Definition at line 62 of file dbcompare.h.

11.41.2.18 #define DB_COMPARE_SORT_MAX 15

Definition at line 58 of file dbcompare.h.

11.41.2.19 #define DB_COMPARE_SORT_MIN 14

Definition at line 57 of file dbcompare.h.

11.41.2.20 #define DB_COMPARE_SORT_TRUE 18

Definition at line 61 of file dbcompare.h.

11.41.2.21 #define dbc_is_cmp(X) ((X) <= DB_COMPARE_CMP_ABSIN)

Definition at line 53 of file dbcompare.h.

11.41.2.22 #define dbc_is_sort(X) ((X) > DB_COMPARE_CMP_ABSIN)

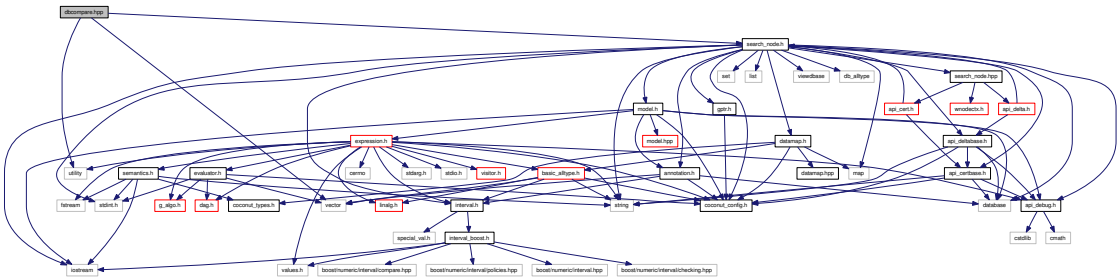
Definition at line 55 of file dbcompare.h.

11.42 dbcompare.hpp File Reference

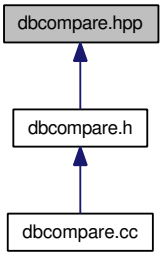
```
#include <vector>
#include <utility>
```

```
#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< _TR >
- class coco::dbccmps_absgt< _TR >
- class coco::dbccmps_lt< _TR >
- class coco::dbccmps_abslt< _TR >
- class coco::dbccmps_true< _TR >
- class coco::dbccmps_false< _TR >
- struct coco::dbccmp_true< _TR >
- struct coco::dbccmp_false< _TR >

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)

11.42.1 Detailed Description

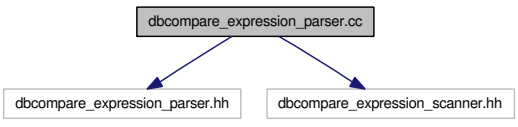
Definition in file dbcompare.hpp.

11.43 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

11.43.1 Define Documentation

11.43.1.1 #define YY_(msgid) msgid

Definition at line 74 of file dbcompare_expression_parser.cc.

11.43.1.2 #define YY_REDUCE_PRINT(Rule)

Definition at line 114 of file dbcompare_expression_parser.cc.

11.43.1.3 #define YY_STACK_PRINT()

Definition at line 115 of file dbcompare_expression_parser.cc.

11.43.1.4 #define YY_SYMBOL_PRINT(Title, Type, Value, Location)

Definition at line 113 of file dbcompare_expression_parser.cc.

11.43.1.5 #define YYABORT goto yyabortlab

Definition at line 120 of file dbcompare_expression_parser.cc.

11.43.1.6 #define YYACCEPT goto yyacceptlab

Definition at line 119 of file dbcompare_expression_parser.cc.

11.43.1.7 #define YYCDEBUG

Value:

```
for (bool yydebugcond_ = yydebug_; yydebugcond_; yydebugcond_ = false)  \
    (*yycdebug_)
```

Definition at line 82 of file dbcompare_expression_parser.cc.

11.43.1.8 #define YYERROR goto yyerrorlab

Definition at line 121 of file dbcompare_expression_parser.cc.

11.43.1.9 #define yylex scanner.yylex

Definition at line 45 of file dbcompare_expression_parser.cc.

11.43.1.10 #define yylex coco_api_internallex

Definition at line 45 of file dbcompare_expression_parser.cc.

11.43.1.11 #define YYUSE(e) ((void) (e))

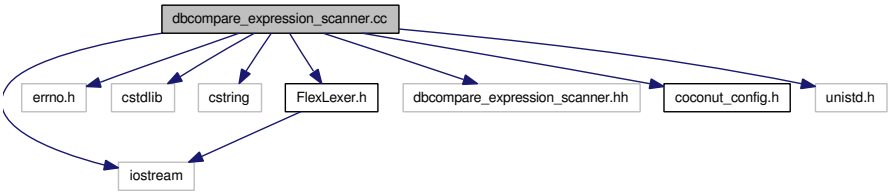
Definition at line 79 of file dbcompare_expression_parser.cc.

11.44 dbcompare_expression_scanner.cc File Reference

```
#include <iostream>
#include <errno.h>
#include <cstdlib>
#include <cstring>
#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 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 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_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_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

11.44.1 Define Documentation

11.44.1.1 #define BEGIN (yy_start) = 1 + 2 *

Definition at line 135 of file dbcompare_expression_scanner.cc.

11.44.1.2 #define ECHO LexerOutput(yytext, yyleng)

Definition at line 500 of file dbcompare_expression_scanner.cc.

11.44.1.3 #define EOB_ACT_CONTINUE_SCAN 0

Definition at line 168 of file dbcompare_expression_scanner.cc.

11.44.1.4 #define EOB_ACT_END_OF_FILE 1

Definition at line 169 of file dbcompare_expression_scanner.cc.

11.44.1.5 #define EOB_ACT_LAST_MATCH 2

Definition at line 170 of file dbcompare_expression_scanner.cc.

11.44.1.6 #define FLEX_BETA

Definition at line 14 of file dbcompare_expression_scanner.cc.

11.44.1.7 #define FLEX_SCANNER

Definition at line 9 of file dbcompare_expression_scanner.cc.

11.44.1.8 #define INITIAL 0

Definition at line 467 of file dbcompare_expression_scanner.cc.

11.44.1.9 #define INT16_MAX (32767)

Definition at line 77 of file dbcompare_expression_scanner.cc.

11.44.1.10 #define INT16_MIN (-32767-1)

Definition at line 68 of file dbcompare_expression_scanner.cc.

11.44.1.11 #define INT32_MAX (2147483647)

Definition at line 80 of file dbcompare_expression_scanner.cc.

11.44.1.12 #define INT32_MIN (-2147483647-1)

Definition at line 71 of file dbcompare_expression_scanner.cc.

11.44.1.13 #define INT8_MAX (127)

Definition at line 74 of file dbcompare_expression_scanner.cc.

11.44.1.14 #define INT8_MIN (-128)

Definition at line 65 of file dbcompare_expression_scanner.cc.

11.44.1.15 #define REJECT reject_used_but_not_detected

Definition at line 424 of file dbcompare_expression_scanner.cc.

11.44.1.16 #define UINT16_MAX (65535U)

Definition at line 86 of file dbcompare_expression_scanner.cc.

11.44.1.17 #define UINT32_MAX (4294967295U)

Definition at line 89 of file dbcompare_expression_scanner.cc.

11.44.1.18 #define UINT8_MAX (255U)

Definition at line 83 of file dbcompare_expression_scanner.cc.

11.44.1.19 #define unput(c) yyunput(c, (yytext_ptr))

Definition at line 188 of file dbcompare_expression_scanner.cc.

11.44.1.20 #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE → yy_at_bol)

Definition at line 307 of file dbcompare_expression_scanner.cc.

11.44.1.21 #define YY_BREAK break;

Definition at line 551 of file dbcompare_expression_scanner.cc.

11.44.1.22 #define YY_BUF_SIZE 16384

Definition at line 154 of file dbcompare_expression_scanner.cc.

11.44.1.23 #define YY_BUFFER_EOF_PENDING 2

Definition at line 261 of file dbcompare_expression_scanner.cc.

11.44.1.24 #define YY_BUFFER_NEW 0

Definition at line 249 of file dbcompare_expression_scanner.cc.

11.44.1.25 #define YY_BUFFER_NORMAL 1

Definition at line 250 of file dbcompare_expression_scanner.cc.

11.44.1.26 #define YY_CURRENT_BUFFER

Value:

```
( (yy_buffer_stack) \
                          ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
                          : NULL)
```

Definition at line 272 of file dbcompare_expression_scanner.cc.

11.44.1.27 #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]

Definition at line 279 of file dbcompare_expression_scanner.cc.

11.44.1.28 #define YY_DECL int dbcompare_expression_scanner::yylex()

Definition at line 323 of file dbcompare_expression_scanner.cc.

11.44.1.29 #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 dbcompare_expression_scanner.cc.

11.44.1.30 #define YY_END_OF_BUFFER 18

Definition at line 336 of file dbcompare_expression_scanner.cc.

11.44.1.31 #define YY_END_OF_BUFFER_CHAR 0

Definition at line 150 of file dbcompare_expression_scanner.cc.

11.44.1.32 #define YY_EXIT_FAILURE 2

Definition at line 1562 of file dbcompare_expression_scanner.cc.

11.44.1.33 #define YY_EXTRA_TYPE void *

Definition at line 478 of file dbcompare_expression_scanner.cc.

11.44.1.34 #define YY_FATAL_ERROR(msg) LexerError(msg)

Definition at line 529 of file dbcompare_expression_scanner.cc.

11.44.1.35 #define YY_FLEX_MAJOR_VERSION 2

Definition at line 10 of file dbcompare_expression_scanner.cc.

11.44.1.36 #define YY_FLEX_MINOR_VERSION 5

Definition at line 11 of file dbcompare_expression_scanner.cc.

11.44.1.37 #define YY_FLEX_SUBMINOR_VERSION 33

Definition at line 12 of file dbcompare_expression_scanner.cc.

11.44.1.38 #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 507 of file dbcompare_expression_scanner.cc.

11.44.1.39 #define YY_INT_ALIGNED short int

Definition at line 5 of file dbcompare_expression_scanner.cc.

11.44.1.40 #define YY_LESS_LINENO(n)

Definition at line 172 of file dbcompare_expression_scanner.cc.

11.44.1.41 #define YY_MORE_ADJ 0

Definition at line 426 of file dbcompare_expression_scanner.cc.

11.44.1.42 #define yy_new_buffer yy_create_buffer

Definition at line 285 of file dbcompare_expression_scanner.cc.

11.44.1.43 #define YY_NEW_FILE yyrestart(yyin)

Definition at line 148 of file dbcompare_expression_scanner.cc.

11.44.1.44 #define YY_NULL 0

Definition at line 122 of file dbcompare_expression_scanner.cc.

11.44.1.45 #define YY_NUM_RULES 17

Definition at line 335 of file dbcompare_expression_scanner.cc.

11.44.1.46 #define YY_READ_BUF_SIZE 8192

Definition at line 495 of file dbcompare_expression_scanner.cc.

11.44.1.47 #define YY_RESTORE_YY_MORE_OFFSET

Definition at line 427 of file dbcompare_expression_scanner.cc.

11.44.1.48 #define YY_RULE_SETUP YY_USER_ACTION

Definition at line 554 of file dbcompare_expression_scanner.cc.

11.44.1.49 #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)

Definition at line 129 of file dbcompare_expression_scanner.cc.

11.44.1.50 #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 dbcompare_expression_scanner.cc.

11.44.1.51 #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 dbcompare_expression_scanner.cc.

11.44.1.52 #define YY_SKIP_YYWRAP

Definition at line 310 of file dbcompare_expression_scanner.cc.

11.44.1.53 #define YY_START (((yy_start) - 1) / 2)

Definition at line 141 of file dbcompare_expression_scanner.cc.

11.44.1.54 #define YY_START_STACK_INCR 25

Definition at line 524 of file dbcompare_expression_scanner.cc.

11.44.1.55 #define YY_STATE_BUF_SIZE ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))

Definition at line 159 of file dbcompare_expression_scanner.cc.

11.44.1.56 #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)

Definition at line 145 of file dbcompare_expression_scanner.cc.

11.44.1.57 #define yyconst

Definition at line 118 of file dbcompare_expression_scanner.cc.

11.44.1.58 #define yyFlexLexer CoconutDBCompareExpressionFlexLexer

Definition at line 23 of file dbcompare_expression_scanner.cc.

11.44.1.59 #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 1574 of file dbcompare_expression_scanner.cc.

11.44.1.60 #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 1574 of file dbcompare_expression_scanner.cc.

11.44.1.61 #define yymore() yymore_used_but_not_detected

Definition at line 425 of file dbcompare_expression_scanner.cc.

11.44.1.62 #define YYSTATE YY_START

Definition at line 142 of file dbcompare_expression_scanner.cc.

11.44.1.63 #define YYTABLES_NAME "yytables"

Definition at line 1636 of file dbcompare_expression_scanner.cc.

11.44.1.64 #define yyterminate() return YY_NULL

Definition at line 519 of file dbcompare_expression_scanner.cc.

11.44.1.65 #define yytext_ptr yytext

Definition at line 314 of file dbcompare_expression_scanner.cc.

11.44.1.66 #define yywrap(n) 1

Definition at line 309 of file dbcompare_expression_scanner.cc.

11.44.2 Typedef Documentation

11.44.2.1 typedef short int flex_int16_t

Definition at line 56 of file dbcompare_expression_scanner.cc.

11.44.2.2 typedef int flex_int32_t

Definition at line 57 of file dbcompare_expression_scanner.cc.

11.44.2.3 typedef signed char flex_int8_t

Definition at line 55 of file dbcompare_expression_scanner.cc.

11.44.2.4 typedef unsigned short int flex_uint16_t

Definition at line 59 of file dbcompare_expression_scanner.cc.

11.44.2.5 typedef unsigned int flex_uint32_t

Definition at line 60 of file dbcompare_expression_scanner.cc.

11.44.2.6 typedef unsigned char flex_uint8_t

Definition at line 58 of file dbcompare_expression_scanner.cc.

11.44.2.7 typedef coco_api_internal::dbcompare_expression_parser::token token

Definition at line 460 of file dbcompare_expression_scanner.cc.

11.44.2.8 typedef struct yy_buffer_state* YY_BUFFER_STATE

Definition at line 163 of file dbcompare_expression_scanner.cc.

11.44.2.9 typedef unsigned char YY_CHAR

Definition at line 312 of file dbcompare_expression_scanner.cc.

11.44.2.10 typedef unsigned int yy_size_t

Definition at line 197 of file dbcompare_expression_scanner.cc.

11.44.3 Function Documentation

11.44.3.1 void * CoconutDBCompareExpressionalloc (yy_size_t *size*)

Definition at line 1614 of file dbcompare_expression_scanner.cc.

11.44.3.2 void CoconutDBCompareExpressionfree (void * *ptr*)

Definition at line 1631 of file dbcompare_expression_scanner.cc.

11.44.3.3 void * CoconutDBCompareExpressionrealloc (void * *ptr*, yy_size_t *size*)

Definition at line 1619 of file dbcompare_expression_scanner.cc.

11.44.3.4 if (!yy_init)

Definition at line 570 of file dbcompare_expression_scanner.cc.

11.44.3.5 while (1)

Definition at line 596 of file dbcompare_expression_scanner.cc.

11.44.4 Variable Documentation

11.44.4.1 register int yy_act

Definition at line 563 of file dbcompare_expression_scanner.cc.

11.44.4.2 register char * yy_bp

Definition at line 562 of file dbcompare_expression_scanner.cc.

11.44.4.3 register char* yy_cp

Definition at line 562 of file dbcompare_expression_scanner.cc.

11.44.4.4 YY_DECL register yy_state_type yy_current_state

The main scanner function which does all the work.

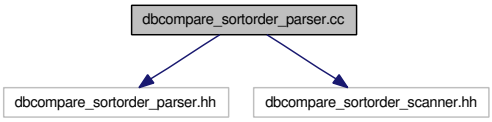
Definition at line 561 of file dbcompare_expression_scanner.cc.

11.44.4.5 int yyleng

11.45 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

11.45.1 Define Documentation

11.45.1.1 #define YY_(msgid) msgid

Definition at line 69 of file dbcompare_sortorder_parser.cc.

11.45.1.2 #define YY_REDUCE_PRINT(Rule)

Definition at line 109 of file dbcompare_sortorder_parser.cc.

11.45.1.3 #define YY_STACK_PRINT()

Definition at line 110 of file dbcompare_sortorder_parser.cc.

11.45.1.4 #define YY_SYMBOL_PRINT(Title, Type, Value, Location)

Definition at line 108 of file dbcompare_sortorder_parser.cc.

11.45.1.5 #define YYABORT goto yyabortlab

Definition at line 115 of file dbcompare_sortorder_parser.cc.

11.45.1.6 #define YYACCEPT goto yyacceptlab

Definition at line 114 of file dbcompare_sortorder_parser.cc.

11.45.1.7 #define YYCDEBUG

Value:

```
for (bool yydebugcond_ = yydebug_; yydebugcond_; yydebugcond_ = false)  \
    (*yycdebug_)
```

Definition at line 77 of file dbcompare_sortorder_parser.cc.

11.45.1.8 #define YYERROR goto yyerrorlab

Definition at line 116 of file dbcompare_sortorder_parser.cc.

11.45.1.9 #define yylex scanner.yylex

Definition at line 45 of file dbcompare_sortorder_parser.cc.

11.45.1.10 #define yylex coco_api_internallex

Definition at line 45 of file dbcompare_sortorder_parser.cc.

11.45.1.11 #define YYUSE(e) ((void) (e))

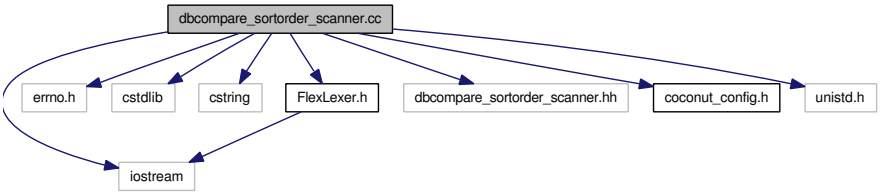
Definition at line 74 of file dbcompare_sortorder_parser.cc.

11.46 dbcompare_sortorder_scanner.cc File Reference

```
#include <iostream>
#include <errno.h>
#include <cstdlib>
#include <cstring>
#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 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 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_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_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

11.46.1 Define Documentation

11.46.1.1 #define BEGIN (yy_start) = 1 + 2 *

Definition at line 124 of file dbcompare_sortorder_scanner.cc.

11.46.1.2 #define ECHO LexerOutput(yytext, yyleng)

Definition at line 497 of file dbcompare_sortorder_scanner.cc.

11.46.1.3 #define EOB_ACT_CONTINUE_SCAN 0

Definition at line 153 of file dbcompare_sortorder_scanner.cc.

11.46.1.4 #define EOB_ACT_END_OF_FILE 1

Definition at line 154 of file dbcompare_sortorder_scanner.cc.

11.46.1.5 #define EOB_ACT_LAST_MATCH 2

Definition at line 155 of file dbcompare_sortorder_scanner.cc.

11.46.1.6 #define FLEX_BETA

Definition at line 14 of file dbcompare_sortorder_scanner.cc.

11.46.1.7 #define FLEX_SCANNER

Definition at line 9 of file dbcompare_sortorder_scanner.cc.

11.46.1.8 #define INITIAL 0

Definition at line 464 of file dbcompare_sortorder_scanner.cc.

11.46.1.9 #define INT16_MAX (32767)

Definition at line 66 of file dbcompare_sortorder_scanner.cc.

11.46.1.10 #define INT16_MIN (-32767-1)

Definition at line 57 of file dbcompare_sortorder_scanner.cc.

11.46.1.11 #define INT32_MAX (2147483647)

Definition at line 69 of file dbcompare_sortorder_scanner.cc.

11.46.1.12 #define INT32_MIN (-2147483647-1)

Definition at line 60 of file dbcompare_sortorder_scanner.cc.

11.46.1.13 #define INT8_MAX (127)

Definition at line 63 of file dbcompare_sortorder_scanner.cc.

11.46.1.14 #define INT8_MIN (-128)

Definition at line 54 of file dbcompare_sortorder_scanner.cc.

11.46.1.15 #define REJECT reject_used_but_not_detected

Definition at line 424 of file dbcompare_sortorder_scanner.cc.

11.46.1.16 #define UINT16_MAX (65535U)

Definition at line 75 of file dbcompare_sortorder_scanner.cc.

11.46.1.17 #define UINT32_MAX (4294967295U)

Definition at line 78 of file dbcompare_sortorder_scanner.cc.

11.46.1.18 #define UINT8_MAX (255U)

Definition at line 72 of file dbcompare_sortorder_scanner.cc.

11.46.1.19 #define unput(c) yyunput(c, (yytext_ptr))

Definition at line 173 of file dbcompare_sortorder_scanner.cc.

11.46.1.20 #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE → yy_at_bol)

Definition at line 292 of file dbcompare_sortorder_scanner.cc.

11.46.1.21 #define YY_BREAK break;

Definition at line 548 of file dbcompare_sortorder_scanner.cc.

11.46.1.22 #define YY_BUF_SIZE 16384

Definition at line 143 of file dbcompare_sortorder_scanner.cc.

11.46.1.23 #define YY_BUFFER_EOF_PENDING 2

Definition at line 246 of file dbcompare_sortorder_scanner.cc.

11.46.1.24 #define YY_BUFFER_NEW 0

Definition at line 234 of file dbcompare_sortorder_scanner.cc.

11.46.1.25 #define YY_BUFFER_NORMAL 1

Definition at line 235 of file dbcompare_sortorder_scanner.cc.

11.46.1.26 #define YY_CURRENT_BUFFER

Value:

```
( (yy_buffer_stack) \
                          ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
                          : NULL)
```

Definition at line 257 of file dbcompare_sortorder_scanner.cc.

11.46.1.27 #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]

Definition at line 264 of file dbcompare_sortorder_scanner.cc.

11.46.1.28 #define YY_DECL int dbcompare_sortorder_scanner::yylex()

Definition at line 308 of file dbcompare_sortorder_scanner.cc.

11.46.1.29 #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 313 of file dbcompare_sortorder_scanner.cc.

11.46.1.30 #define YY_END_OF_BUFFER 12

Definition at line 321 of file dbcompare_sortorder_scanner.cc.

11.46.1.31 #define YY_END_OF_BUFFER_CHAR 0

Definition at line 139 of file dbcompare_sortorder_scanner.cc.

11.46.1.32 #define YY_EXIT_FAILURE 2

11.46.1.33 #define YY_EXTRA_TYPE void *

Definition at line 475 of file dbcompare_sortorder_scanner.cc.

11.46.1.34 #define YY_FATAL_ERROR(msg) LexerError(msg)

Definition at line 526 of file dbcompare_sortorder_scanner.cc.

11.46.1.35 #define YY_FLEX_MAJOR_VERSION 2

Definition at line 10 of file dbcompare_sortorder_scanner.cc.

11.46.1.36 #define YY_FLEX_MINOR_VERSION 5

Definition at line 11 of file dbcompare_sortorder_scanner.cc.

11.46.1.37 #define YY_FLEX_SUBMINOR_VERSION 31

Definition at line 12 of file dbcompare_sortorder_scanner.cc.

11.46.1.38 #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 504 of file dbcompare_sortorder_scanner.cc.

11.46.1.39 #define YY_INT_ALIGNED short int

Definition at line 5 of file dbcompare_sortorder_scanner.cc.

11.46.1.40 #define YY_LESS_LINENO(n)

Definition at line 157 of file dbcompare_sortorder_scanner.cc.

11.46.1.41 #define YY_MORE_ADJ 0

Definition at line 426 of file dbcompare_sortorder_scanner.cc.

11.46.1.42 #define yy_new_buffer yy_create_buffer

Definition at line 270 of file dbcompare_sortorder_scanner.cc.

11.46.1.43 #define YY_NEW_FILE yyrestart(yyin)

Definition at line 137 of file dbcompare_sortorder_scanner.cc.

11.46.1.44 #define YY_NULL 0

Definition at line 111 of file dbcompare_sortorder_scanner.cc.

11.46.1.45 #define YY_NUM_RULES 11

Definition at line 320 of file dbcompare_sortorder_scanner.cc.

11.46.1.46 #define YY_READ_BUF_SIZE 8192

Definition at line 492 of file dbcompare_sortorder_scanner.cc.

11.46.1.47 #define YY_RESTORE_YY_MORE_OFFSET

Definition at line 427 of file dbcompare_sortorder_scanner.cc.

11.46.1.48 #define YY_RULE_SETUP YY_USER_ACTION

Definition at line 551 of file dbcompare_sortorder_scanner.cc.

11.46.1.49 #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)

Definition at line 118 of file dbcompare_sortorder_scanner.cc.

11.46.1.50 #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 282 of file dbcompare_sortorder_scanner.cc.

11.46.1.51 #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 272 of file dbcompare_sortorder_scanner.cc.

11.46.1.52 #define YY_SKIP_YYWRAP

Definition at line 295 of file dbcompare_sortorder_scanner.cc.

11.46.1.53 #define YY_START (((yy_start) - 1) / 2)

Definition at line 130 of file dbcompare_sortorder_scanner.cc.

11.46.1.54 #define YY_START_STACK_INCR 25

Definition at line 521 of file dbcompare_sortorder_scanner.cc.

11.46.1.55 #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)

Definition at line 134 of file dbcompare_sortorder_scanner.cc.

11.46.1.56 #define yyconst

Definition at line 107 of file dbcompare_sortorder_scanner.cc.

11.46.1.57 #define yyFlexLexer CoconutDBCompareSortOrderFlexLexer

Definition at line 20 of file dbcompare_sortorder_scanner.cc.

11.46.1.58 #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 160 of file dbcompare_sortorder_scanner.cc.

11.46.1.59 #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 160 of file dbcompare_sortorder_scanner.cc.

11.46.1.60 #define yymore() yymore_used_but_not_detected

Definition at line 425 of file dbcompare_sortorder_scanner.cc.

11.46.1.61 #define YYSTATE YY_START

Definition at line 131 of file dbcompare_sortorder_scanner.cc.

11.46.1.62 #define YYTABLES_NAME "yytables"

11.46.1.63 #define yyterminate() return YY_NULL

Definition at line 516 of file dbcompare_sortorder_scanner.cc.

11.46.1.64 #define yytext_ptr yytext

Definition at line 299 of file dbcompare_sortorder_scanner.cc.

11.46.1.65 #define yywrap() 1

Definition at line 294 of file dbcompare_sortorder_scanner.cc.

11.46.2 Typedef Documentation

11.46.2.1 typedef short int flex_int16_t

Definition at line 45 of file dbcompare_sortorder_scanner.cc.

11.46.2.2 typedef int flex_int32_t

Definition at line 46 of file dbcompare_sortorder_scanner.cc.

11.46.2.3 typedef signed char flex_int8_t

Definition at line 44 of file dbcompare_sortorder_scanner.cc.

11.46.2.4 typedef unsigned short int flex_uint16_t

Definition at line 48 of file dbcompare_sortorder_scanner.cc.

11.46.2.5 typedef unsigned int flex_uint32_t

Definition at line 49 of file dbcompare_sortorder_scanner.cc.

11.46.2.6 typedef unsigned char flex_uint8_t

Definition at line 47 of file dbcompare_sortorder_scanner.cc.

11.46.2.7 typedef coco_api_internal::dbcompare_sortorder_parser::token token

Definition at line 457 of file dbcompare_sortorder_scanner.cc.

11.46.2.8 typedef struct yy_buffer_state* YY_BUFFER_STATE

Definition at line 148 of file dbcompare_sortorder_scanner.cc.

11.46.2.9 typedef unsigned char YY_CHAR

Definition at line 297 of file dbcompare_sortorder_scanner.cc.

11.46.2.10 typedef unsigned int yy_size_t

Definition at line 182 of file dbcompare_sortorder_scanner.cc.

11.46.3 Function Documentation

11.46.3.1 void* CoconutDBCompareSortOrderalloc (yy_size_t)

11.46.3.2 void CoconutDBCompareSortOrderfree (void *)

11.46.3.3 void* CoconutDBCompareSortOrderrealloc (void *, yy_size_t)

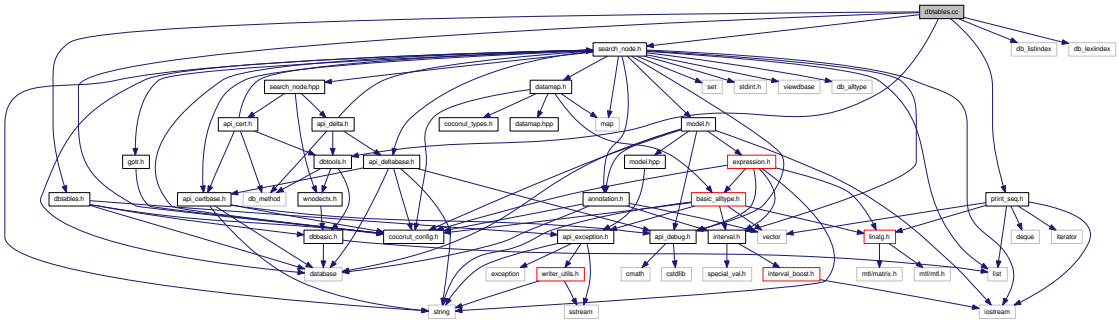
11.46.4 Variable Documentation

11.46.4.1 int yyleng

11.47 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)

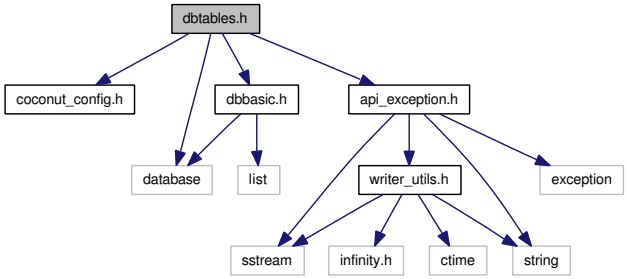
11.47.1 Detailed Description

Definition in file dbtables.cc.

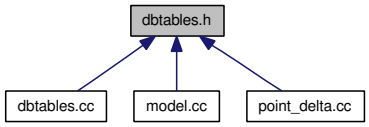
11.48 dbtables.h File Reference

```
#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.

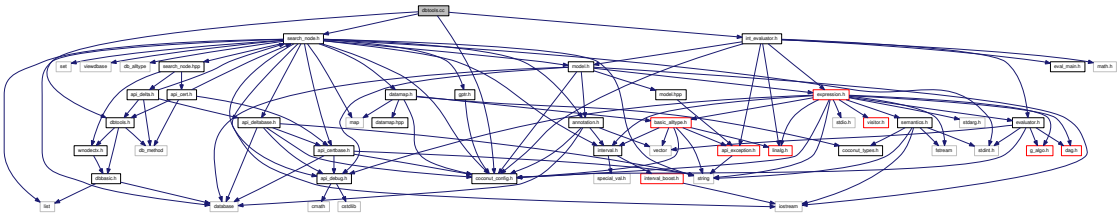
11.48.1 Detailed Description

Definition in file dbtables.h.

11.49 dbtools.cc File Reference

```
#include <coconut_config.h>
#include <search_node.h>
#include <dbtools.h>
#include <int_evaluator.h>
```

Include dependency graph for dbtools.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

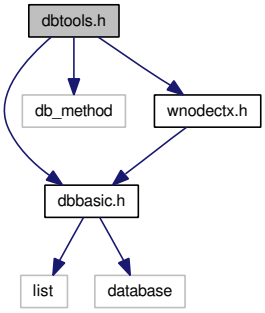
11.49.1 Detailed Description

Definition in file dbtools.cc.

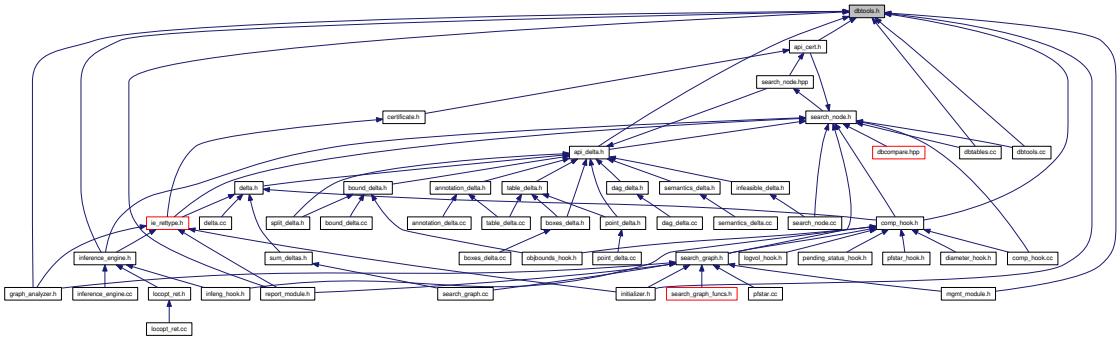
11.50 dbtools.h File Reference

```
#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

11.50.1 Detailed Description

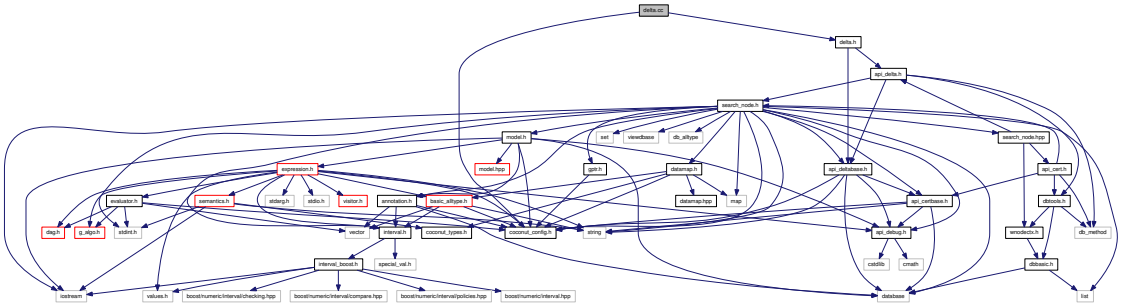
Definition in file dbtools.h.

11.51 delta.cc File Reference

```
#include <coconut_config.h>
```

```
#include <delta.h>
```

Include dependency graph for delta.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

11.51.1 Detailed Description

This is an internal header file, which is not intended for use outside the API.

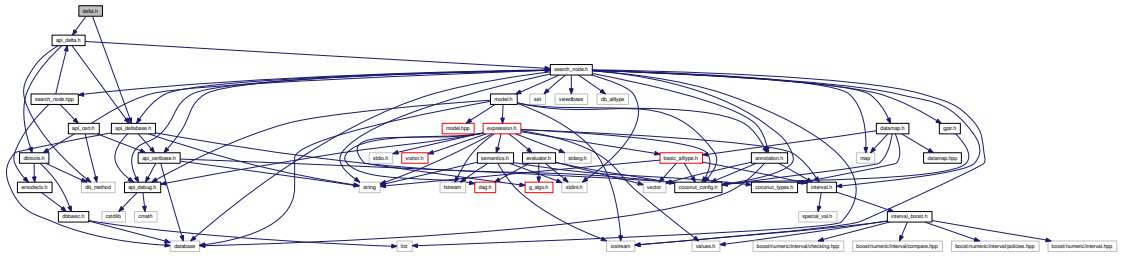
Definition in file delta.cc.

11.52 delta.h File Reference

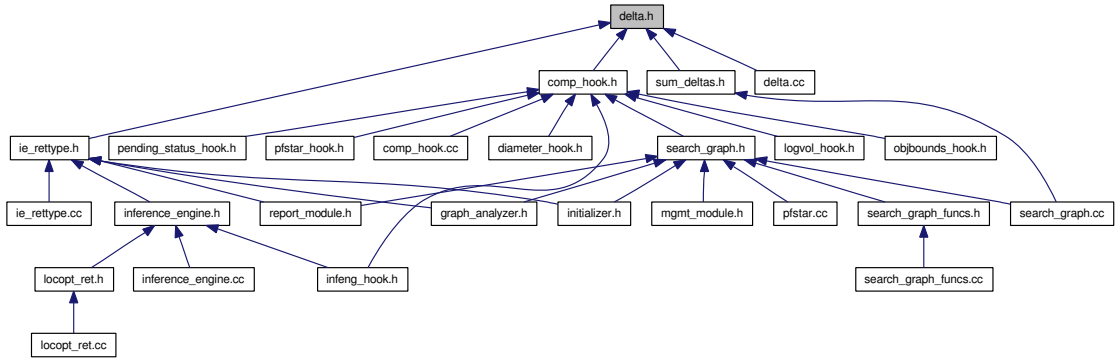
```
#include <api_deltabase.h>
```

```
#include <api_delta.h>
```

Include dependency graph for delta.h:



This graph shows which files directly or indirectly include this file:



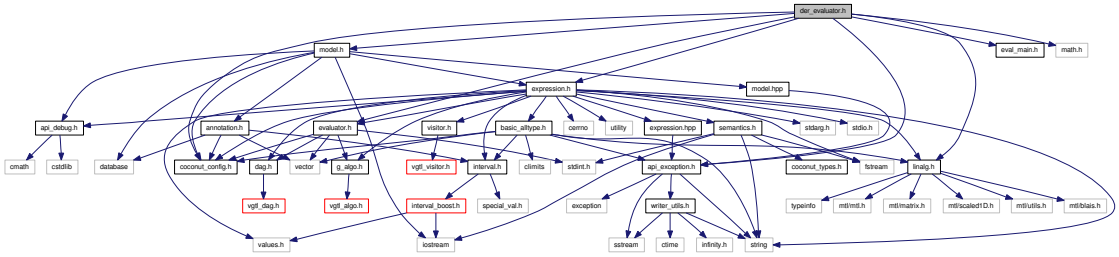
11.52.1 Detailed Description

Definition in file delta.h.

11.53 der_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <model.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.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)

11.53.1 Detailed Description

Definition in file der_evaluator.h.

11.54 der_evaluator_n.h File Reference

```
#include <coconut_config.h>

#include <evaluator.h>

#include <expression.h>

#include <model.h>

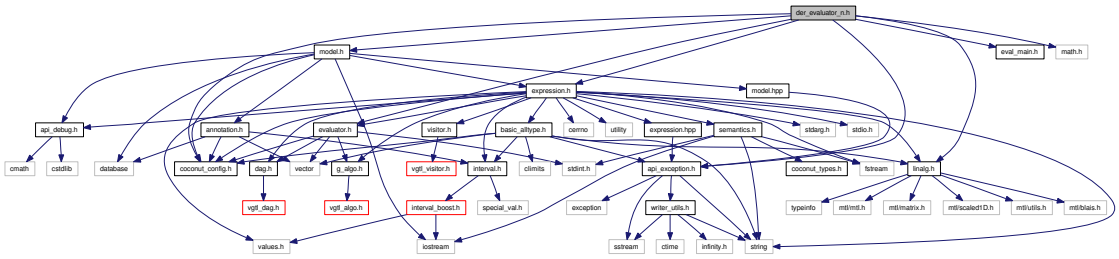
#include <eval_main.h>

#include <linalg.h>

#include <math.h>

#include <api_exception.h>
```

Include dependency graph for der_evaluator_n.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_cache

  Cache 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

Variables

- struct coco::der_cache ∗ coco::f_data

  Cache data for der_eval.

- std::vector< der_cache > ∗ coco::d_data
- std::vector< double > ∗ coco::grad_vec
- const model ∗ coco::mod
- double coco::mult
- double coco::mult_trans
- unsigned int coco::child_n

11.55 dfunc_evaluator.h File Reference

```
#include <coconut_config.h>

#include <evaluator.h>

#include <expression.h>

#include <model.h>

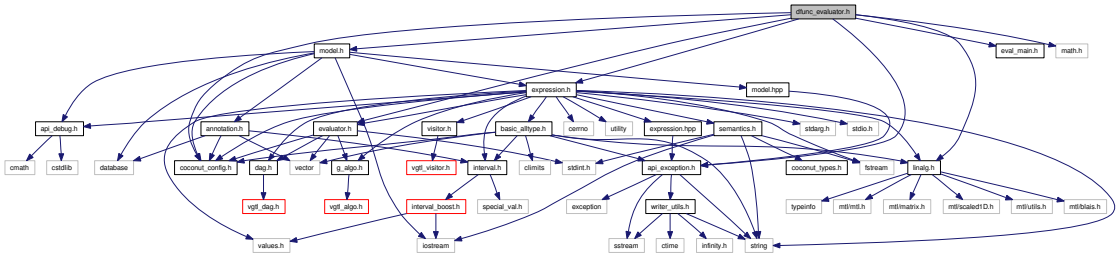
#include <eval_main.h>

#include <linalg.h>

#include <math.h>

#include <api_exception.h>
```

Include dependency graph for dfunc_evaluator.h:



Classes

- struct coco::dfunc_eval_rettype< _T, DN >
- struct coco::dfunc_eval_type< _T, DN >
- class coco::dfunc_eval< _T, DN >

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)

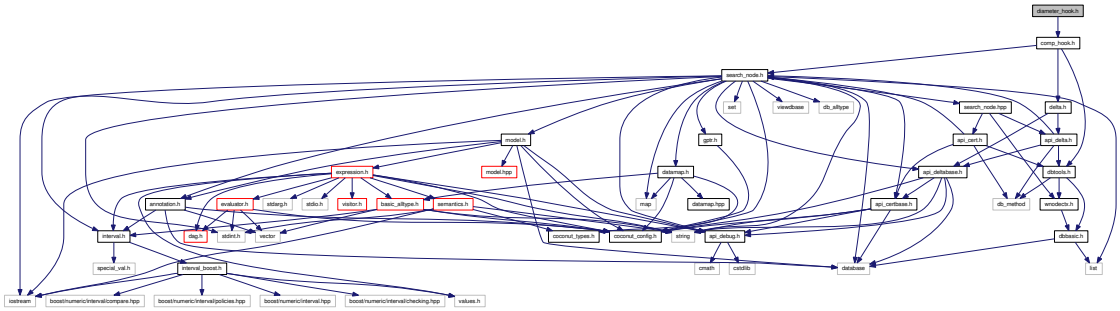
11.55.1 Detailed Description

Definition in file dfunc_evaluator.h.

11.56 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

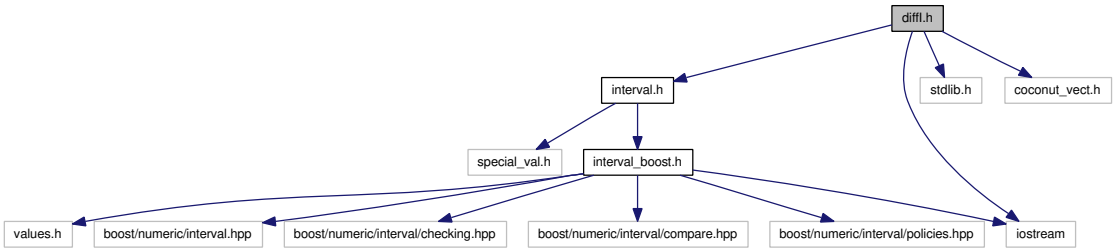
11.56.1 Detailed Description

Definition in file diameter_hook.h.

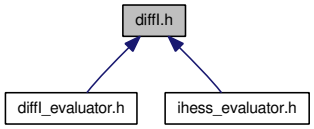
11.57 diffI.h File Reference

```
#include "interval.h"
#include <stdlib.h>
#include <iostream>
#include <coconut_vect.h>
```

Include dependency graph for diffI.h:



This graph shows which files directly or indirectly include this file:



Classes

- class diffI

Functions

- 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 diffI &a, const diffI &b)
- diffI operator- (const double &a, const diffI &b)
- diffI operator- (const diffI &a, const double &b)
- diffI operator- (const diffI &a)
- 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 diffI &a, const diffI &b)
- diffI operator/ (const diffI &a, const double &b)
- std::ostream & operator<< (std::ostream &s, const diffI &a)
- diffI exp (const diffI &a)
- diffI sin (const diffI &a)
- diffI cos (const diffI &a)
- diffI sqrt (const diffI &a)
- diffI power (const diffI &a, int n)
- diffI power (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 square (const diffI &a)
- diffI operator/ (const double &a, const diffI &b)

Variables

- const int STANDARD_MAXDEG_DIFFINTERVAL = 4

11.57.1 Function Documentation

11.57.1.1 diffI atan (const diffI & *a*) [inline]

Definition at line 567 of file diffI.h.

11.57.1.2 diffI atan2 (const diffI & *a*, const diffI & *b*) [inline]

Definition at line 550 of file diffI.h.

11.57.1.3 diffI cos (const diffI & *a*) [inline]

Definition at line 416 of file diffI.h.

11.57.1.4 diffI cosh (const diffI & *a*) [inline]

Definition at line 535 of file diffI.h.

11.57.1.5 diffI exp (const diffI & *a*) [inline]

Definition at line 384 of file diffI.h.

11.57.1.6 diffI log (const diffI & *a*) [inline]

Definition at line 507 of file diffI.h.

11.57.1.7 diffI operator∗ (const diffI & *a*, const double & *b*) [inline]

Definition at line 333 of file diffI.h.

11.57.1.8 diffI operator∗ (const double & *a*, const diffI & *b*) [inline]

Definition at line 324 of file diffI.h.

11.57.1.9 diffI operator∗ (const diffI & *a*, const diffI & *b*) [inline]

Definition at line 309 of file diffI.h.

11.57.1.10 diffI operator+ (const diffI & *a*, const double & *b*) [inline]

Definition at line 263 of file diffI.h.

11.57.1.11 diffI operator+ (const double & *a*, const diffI & *b*) [inline]

Definition at line 255 of file diffI.h.

11.57.1.12 diffI operator+ (const diffI & *a*, const diffI & *b*) [inline]

Definition at line 242 of file diffI.h.

11.57.1.13 diffI operator- (const diffI & *a*) [inline]

Definition at line 300 of file diffI.h.

11.57.1.14 diffI operator- (const diffI & *a*, const double & *b*) [inline]

Definition at line 292 of file diffI.h.

11.57.1.15 diffI operator- (const double & *a*, const diffI & *b*) [inline]

Definition at line 284 of file diffI.h.

11.57.1.16 diffI operator- (const diffI & *a*, const diffI & *b*) [inline]

Definition at line 271 of file diffI.h.

11.57.1.17 diffI operator/ (const double & *a*, const diffI & *b*) [inline]

Definition at line 360 of file diffI.h.

11.57.1.18 diffI operator/ (const diffI & *a*, const double & *b*) [inline]

Definition at line 375 of file diffI.h.

11.57.1.19 diffI operator/ (const diffI & *a*, const diffI & *b*) [inline]

Definition at line 342 of file diffI.h.

11.57.1.20 std::ostream & operator<< (std::ostream & *s*, const diffI & *a*) [inline]

Definition at line 656 of file diffI.h.

11.57.1.21 diffI power (const diffI & *a*, const diffI & *b*) [inline]

Definition at line 491 of file diffI.h.

11.57.1.22 diffI power (const diffI & *a*, int *n*) [inline]

Definition at line 460 of file diffI.h.

11.57.1.23 diffI sin (const diffI & *a*) [inline]

Definition at line 396 of file diffI.h.

11.57.1.24 diffI sinh (const diffI & *a*) [inline]

Definition at line 520 of file diffI.h.

11.57.1.25 diffI sqrt (const diffI & *a*) [inline]

Definition at line 437 of file diffI.h.

11.57.1.26 diffI square (const diffI & *a*) [inline]

Definition at line 582 of file diffI.h.

11.57.2 Variable Documentation

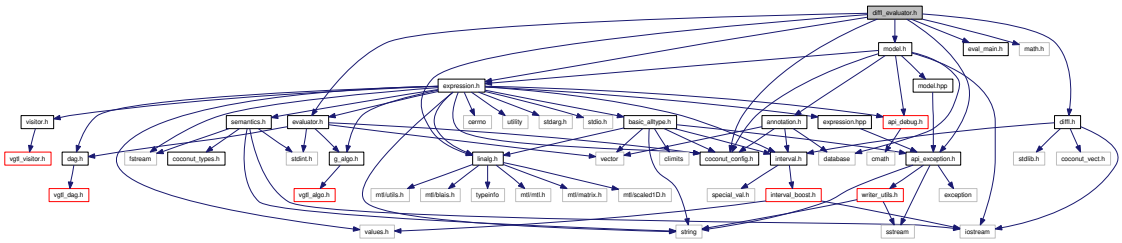
11.57.2.1 const int STANDARD_MAXDEG_DIFFINTERVAL = 4

Definition at line 11 of file diffI.h.

11.58 diffI_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <model.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <api_exception.h>
#include <diffI.h>
```

Include dependency graph for diffI_evaluator.h:



Classes

- struct coco::diffI_eval_type
  Visitor data for diffI_eval.
- class coco::diffI_eval
  Forward function range evaluation.

Namespaces

- namespace coco
  the main namespace of the COCONUT API

Typedefs

- typedef diffI(∗ coco::diffI_evaluator)(const std::vector< diffI > ∗__x, const variable_indicator &_-_v)

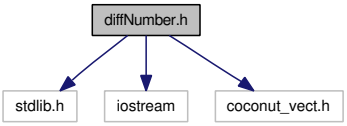
11.58.1 Detailed Description

Definition in file diffI_evaluator.h.

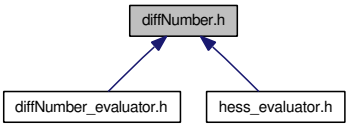
11.59 diffNumber.h File Reference

```
#include <stdlib.h>
#include <iostream>
#include <coconut_vect.h>
```

Include dependency graph for diffNumber.h:



This graph shows which files directly or indirectly include this file:



Classes

- class diffNumber

Functions

- 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 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 diffNumber &a, const double &b)
- diffNumber operator/ (const double &a, const diffNumber &b)
- std::ostream & operator<< (std::ostream &s, const diffNumber &a)
- diffNumber exp (const diffNumber &a)
- diffNumber sin (const diffNumber &a)
- diffNumber cos (const diffNumber &a)
- diffNumber sqrt (const diffNumber &a)
- diffNumber power (const diffNumber &a, int n)
- diffNumber power (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)

Variables

- const int STANDARD_MAXDEG_DIFFNUMBER = 4

11.59.1 Function Documentation

11.59.1.1 diffNumber atan (const diffNumber & *a*) [inline]

Definition at line 628 of file diffNumber.h.

11.59.1.2 diffNumber atan2 (const diffNumber & *a*, const diffNumber & *b*) [inline]

Definition at line 611 of file diffNumber.h.

11.59.1.3 diffNumber cos (const diffNumber & *a*) [inline]

Definition at line 367 of file diffNumber.h.

11.59.1.4 diffNumber cosh (const diffNumber & *a*) [inline]

Definition at line 596 of file diffNumber.h.

11.59.1.5 diffNumber exp (const diffNumber & *a*) [inline]

Definition at line 340 of file diffNumber.h.

11.59.1.6 diffNumber log (const diffNumber & *a*) [inline]

Definition at line 568 of file diffNumber.h.

11.59.1.7 diffNumber operator∗ (const diffNumber & *a*, const double & *b*) [inline]

Definition at line 290 of file diffNumber.h.

11.59.1.8 diffNumber operator∗ (const double & *a*, const diffNumber & *b*) [inline]

Definition at line 281 of file diffNumber.h.

11.59.1.9 diffNumber operator∗ (const diffNumber & *a*, const diffNumber & *b*) [inline]

Definition at line 265 of file diffNumber.h.

11.59.1.10 diffNumber operator+ (const diffNumber & *a*, const double & *b*) [inline]

Definition at line 214 of file diffNumber.h.

11.59.1.11 diffNumber operator+ (const double & *a*, const diffNumber & *b*) [inline]

Definition at line 204 of file diffNumber.h.

11.59.1.12 diffNumber operator+ (const diffNumber & *a*, const diffNumber & *b*) [inline]

Definition at line 191 of file diffNumber.h.

11.59.1.13 diffNumber operator- (const diffNumber & *a*) [inline]

Definition at line 257 of file diffNumber.h.

11.59.1.14 diffNumber operator- (const diffNumber & *a*, const double & *b*) [inline]

Definition at line 247 of file diffNumber.h.

11.59.1.15 diffNumber operator- (const double & *a*, const diffNumber & *b*) [inline]

Definition at line 237 of file diffNumber.h.

11.59.1.16 diffNumber operator- (const diffNumber & *a*, const diffNumber & *b*) [inline]

Definition at line 224 of file diffNumber.h.

11.59.1.17 diffNumber operator/ (const double & *a*, const diffNumber & *b*) [inline]

Definition at line 317 of file diffNumber.h.

11.59.1.18 diffNumber operator/ (const diffNumber & *a*, const double & *b*) [inline]

Definition at line 331 of file diffNumber.h.

11.59.1.19 diffNumber operator/ (const diffNumber & *a*, const diffNumber & *b*) [inline]

Definition at line 299 of file diffNumber.h.

11.59.1.20 std::ostream & operator<< (std::ostream & *s*, const diffNumber & *a*) [inline]

Definition at line 701 of file diffNumber.h.

11.59.1.21 diffNumber power (const diffNumber & *a*, const diffNumber & *b*) [inline]

Definition at line 422 of file diffNumber.h.

11.59.1.22 diffNumber power (const diffNumber & *a*, int *n*) [inline]

Definition at line 401 of file diffNumber.h.

11.59.1.23 diffNumber sin (const diffNumber & *a*) [inline]

Definition at line 352 of file diffNumber.h.

11.59.1.24 diffNumber sinh (const diffNumber & *a*) [inline]

Definition at line 581 of file diffNumber.h.

11.59.1.25 diffNumber sqrt (const diffNumber & *a*) [inline]

Definition at line 382 of file diffNumber.h.

11.59.1.26 diffNumber square (const diffNumber & *a*) [inline]

Definition at line 644 of file diffNumber.h.

11.59.2 Variable Documentation

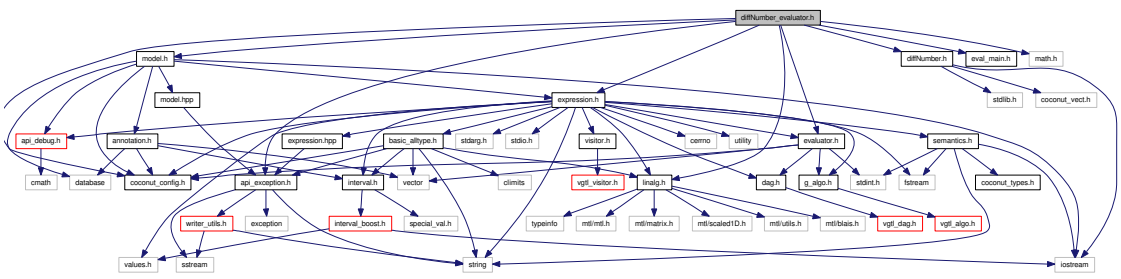
11.59.2.1 const int STANDARD_MAXDEG_DIFFNUMBER = 4

Definition at line 9 of file diffNumber.h.

11.60 diffNumber_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <model.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <api_exception.h>
#include <diffNumber.h>
```

Include dependency graph for diffNumber_evaluator.h:



Classes

- struct coco::diffNumber_eval_type
  Visitor data for diffNumber_eval.
- class coco::diffNumber_eval
  Forward function range evaluation.

Namespaces

- namespace coco
  the main namespace of the COCONUT API

Typedefs

- typedef diffNumber(∗ coco::diffNumber_evaluator)(const std::vector< diffNumber > ∗__x, const variable_indicator &__v)

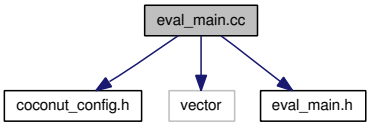
11.60.1 Detailed Description

Definition in file diffNumber_evaluator.h.

11.61 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)

11.61.1 Detailed Description

Definition in file eval_main.cc.

11.62 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

11.62.1 Detailed Description

Definition in file eval_main.h.

11.62.2 Define Documentation

11.62.2.1 #define CONVEX_EVALUATOR 17

function convexity evaluation

Definition at line 54 of file eval_main.h.

11.62.2.2 #define DER_EVALUATOR 4

derivative evaluation

Definition at line 39 of file eval_main.h.

11.62.2.3 #define DER_RANGE 5

derivative range evaluation

Definition at line 40 of file eval_main.h.

11.62.2.4 #define DIFF_EVALUATOR 18

function differentiability evaluation

Definition at line 55 of file eval_main.h.

11.62.2.5 #define FUNC_ANALYTICD 15

function analytic range

Definition at line 52 of file eval_main.h.

11.62.2.6 #define FUNC_BRANGE 14

function bounded range

Definition at line 51 of file eval_main.h.

11.62.2.7 #define FUNC_CRANGE 13

function complex range

Definition at line 50 of file eval_main.h.

11.62.2.8 #define FUNC_D_EVALUATOR 1

function evaluation with partial derivative

Definition at line 34 of file eval_main.h.

11.62.2.9 #define FUNC_D_RANGE 3

function range evaluation with partial derivative

Definition at line 37 of file eval_main.h.

11.62.2.10 #define FUNC_EVALUATOR 0

function evaluation

Definition at line 33 of file eval_main.h.

11.62.2.11 #define FUNC_FW_PREP 8

function evaluation

Definition at line 43 of file eval_main.h.

11.62.2.12 #define FUNC_ID_EVALUATOR 11

function range evaluation with interval derivative

Definition at line 47 of file eval_main.h.

11.62.2.13 #define FUNC_IDER_FORWARD 19

function range evaluation

Definition at line 57 of file eval_main.h.

11.62.2.14 #define FUNC_INFB 16

function bounds at infinity

Definition at line 53 of file eval_main.h.

11.62.2.15 #define FUNC_ISLP_EVALUATOR 9

function evaluation with partial slope

Definition at line 44 of file eval_main.h.

11.62.2.16 #define FUNC_RANGE 2

function range evaluation

Definition at line 36 of file eval_main.h.

11.62.2.17 #define IDER_EVALUATOR 12

interval derivative evaluation

Definition at line 49 of file eval_main.h.

11.62.2.18 #define ISLP_EVALUATOR 10

slope evaluation

Definition at line 46 of file eval_main.h.

11.62.2.19 #define NUM_EVALUATORS 20

number of evaluators for user defined nodes

Definition at line 59 of file eval_main.h.

11.62.2.20 #define PRINT 7

print function

Definition at line 42 of file eval_main.h.

11.62.2.21 #define REDUCE_RANGE 6

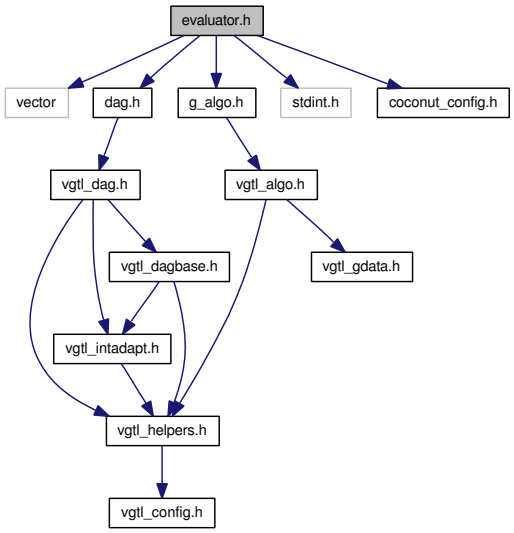
function backwards reduction

Definition at line 41 of file eval_main.h.

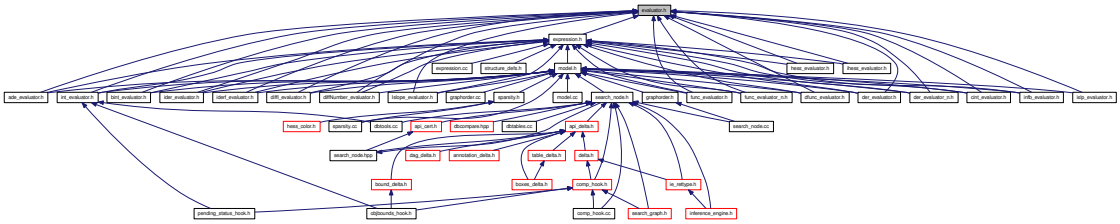
11.63 evaluator.h File Reference

```
#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< _Tp, _NData, _Result, _Walker >

  Base class of all evaluators.

- class coco::evaluator_base< _Tp, _NData, _Result, _Walker >

  Base class of all (non-caching) evaluators.

- class coco::cached_evaluator_base< _Tp, _NData, _Result, _Walker >

  Base class of all caching evaluators.

- class coco::forward_evaluator_base< _Tp, _NData, _Result, _Walker >

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

- class coco::backward_evaluator_base< _Tp, _NData, _Result, _Walker >

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

- class coco::cached_forward_evaluator_base< _Tp, _NData, _Result, _Walker >
  Base class of all (non-caching) forward evaluators.
- class coco::cached_backward_evaluator_base< _Tp, _NData, _Result, _Walker >
  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.

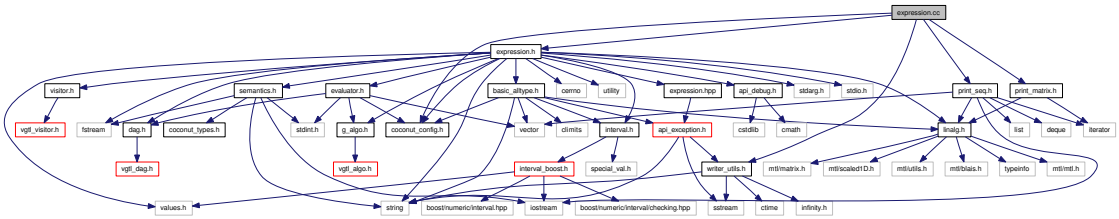
11.63.1 Detailed Description

Definition in file evaluator.h.

11.64 expression.cc File Reference

```
#include <coconut_config.h>
#include <expression.h>
#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)

Variables

- const char * coco::expr_names [ ]

11.64.1 Detailed Description

Definition in file expression.cc.

11.65 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>

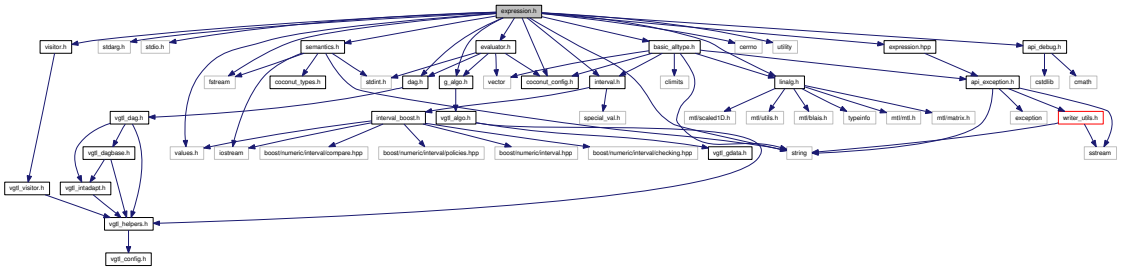
#include <values.h>

#include <linalg.h>

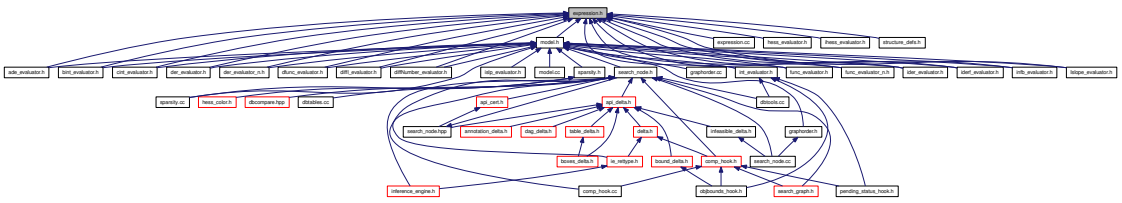
#include <api_debug.h>

#include <expression.hpp>
```

Include dependency graph for expression.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::expression_node

  The base class for a node in the expression DAGs.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #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_GAUSS -20
- #define EXPRINFO_POLY -21
- #define EXPRINFO_POW -22
- #define EXPRINFO_DIV -23
- #define EXPRINFO_ATAN2 -24
- #define EXPRINFO_LIN -25
- #define EXPRINFO_QUAD -26
- #define EXPRINFO_RE -27
- #define EXPRINFO_IM -28
- #define EXPRINFO_ARG -29
- #define EXPRINFO_CPLXCONJ -30
- #define EXPRINFO_LOOKUP -31
- #define EXPRINFO_PWLIN -32
- #define EXPRINFO_SPLINE -33
- #define EXPRINFO_PWCONSTLC -34
- #define EXPRINFO_PWCONSTRC -35
- #define EXPRINFO_IN -36
- #define EXPRINFO_IF -37
- #define EXPRINFO_AND -38
- #define EXPRINFO_OR -39
- #define EXPRINFO_NOT -40
- #define EXPRINFO_IMPLIES -41
- #define EXPRINFO_COUNT -42
- #define EXPRINFO_ALLDIFF -43
- #define EXPRINFO_HISTOGRAM -44
- #define EXPRINFO_LEVEL -45
- #define EXPRINFO_NEIGHBOR -46
- #define EXPRINFO_NOGOOD -47
- #define EXPRINFO_EXPECTATION -48
- #define EXPRINFO_INTEGRAL -49
- #define EXPRINFO_DET -50
- #define EXPRINFO_COND -51
- #define EXPRINFO_PSD -52
- #define EXPRINFO_MPROD -53

- #define EXPRINFO_FEM -54
- #define EXPRINFO_CMPROD -55
- #define EXPRINFO_CGFEM -56
- #define EXPRINFO_UNDEFINED -57
- #define EXPRINFO_NUMOFPREDEF -(EXPRINFO_UNDEFINED)

Typedefs

- typedef interval coco::rhs_t
- typedef std::vector< void * > coco::evaluator_v

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 }

11.65.1 Detailed Description

Definition in file expression.h.

11.65.2 Define Documentation

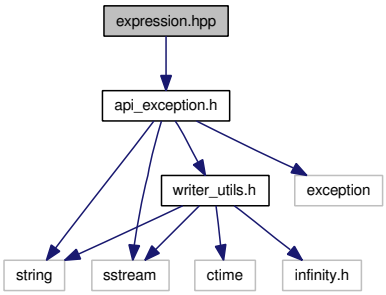
11.65.2.1 #define EXPR_LASTARG NULL

Definition at line 362 of file expression.h.

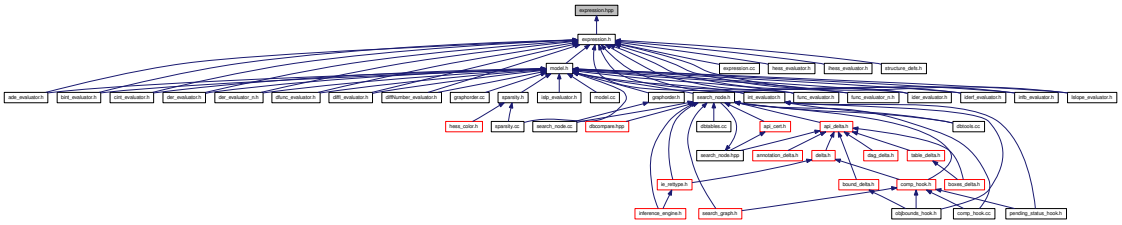
11.66 expression.hpp File Reference

```
#include <api_exception.h>
```

Include dependency graph for expression.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::expression_node::expression_node::parents_compare
- class coco::expression_node::expression_node::children_compare
- class coco::expression_node::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)

11.66.1 Detailed Description

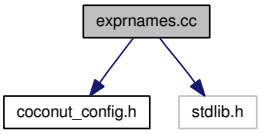
Definition in file expression.hpp.

11.67 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

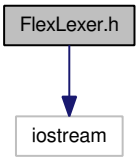
11.67.1 Detailed Description

Definition in file exprnames.cc.

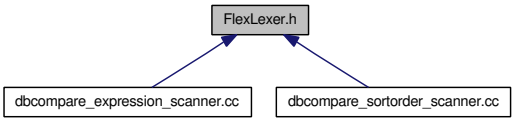
11.68 FlexLexer.h File Reference

```
#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

11.68.1 Define Documentation

11.68.1.1 #define FLEX_STD std::

Definition at line 53 of file FlexLexer.h.

11.68.1.2 #define yyFlexLexerOnce

Definition at line 108 of file FlexLexer.h.

11.68.2 Typedef Documentation

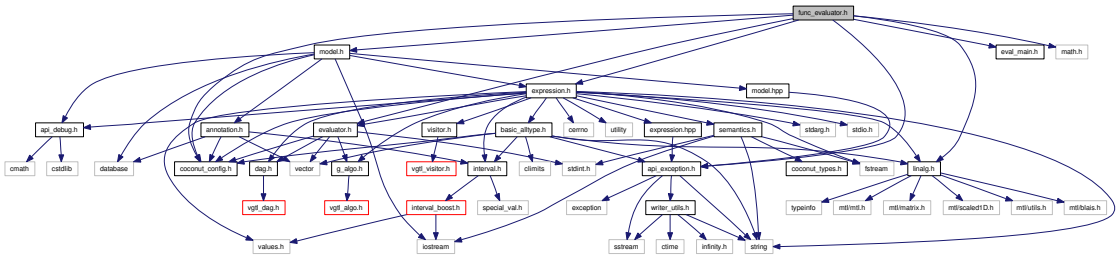
11.68.2.1 typedef int yy_state_type

Definition at line 58 of file FlexLexer.h.

11.69 func_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <model.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <api_exception.h>
```

Include dependency graph for func_evaluator.h:



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

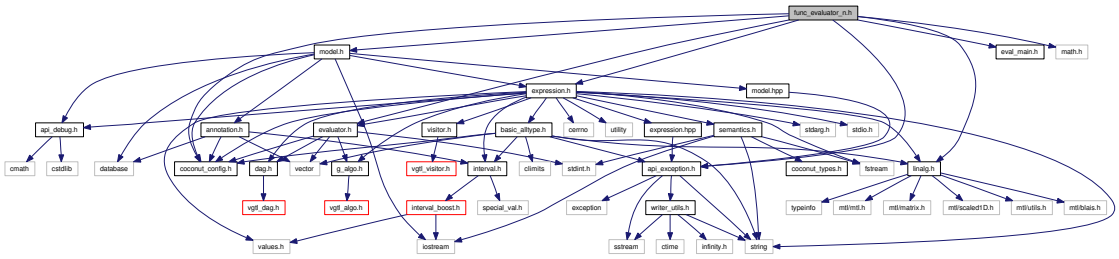
11.69.1 Detailed Description

Definition in file func_evaluator.h.

11.70 func_evaluator_n.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <model.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <api_exception.h>
```

Include dependency graph for func_evaluator_n.h:



Classes

- struct coco::func_cache

  Cache data for func_eval.

- struct coco::func_eval_type

  Visitor data for func_eval.

- class coco::func_eval

  Forward function evaluation.

Namespaces

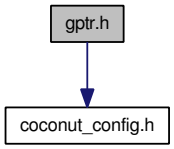
- namespace coco

  the main namespace of the COCONUT API

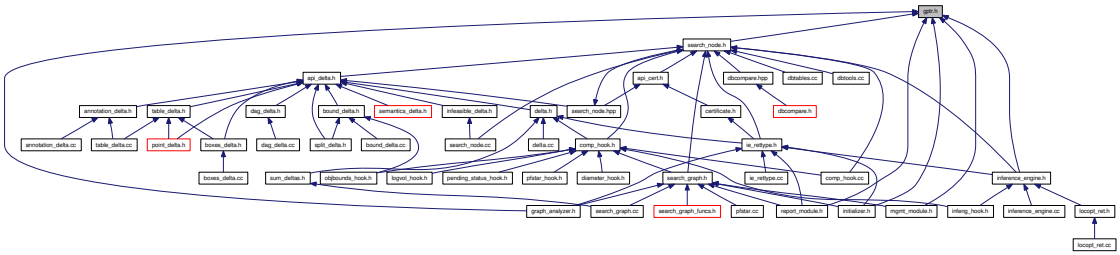
11.71 gptr.h File Reference

```
#include <coconut_config.h>
```

Include dependency graph for gptr.h:



This graph shows which files directly or indirectly include this file:



Classes

- class gptr< _Tp >
  Global pointer class.
- class ptr< _Tp >
  Local global pointer class.

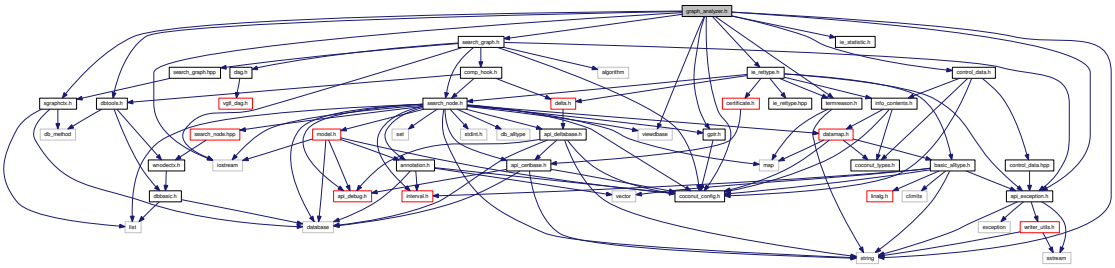
11.71.1 Detailed Description

Definition in file gptr.h.

11.72 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.

11.72.1 Detailed Description

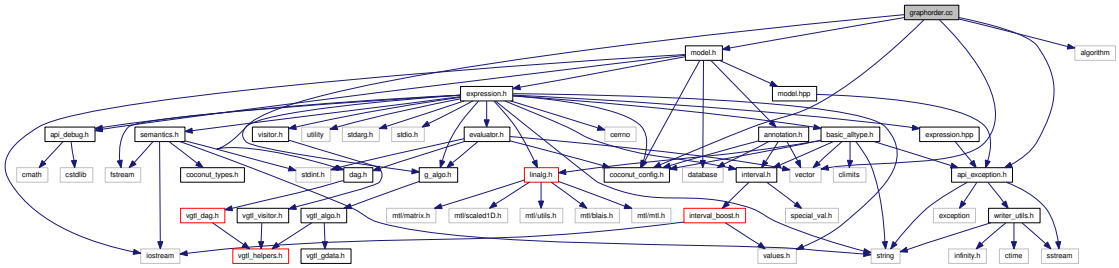
Definition in file graph_analyzer.h.

11.73 graphorder.cc File Reference

```
#include <coconut_config.h>
#include <model.h>
#include <vector>
#include <algorithm>
#include <g_algo.h>
```

```
#include <api_exception.h>
```

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

  Compute a graph order.

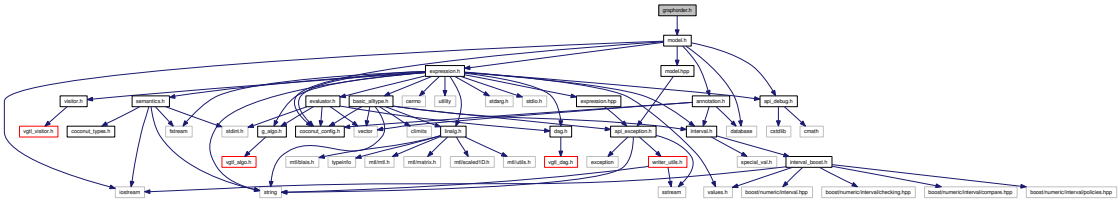
11.73.1 Detailed Description

Definition in file graphorder.cc.

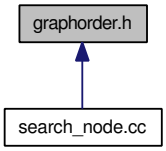
11.74 graphorder.h File Reference

```
#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.

11.74.1 Detailed Description

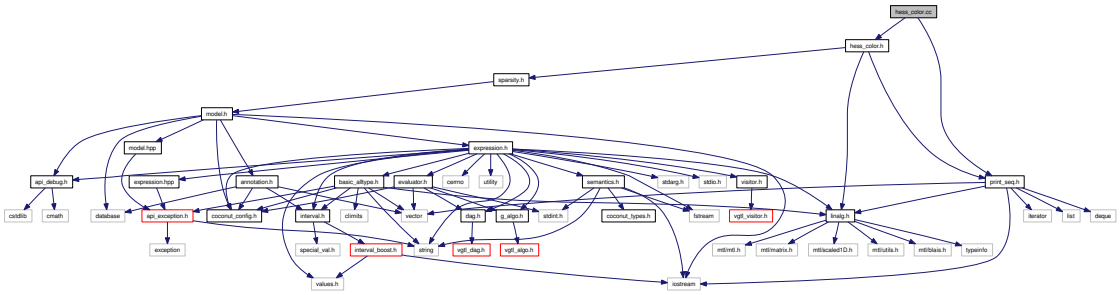
Definition in file graphorder.h.

11.75 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 linalg::matrix< int > &H, std::vector< std::vector< unsigned int > > &parts, std::vector< linalg::sparse_vector< unsigned int > > &provides)

11.75.1 Detailed Description

Definition in file hess_color.cc.

11.75.2 Define Documentation

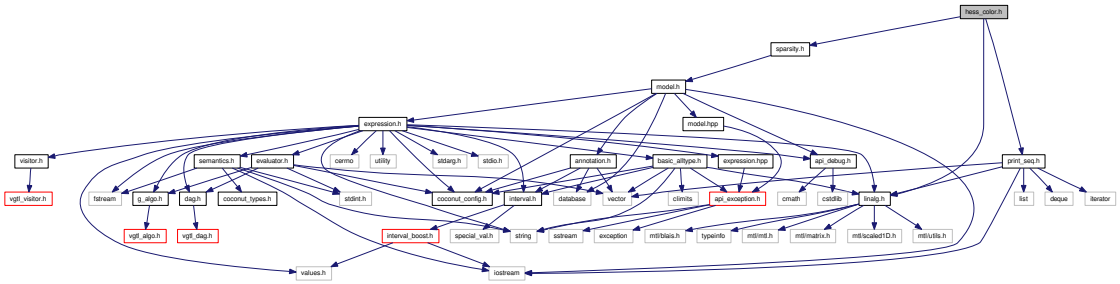
11.75.2.1 #define HESS_COLOR_DEBUG 0

Definition at line 33 of file hess_color.cc.

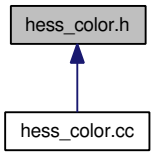
11.76 hess_color.h File Reference

```
#include <sparsity.h>
#include <linalg.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 linalg::matrix< int > &H, std::vector< std::vector< unsigned int > > &parts, std::vector< linalg::sparse_vector< unsigned int > > &provides)

11.76.1 Detailed Description

Definition in file hess_color.h.

11.76.2 Define Documentation

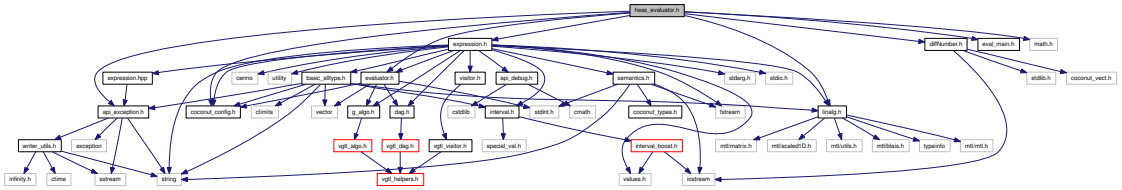
11.76.2.1 #define HESS_VEC_THRESHOLD 0.9

Definition at line 40 of file hess_color.h.

11.77 hess_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <api_exception.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

11.77.1 Detailed Description

Definition in file hess_evaluator.h.

11.77.2 Define Documentation

11.77.2.1 #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

Definition at line 31 of file hess_evaluator.h.

11.77.2.2 #define DEBUG_HESS_EVALUATOR 0

Definition at line 30 of file hess_evaluator.h.

11.78 ider_evaluator.h File Reference

```
#include <coconut_config.h>

#include <evaluator.h>

#include <expression.h>

#include <model.h>

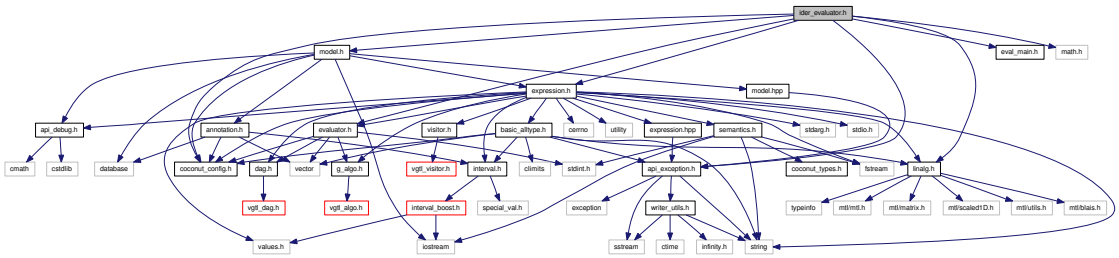
#include <eval_main.h>

#include <linalg.h>

#include <math.h>

#include <api_exception.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)

11.78.1 Detailed Description

Definition in file ider_evaluator.h.

11.79 iderf_evaluator.h File Reference

```
#include <coconut_config.h>

#include <evaluator.h>

#include <expression.h>

#include <model.h>

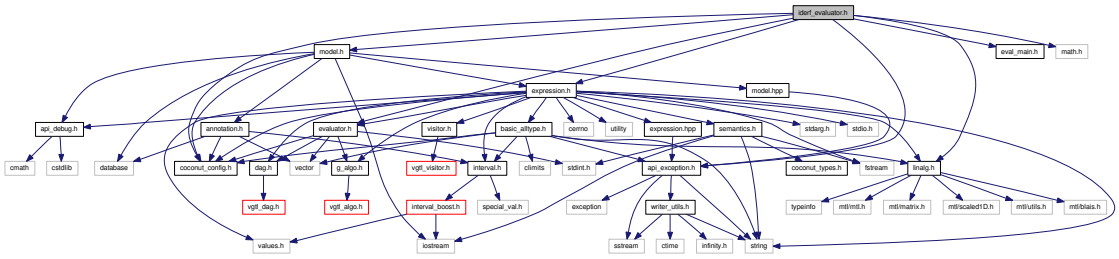
#include <eval_main.h>

#include <linalg.h>

#include <math.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)

11.79.1 Detailed Description

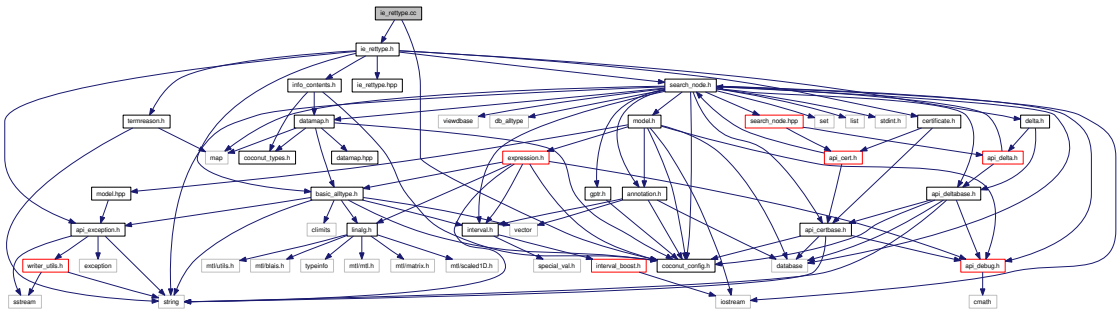
Definition in file iderf_evaluator.h.

11.80 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

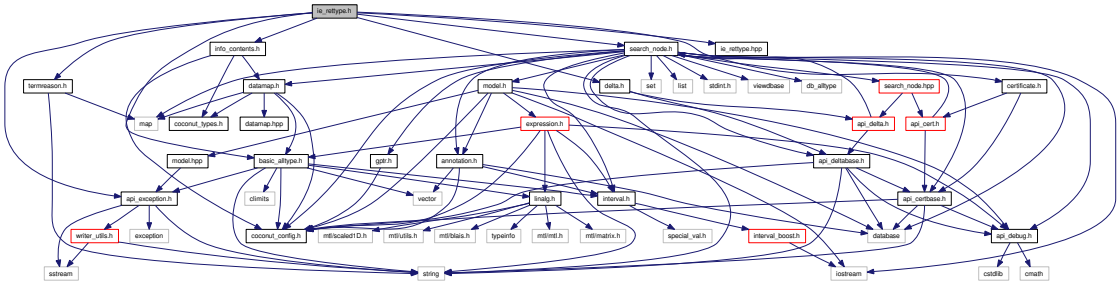
11.80.1 Detailed Description

Definition in file ie_rettype.cc.

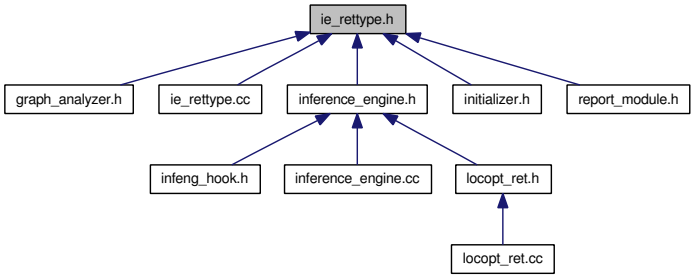
11.81 ie_rettype.h File Reference

```
#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 <ie_rettype.hpp>
```

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

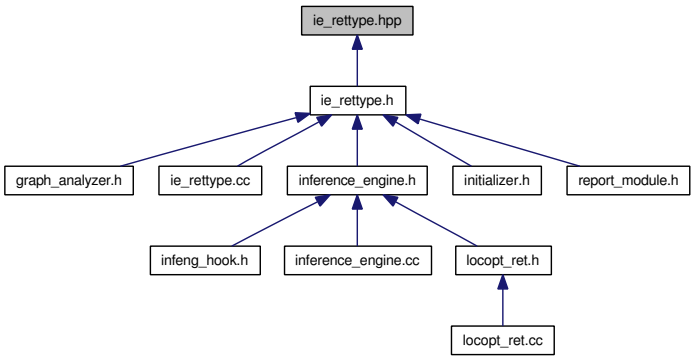
- namespace coco
  the main namespace of the COCONUT API

11.81.1 Detailed Description

Definition in file ie_rettype.h.

11.82 ie_rettype.hpp File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace coco

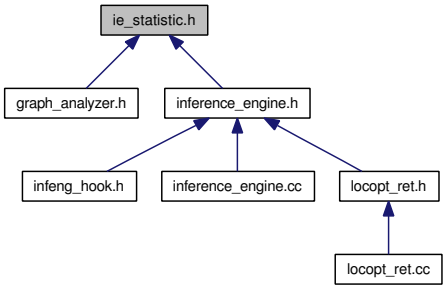
  the main namespace of the COCONUT API

11.82.1 Detailed Description

Definition in file ie_rettype.hpp.

11.83 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

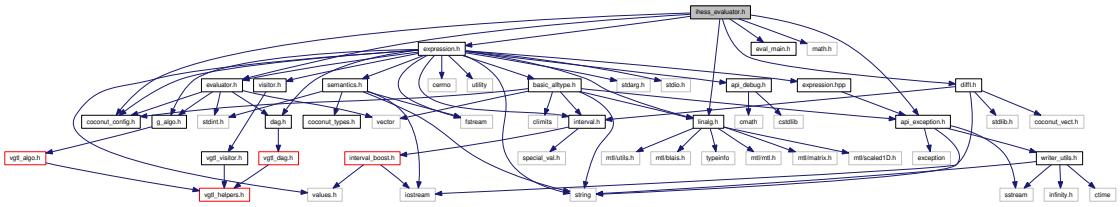
11.83.1 Detailed Description

Definition in file ie_statistic.h.

11.84 ihess_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <api_exception.h>
#include <diffI.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

11.84.1 Detailed Description

Definition in file ihess_evaluator.h.

11.84.2 Define Documentation

11.84.2.1 #define DEBUG_FHE_LIST_NODES_WITH_PARAMETERS 0

Definition at line 31 of file ihess_evaluator.h.

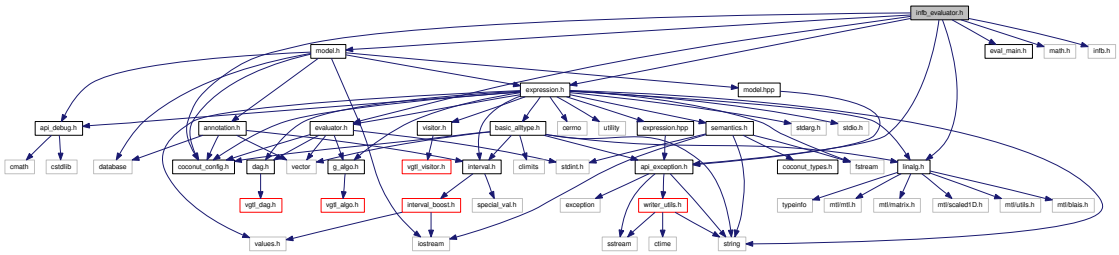
11.84.2.2 #define DEBUG_IHESS_EVALUATOR 0

Definition at line 30 of file ihess_evaluator.h.

11.85 infb_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <model.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <infb.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)

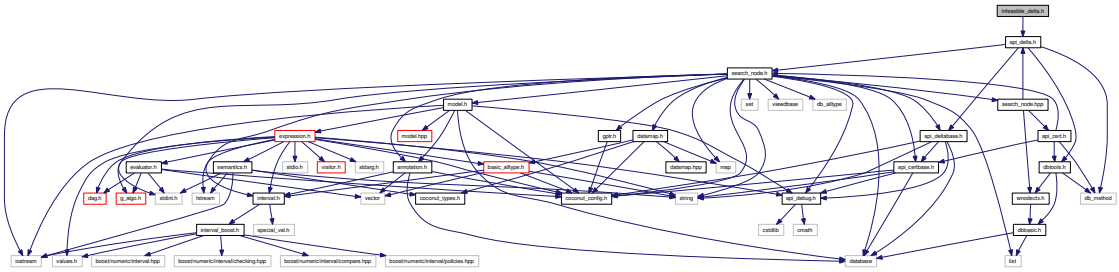
11.85.1 Detailed Description

Definition in file infb_evaluator.h.

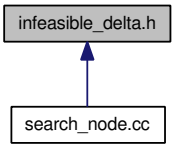
11.86 infeasible_delta.h File Reference

```
#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

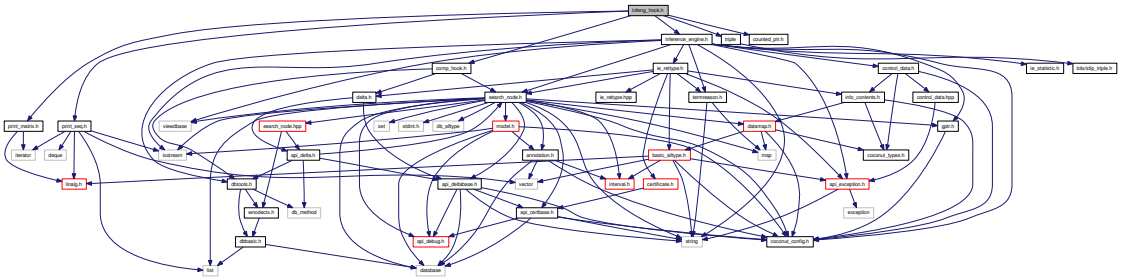
11.86.1 Detailed Description

Definition in file infeasible_delta.h.

11.87 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

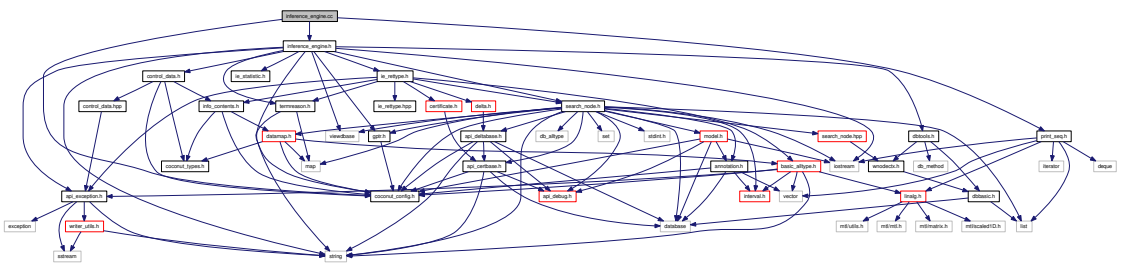
11.87.1 Detailed Description

Definition in file infeng_hook.h.

11.88 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

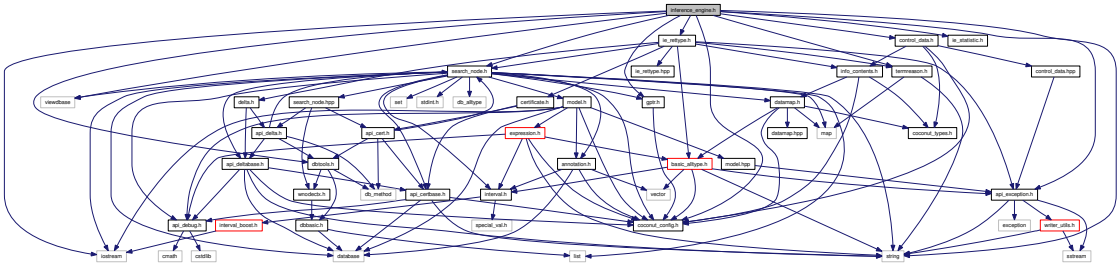
11.88.1 Detailed Description

Definition in file inference_engine.cc.

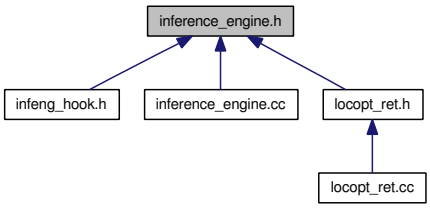
11.89 inference_engine.h File Reference

```
#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 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

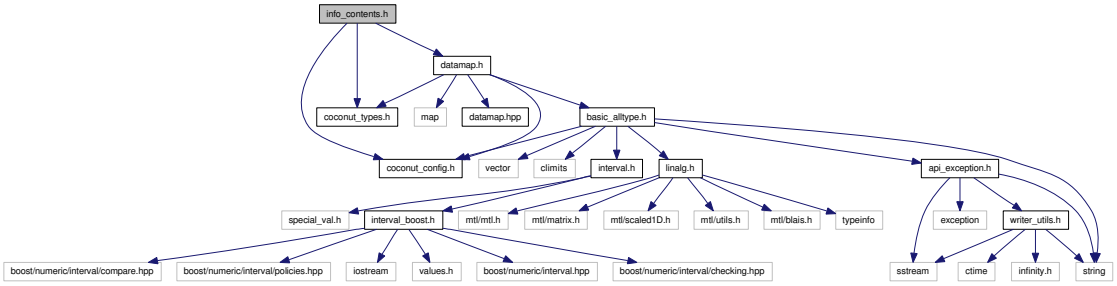
11.89.1 Detailed Description

Definition in file inference_engine.h.

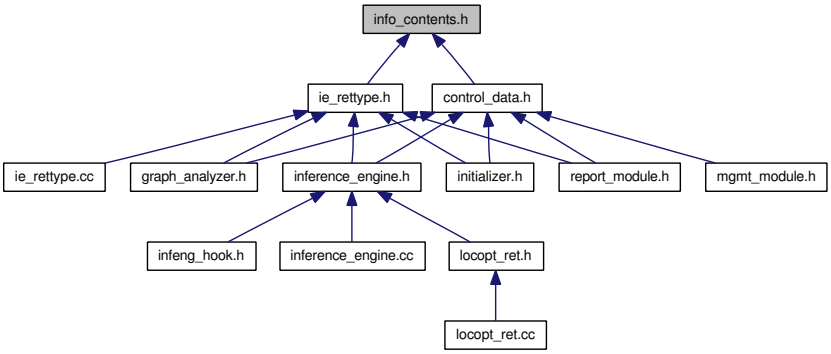
11.90 info_contents.h File Reference

```
#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

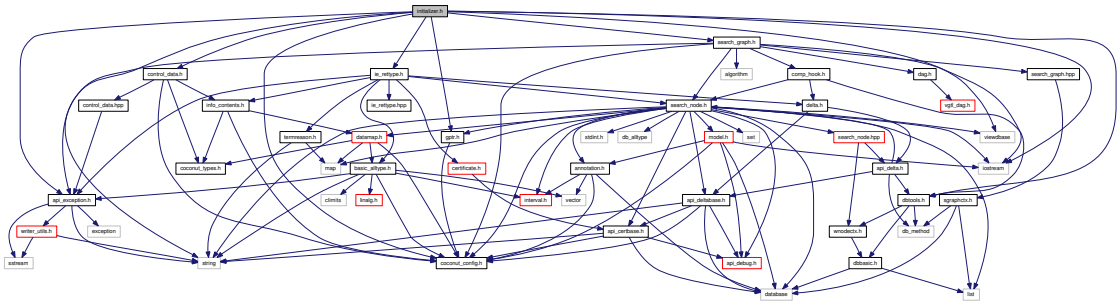
11.90.1 Detailed Description

Definition in file info_contents.h.

11.91 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

11.91.1 Detailed Description

Definition in file initializer.h.

11.92 int_evaluator.h File Reference

```
#include <coconut_config.h>

#include <evaluator.h>

#include <expression.h>

#include <model.h>

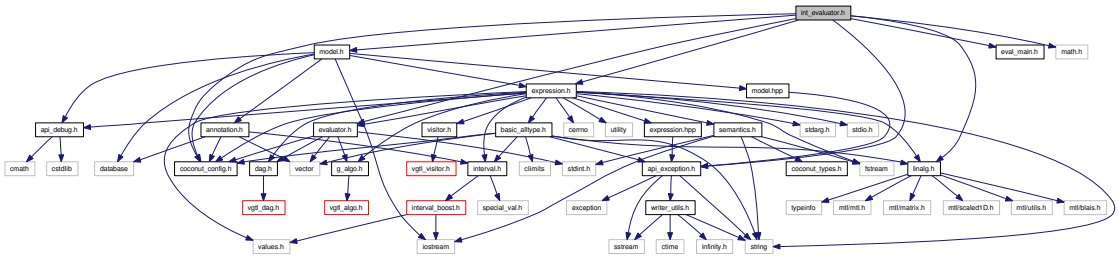
#include <eval_main.h>

#include <linalg.h>

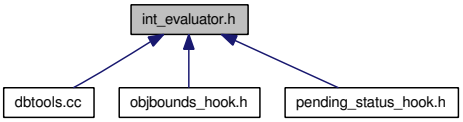
#include <math.h>

#include <api_exception.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)

11.92.1 Detailed Description

Definition in file int_evaluator.h.

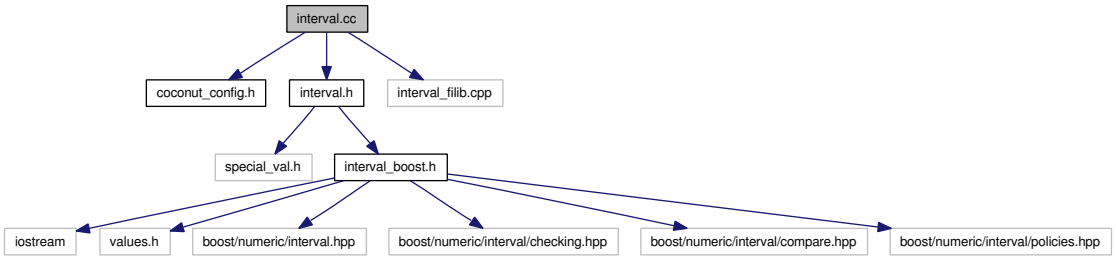
11.93 interval.cc File Reference

```
#include <coconut_config.h>
```

```
#include <interval.h>
```

```
#include <interval_filib.cpp>
```

Include dependency graph for interval.cc:



11.93.1 Detailed Description

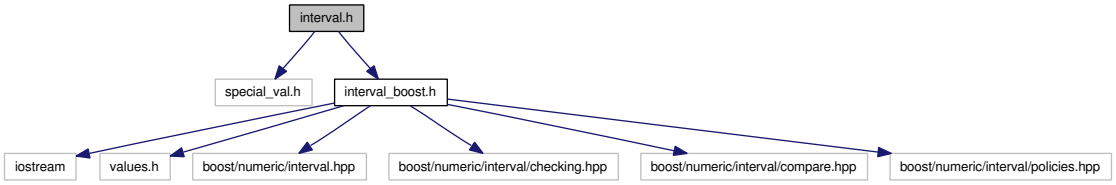
Definition in file interval.cc.

11.94 interval.h File Reference

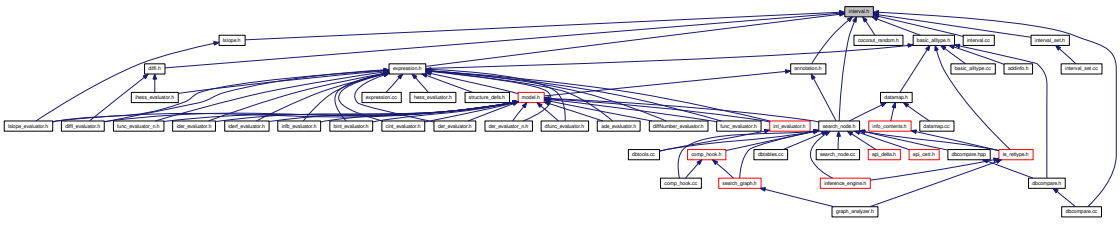
```
#include <special_val.h>
```

```
#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)))

11.94.1 Detailed Description

Definition in file interval.h.

11.94.2 Define Documentation

11.94.2.1 #define get_integer(x) ((int) rint((x)))

This macro returns the integer part of a double (as integer).

Definition at line 56 of file interval.h.

11.94.2.2 #define is_integer(x) (rint((x)) == (x) && rint((x)) < MAXINT && rint((x)) > -MAXINT)

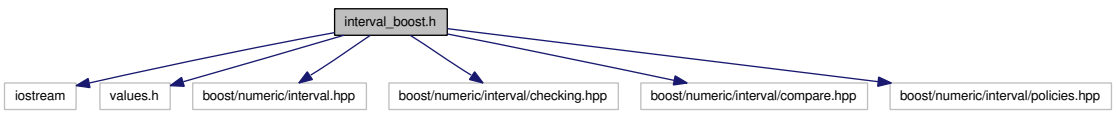
This macro checks whether a double number represents an integer.

Definition at line 52 of file interval.h.

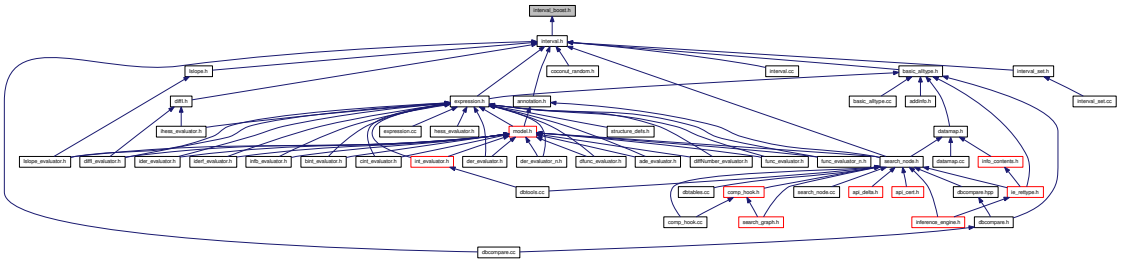
11.95 interval_boost.h File Reference

```
#include <iostream>
#include <values.h>
#include <boost/numeric/interval.hpp>
#include <boost/numeric/interval/checking.hpp>
#include <boost/numeric/interval/compare.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< T >
- struct coco::my_rounded_math< T >
- 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()

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, double b)
- bool coco::operator!= (const interval &a, double b)

- interval coco::cancel (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)

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

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

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

- interval coco::division_part1 (const interval &x, const interval &y, bool &b)
- interval coco::division_part2 (const interval &x, const interval &y, bool b=true)

11.95.1 Detailed Description

Definition in file interval_boost.h.

11.95.2 Define Documentation

11.95.2.1 #define coconut_init_interval()

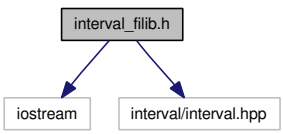
Definition at line 45 of file interval_boost.h.

11.96 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)
- 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)
- interval coco::cancel (const interval &a, const interval &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)

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

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

- 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)

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

- interval coco::division_part1 (const interval &x, const interval &y, bool &b)
- interval coco::division_part2 (const interval &x, const interval &y, bool b=true)

11.96.1 Detailed Description

Definition in file interval_filib.h.

11.96.2 Define Documentation

11.96.2.1 #define coconut_init_interval() filib::fp_traits<double>::setup()

Definition at line 45 of file interval_filib.h.

11.96.2.2 #define FILIB_EXTENDED 1

Definition at line 32 of file interval_filib.h.

11.96.2.3 #define FILIB_USE_MACROS 0

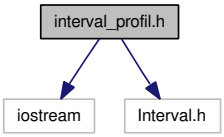
Definition at line 31 of file interval_filib.h.

11.97 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)
- 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)
- interval coco::cancel (const interval &a, const interval &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)

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

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

- 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)

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

- interval coco::division_part1 (const interval &x, const interval &y, bool &b)
- interval coco::division_part2 (const interval &x, const interval &y, bool b=true)

11.97.1 Detailed Description

Definition in file interval_profil.h.

11.97.2 Define Documentation

11.97.2.1 #define coconut_init_interval()

Definition at line 37 of file interval_profil.h.

11.98 interval_set.cc File Reference

```
#include <coconut_config.h>

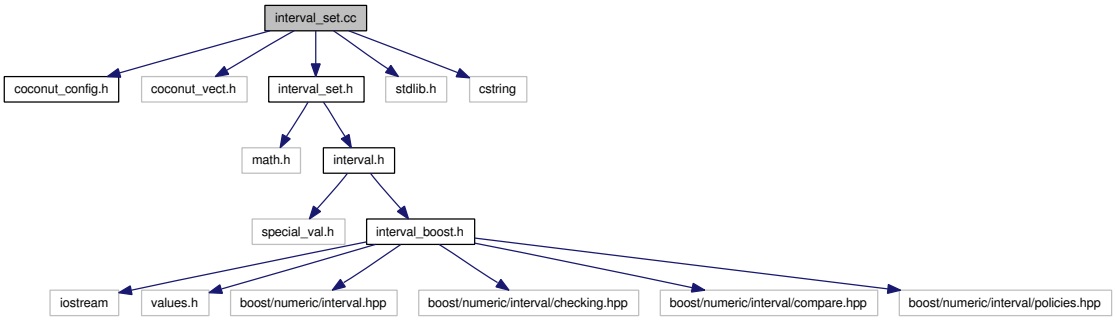
#include <coconut_vect.h>

#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.

Functions

- double coco::width (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::log (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::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)

11.98.1 Detailed Description

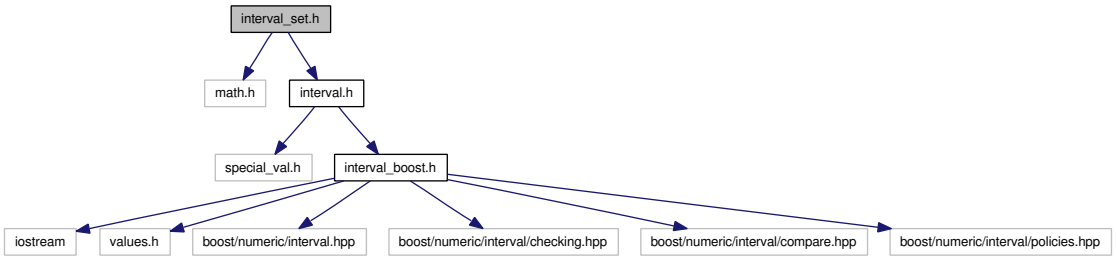
Definition in file interval_set.cc.

11.99 interval_set.h File Reference

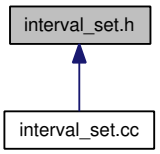
```
#include <math.h>
```

```
#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)
- 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)
- 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::log (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::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)

11.99.1 Detailed Description

Definition in file interval_set.h.

11.99.2 Define Documentation

11.99.2.1 #define SET_BASE_LENGTH 5

Definition at line 37 of file interval_set.h.

11.99.2.2 #define SET_DEF_FILL 2

Definition at line 40 of file interval_set.h.

11.99.2.3 #define SET_DEF_MAX_LENGTH 20

Definition at line 39 of file interval_set.h.

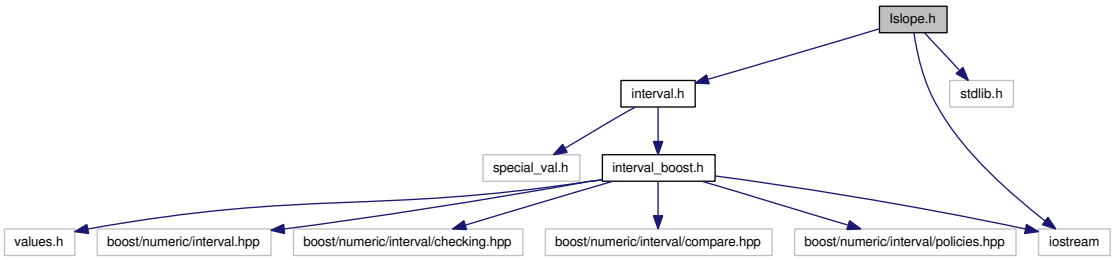
11.99.2.4 #define SET_EXT 5

Definition at line 38 of file interval_set.h.

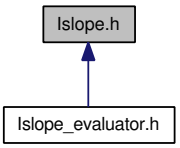
11.100 Islope.h File Reference

```
#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 power (const Islope &a, int n)
- Islope log (const Islope &a)
- Islope atan (const Islope &a)
- Islope atan2 (const Islope &a, const Islope &b)

11.100.1 Typedef Documentation

11.100.1.1 typedef interval I

Definition at line 10 of file Islope.h.

11.100.2 Function Documentation

11.100.2.1 Islope atan (const Islope & *a*) [inline]

Used to compute the slope for the function atan(x) for a given x.

Definition at line 1658 of file Islope.h.

11.100.2.2 Islope atan2 (const Islope & *a*, const Islope & *b*) [inline]

Used to compute the slope for the function atan2(x) for a given x.

Definition at line 1747 of file Islope.h.

11.100.2.3 Islope cos (const Islope & *a*) [inline]

Used to compute the slope for the function cos(x) for a given x.

Definition at line 1090 of file Islope.h.

11.100.2.4 Islope exp (const Islope & *a*) [inline]

Used to compute the slope for the function exp(x) for a given x.

Definition at line 796 of file Islope.h.

11.100.2.5 Islope log (const Islope & *a*) [inline]

Used to compute the slope for the function log(x) for a given x.

Definition at line 1573 of file Islope.h.

11.100.2.6 Islope operator∗ (const Islope & *a*, const double & *b*) [inline]

Operator ∗. Multiplicates slope a with double b.

Definition at line 695 of file Islope.h.

11.100.2.7 Islope operator∗ (const double & *a*, const Islope & *b*) [inline]

Operator ∗. Multiplicates double a with slope b.

Definition at line 668 of file Islope.h.

11.100.2.8 Islope operator∗ (const Islope & *a*, const Islope & *b*) [inline]

Operator ∗. Multiplicates slope a with slope b.

Definition at line 632 of file Islope.h.

11.100.2.9 Islope operator+ (const Islope & *a*, const double & *b*) [inline]

Opertor +. Adds slope a and double b.

Definition at line 488 of file Islope.h.

11.100.2.10 Islope operator+ (const double & *a*, const Islope & *b*) [inline]

Opertor +. Adds slope b and double a.

Definition at line 461 of file Islope.h.

11.100.2.11 Islope operator+ (const Islope & *a*, const Islope & *b*) [inline]

Opertor +. Adds slope a and b.

Definition at line 425 of file Islope.h.

11.100.2.12 Islope operator- (const Islope & *a*) [inline]

Operator -. Changes sign of slope a.

Definition at line 605 of file Islope.h.

11.100.2.13 Islope operator- (const Islope & *a*, const double & *b*) [inline]

Operator -. Subtracts double b from slope a.

Definition at line 578 of file Islope.h.

11.100.2.14 Islope operator- (const double & *a*, const Islope & *b*) [inline]

Operator -. Subtracts slope b from daouble a.

Definition at line 551 of file Islope.h.

11.100.2.15 Islope operator- (const Islope & *a*, const Islope & *b*) [inline]

Operator -. Subtracts slope b from slope a.

Definition at line 515 of file Islope.h.

11.100.2.16 Islope operator/ (const Islope & *a*, const double & *b*) [inline]

Operator /. Divides the slope a by double b.

Definition at line 769 of file Islope.h.

11.100.2.17 Islope operator/ (const double & *a*, const Islope & *b*) [inline]

Operator /. Divides the double a by slope b.

Definition at line 722 of file Islope.h.

11.100.2.18 Islope operator/ (const Islope & *a*, const Islope & *b*) [inline]

Operator /. Divides the slope a by slope b.

Definition at line 737 of file Islope.h.

11.100.2.19 std::ostream & operator<< (std::ostream & *s*, const Islope & *a*) [inline]

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.

Definition at line 1761 of file Islope.h.

11.100.2.20 Islope power (const Islope & *a*, int *n*) [inline]

Used to compute the slope for the function power(x,n) for a given x.

Definition at line 1345 of file Islope.h.

11.100.2.21 Islope sin (const Islope & *a*) [inline]

Used to compute the slope for the function sin(x) for a given x.

Definition at line 877 of file Islope.h.

11.100.2.22 Islope sqrt (const Islope & *a*) [inline]

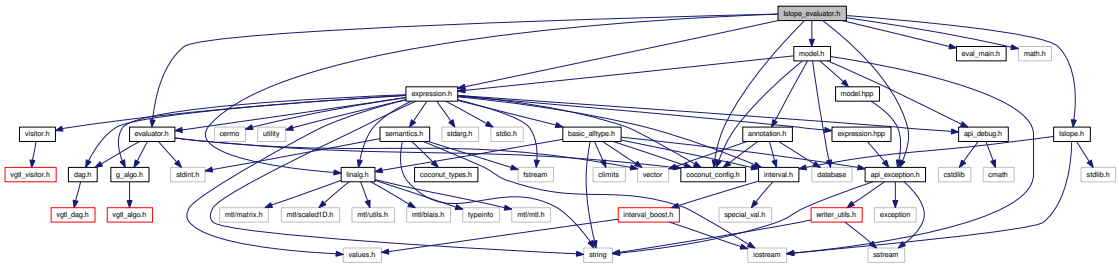
Used to compute the slope for the function sqrt(x) for a given x.

Definition at line 1303 of file Islope.h.

11.101 Islope_evaluator.h File Reference

```
#include <coconut_config.h>
#include <evaluator.h>
#include <expression.h>
#include <model.h>
#include <eval_main.h>
#include <linalg.h>
#include <math.h>
#include <api_exception.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)

11.101.1 Detailed Description

Definition in file Islope_evaluator.h.

11.102 islp_evaluator.h File Reference

```
#include <coconut_config.h>

#include <evaluator.h>

#include <model.h>

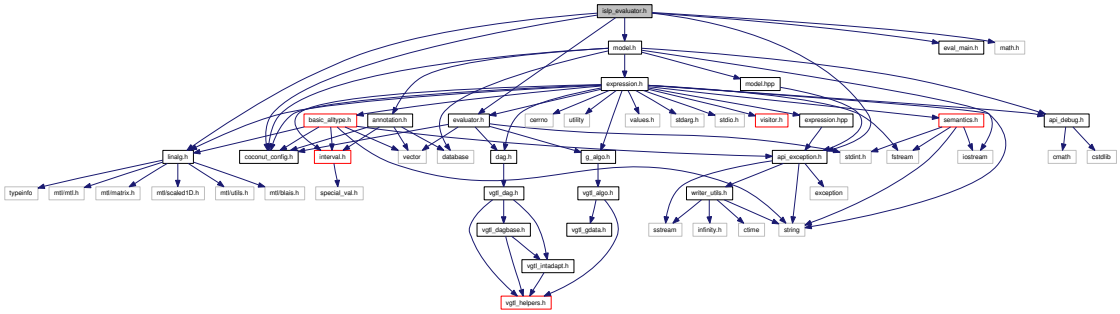
#include <eval_main.h>

#include <linalg.h>

#include <math.h>
```

```
#include <api_exception.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

  Visitor data for islp_eval.

- class coco::islp_eval

  Backward first order slope evaluation with prepared first order slope data.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

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)

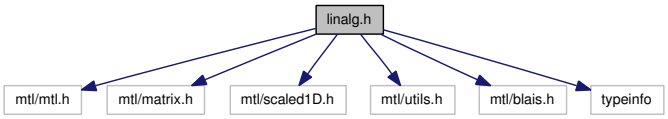
11.102.1 Detailed Description

Definition in file islp_evaluator.h.

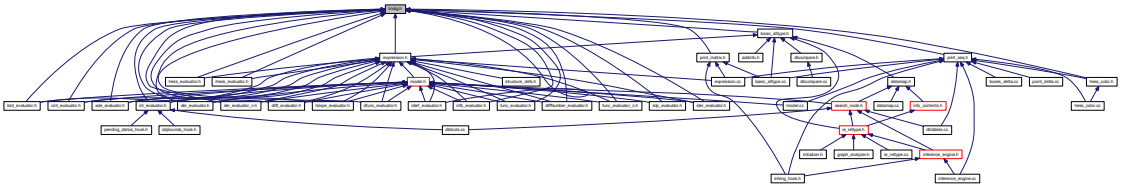
11.103 linalg.h File Reference

```
#include <mtl/mtl.h>
#include <mtl/matrix.h>
#include <mtl/scaled1D.h>
#include <mtl/utils.h>
#include <mtl/blais.h>
#include <typeinfo>
```

Include dependency graph for linalg.h:



This graph shows which files directly or indirectly include this file:



Classes

- class linalg::matrix< _Tp >
  Sparse Matrix Template Class (row major).
- class linalg::c_matrix< _Tp >
  Sparse Matrix Template Class (column major).
- class linalg::sparse_vector< _Tp >
  Sparse Vector Template Class.
- class linalg::__linalg_cvec< _TS >
  Adapter class for a const std::vector to a linear algebra vector.
- class linalg::__linalg_vec< _TS >
  Adapter class for a std::vector to a linear algebra vector.

Namespaces

- namespace linalg

  The namespace for all linear algebra types and functions.

Defines

- #define __LINALG_RANGE_CHECK 0
- #define linalg_trans(A) mtl::trans((A))
- #define linalg_scale(X, Y) mtl::scaled(X, Y)

Enumerations

- enum linalg::linalg_output_format { linalg::linalg_output_sparse, linalg::linalg_output_full }

  The possible output formats of a matrix.

Functions

- template<class _Tp >
  matrix< _Tp > linalg::operator+ (const matrix< _Tp > &A, const matrix< _Tp > &B)

  Sparse matrix addition operator.

- template<class _Tp >
  matrix< _Tp > linalg::operator* (_Tp a, const matrix< _Tp > &B)

  Sparse scalar matrix multiplication operator.

- template<class _Tp >
  c_matrix< _Tp > linalg::operator+ (const c_matrix< _Tp > &A, const c_matrix< _Tp > &B)

  Sparse matrix addition operator.

- template<class _Tp >
  c_matrix< _Tp > linalg::operator* (_Tp a, const c_matrix< _Tp > &B)

  Sparse matrix scalar multiplication operator.

- template<class _Tp >
  bool linalg::operator== (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector comparison operator.

- template<class _Tp >
  sparse_vector< _Tp > linalg::operator+ (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector addition operator.

- template<class _Tp >
  _Tp linalg::operator* (const sparse_vector< _Tp > &a, const sparse_vector< _Tp > &b)

  Sparse vector scalar product operator.

- template<class _Tp >
  sparse_vector< _Tp > linalg::operator* (_Tp a, const sparse_vector< _Tp > &b)

  Sparse vector scaling operator.

- template<class _Ve >
  std::vector< _Ve > & linalg::linalg_ssum (std::vector< _Ve > &a, _Ve b, const std::vector< _Ve > &c)
- template<class _Ve >
  std::vector< _Ve > & linalg::linalg_smult (std::vector< _Ve > &a, _Ve b)

- template<class _TS >
  _TS linalg::linalg_dot (const std::vector< _TS > &a, const std::vector< _TS > &b, _TS s)
- template<class _Va , class _TS >
  _TS linalg::linalg_dot (const _Va &a, const std::vector< _TS > &b, _TS s)
- template<class _Vb , class _TS >
  _TS linalg::linalg_dot (const std::vector< _TS > &a, const _Vb &b, _TS s)
- template<class _Va , class _Vb , class _TS >
  _TS linalg::linalg_dot (const _Va &a, const _Vb &b, _TS s)

- template<class _TS >
  void linalg::linalg_add (const std::vector< _TS > &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Va , class _TS >
  void linalg::linalg_add (const _Va &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Vb , class _TS >
  void linalg::linalg_add (const std::vector< _TS > &a, const _Vb &b, std::vector< _TS > &c)
- template<class _TS , class _Vc >
  void linalg::linalg_add (const std::vector< _TS > &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Va , class _Vb , class _TS >
  void linalg::linalg_add (const _Va &a, const _Vb &b, std::vector< _TS > &c)
- template<class _Va , class _TS , class _Vc >
  void linalg::linalg_add (const _Va &a, const std::vector< _TS > &b, _Vc &c)
- template<class _TS , class _Vb , class _Vc >
  void linalg::linalg_add (const std::vector< _TS > &a, const _Vb &b, _Vc &c)
- template<class _Va , class _Vb , class _Vc >
  void linalg::linalg_add (const _Va &a, const _Vb &b, _Vc &c)

- template<class _Matrix , class _TS >
  void linalg::linalg_matvec (const _Matrix &A, const std::vector< _TS > &a, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Va >
  void linalg::linalg_matvec (const _Matrix &A, const _Va &a, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Vc >
  void linalg::linalg_matvec (const _Matrix &A, const std::vector< _TS > &a, _Vc &c)
- template<class _Matrix , class _Va , class _Vc >
  void linalg::linalg_matvec (const _Matrix &A, const _Va &a, _Vc &c)

- template<class _Matrix , class _TS >
  void linalg::linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const std::vector< _TS > &b, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Va >
  void linalg::linalg_matvecv (const _Matrix &A, const _Va &a, const std::vector< _TS > &b, std::vector< _TS > &c)

- template<class _Matrix , class _TS , class _Vb >
  void linalg::linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const _Vb &b, std::vector< _TS > &c)
- template<class _Matrix , class _TS , class _Vc >
  void linalg::linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Matrix , class _Va , class _Vb , class _TS >
  void linalg::linalg_matvecv (const _Matrix &A, _Va &a, const _Vb &b, const std::vector< _TS > &c)
- template<class _Matrix , class _Va , class _TS , class _Vc >
  void linalg::linalg_matvecv (const _Matrix &A, const _Va &a, const std::vector< _TS > &b, _Vc &c)
- template<class _Matrix , class _TS , class _Vb , class _Vc >
  void linalg::linalg_matvecv (const _Matrix &A, const std::vector< _TS > &a, const _Vb &b, _Vc &c)
- template<class _Matrix , class _Va , class _Vb , class _Vc >
  void linalg::linalg_matvecv (const _Matrix &A, const _Va &a, const _Vb &b, _Vc &c)

11.103.1 Detailed Description

Definition in file linalg.h.

11.103.2 Define Documentation

11.103.2.1 #define __LINALG_RANGE_CHECK 0

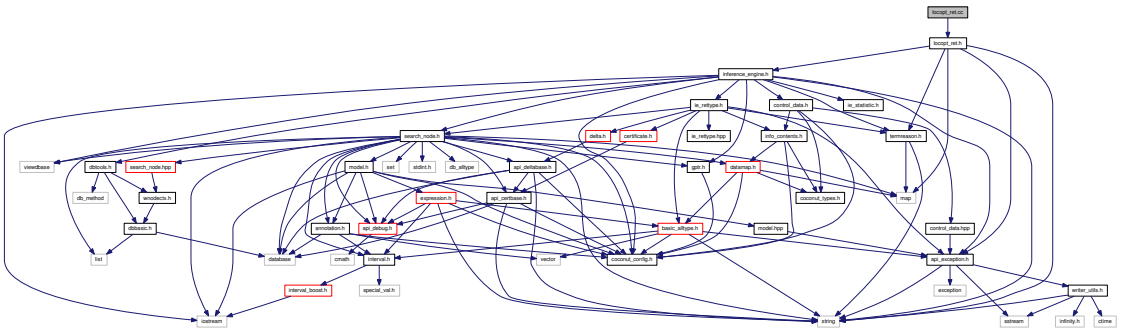
If this macro has a nonzero value, for all linear algebra operations involving matrices or sparse vectors range checks are performed.

Definition at line 215 of file linalg.h.

11.104 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)

11.104.1 Detailed Description

Definition in file locopt_ret.cc.

11.105 locopt_ret.h File Reference

```
#include <string>

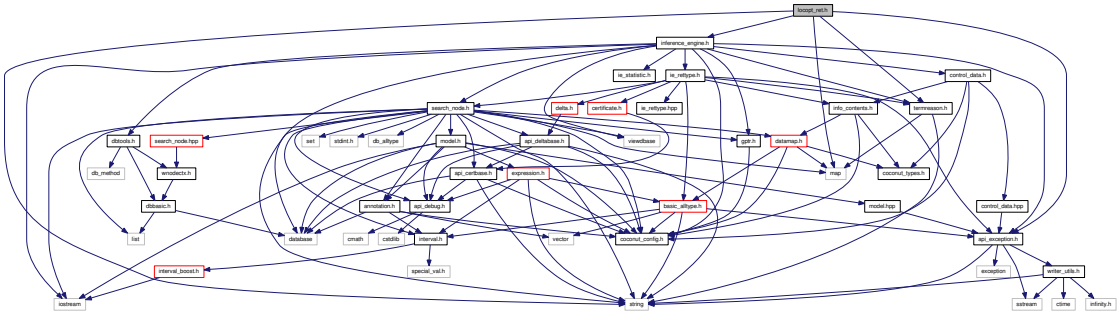
#include <map>

#include <termreason.h>

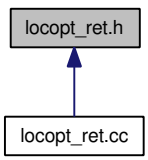
#include <inference_engine.h>

#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)

11.105.1 Detailed Description

Definition in file locopt_ret.h.

11.105.2 Define Documentation

11.105.2.1 #define LOPT_ERROR 5

Definition at line 53 of file locopt_ret.h.

11.105.2.2 #define LOPT_INFEAS 2

Definition at line 50 of file locopt_ret.h.

11.105.2.3 #define LOPT_LIMIT 4

Definition at line 52 of file locopt_ret.h.

11.105.2.4 #define LOPT_PARTSUCC 1

Definition at line 49 of file locopt_ret.h.

11.105.2.5 #define LOPT_SUCCESS 0

Definition at line 48 of file locopt_ret.h.

11.105.2.6 #define LOPT_UNBND 3

Definition at line 51 of file locopt_ret.h.

11.105.2.7 #define LOPT_WGHT_ERROR 0.0

Definition at line 61 of file locopt_ret.h.

11.105.2.8 #define LOPT_WGHT_INFEAS 3.0

Definition at line 58 of file locopt_ret.h.

11.105.2.9 #define LOPT_WGHT_LIMIT 1.0

Definition at line 60 of file locopt_ret.h.

11.105.2.10 #define LOPT_WGHT_PARTSUCC 5.0

Definition at line 57 of file locopt_ret.h.

11.105.2.11 #define LOPT_WGHT_SUCCESS 10.0

Definition at line 56 of file locopt_ret.h.

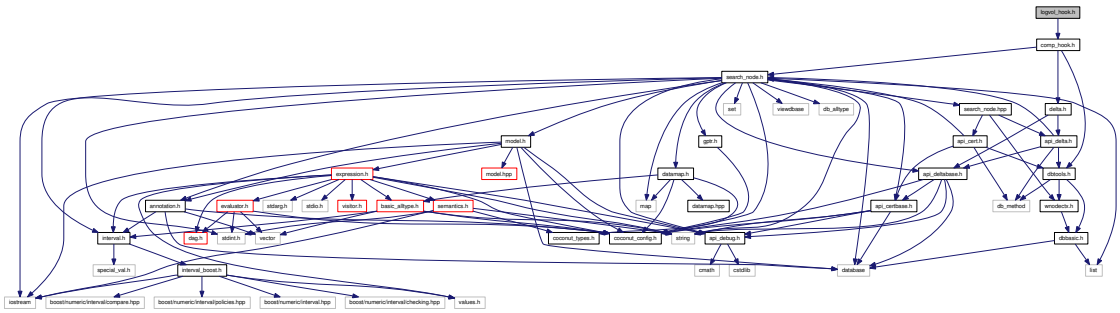
11.105.2.12 #define LOPT_WGHT_UNBND 3.0

Definition at line 59 of file locopt_ret.h.

11.106 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

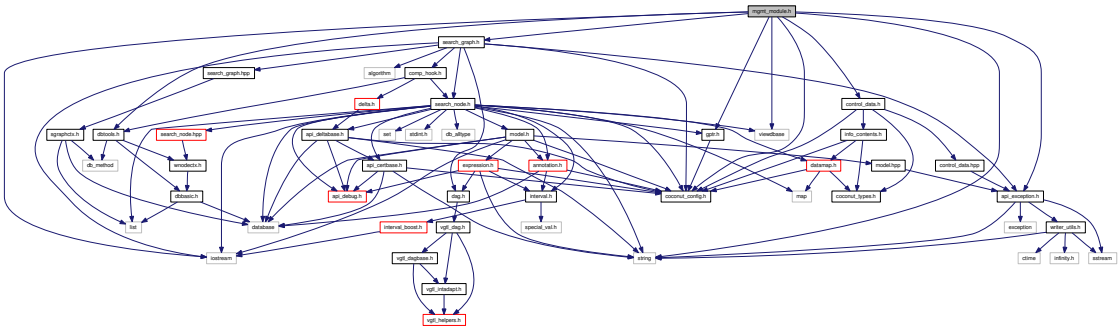
11.106.1 Detailed Description

Definition in file logvol_hook.h.

11.107 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

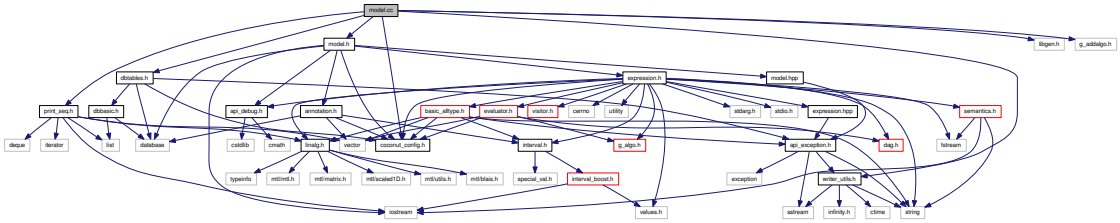
11.107.1 Detailed Description

Definition in file mgmt_module.h.

11.108 model.cc File Reference

```
#include <coconut_config.h>
#include <libgen.h>
#include <model.h>
#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

11.108.1 Detailed Description

Definition in file model.cc.

11.108.2 Define Documentation

11.108.2.1 #define MODEL_INITIAL_BUF_SIZE 1024

Definition at line 539 of file model.cc.

11.108.2.2 #define MT_DBL 1

Definition at line 482 of file model.cc.

11.108.2.3 #define MT_INT 3

Definition at line 484 of file model.cc.

11.108.2.4 #define MT_INTV 2

Definition at line 483 of file model.cc.

11.108.2.5 #define MT_STORED 8

Definition at line 485 of file model.cc.

11.108.2.6 #define throw_read(MSG)

Value:

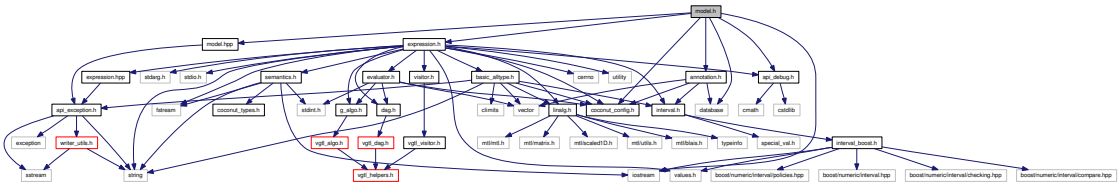
```
do { throw api_exception(apiee_io, \
                                 std::string("Malformed Input: Line ")+\
                                 convert_to_str(ln)+": "\
                                 +MSG); } while(0)
```

Definition at line 478 of file model.cc.

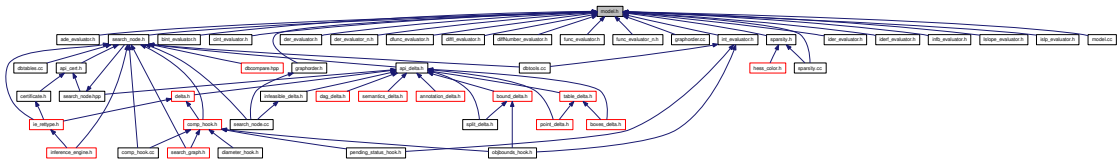
11.109 model.h File Reference

```
#include <iostream>
#include <coconut_config.h>
#include <expression.h>
#include <database>
#include <annotation.h>
#include <api_debug.h>
#include <model.hpp>
```

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.

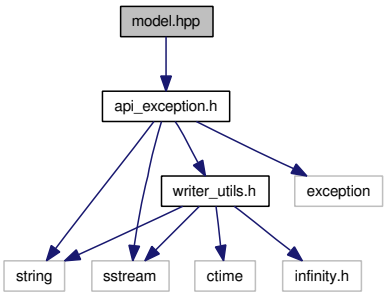
11.109.1 Detailed Description

Definition in file model.h.

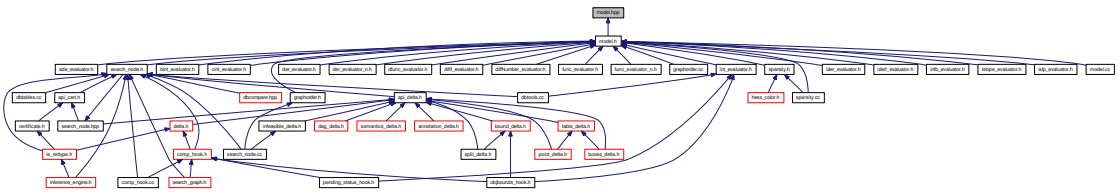
11.110 model.hpp File Reference

```
#include <api_exception.h>
```

Include dependency graph for model.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::model::model::sort_constraints
- class coco::model::model::simplify_visitor_0
- class coco::model::model::simplify_visitor_m
- struct coco::model::model::__docompare_nodes
- struct coco::model::model::__docompare_variables
- struct coco::model::model::detect_0chain_visitor_st
- class coco::model::model::detect_0chain_visitor

- struct coco::model::model::lincoeff_visitor_ret
- struct coco::model::model::lincoeff_visitor_st
- class coco::model::model::lincoeff_visitor

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Defines

- #define delnodes_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)

11.110.1 Detailed Description

Definition in file model.hpp.

11.110.2 Define Documentation

11.110.2.1 #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 35 of file model.hpp.

11.110.2.2 #define SIMPLIFY_0_CONST_IS_EQUATION (1<<25)

Definition at line 698 of file model.hpp.

11.110.2.3 #define SIMPLIFY_0_CONST_IS_INTEGER (1<<24)

Definition at line 697 of file model.hpp.

11.110.2.4 #define SIMPLIFY_0_IS_CONST 1

Definition at line 696 of file model.hpp.

11.110.2.5 #define SIMPLIFY_0_IS_CORRECTEDMULT (1<<5)

Definition at line 709 of file model.hpp.

11.110.2.6 #define SIMPLIFY_0_IS_GHOST (1<<31)

Definition at line 711 of file model.hpp.

11.110.2.7 #define SIMPLIFY_0_IS_MULTIPLICATIVE (1<<4)

Definition at line 708 of file model.hpp.

11.110.2.8 #define SIMPLIFY_0_IS_PROD (1<<3)

Definition at line 705 of file model.hpp.

11.110.2.9 #define SIMPLIFY_0_IS_SUM (1<<2)

Definition at line 702 of file model.hpp.

11.110.2.10 #define SIMPLIFY_0_IS_VAR (1<<1)

Definition at line 700 of file model.hpp.

11.110.2.11 #define SIMPLIFY_0_PROD_IS_SIMPLE (1<<24)

Definition at line 706 of file model.hpp.

11.110.2.12 #define SIMPLIFY_0_SKIP_THIS_NODE (1<<30)

Definition at line 713 of file model.hpp.

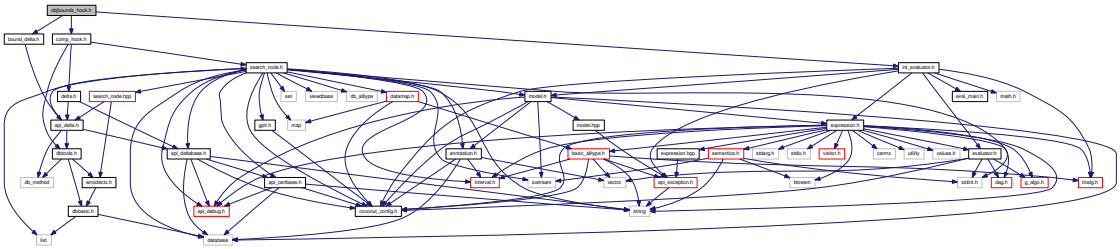
11.110.2.13 #define SIMPLIFY_0_SUM_IS_SIMPLE (1<<24)

Definition at line 703 of file model.hpp.

11.111 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

11.111.1 Detailed Description

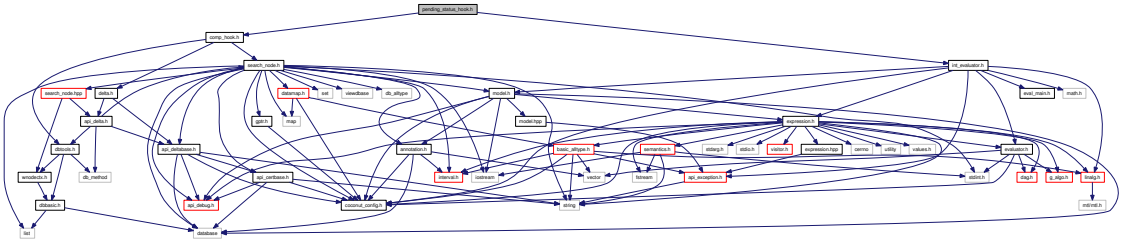
Definition in file objbounds_hook.h.

11.112 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

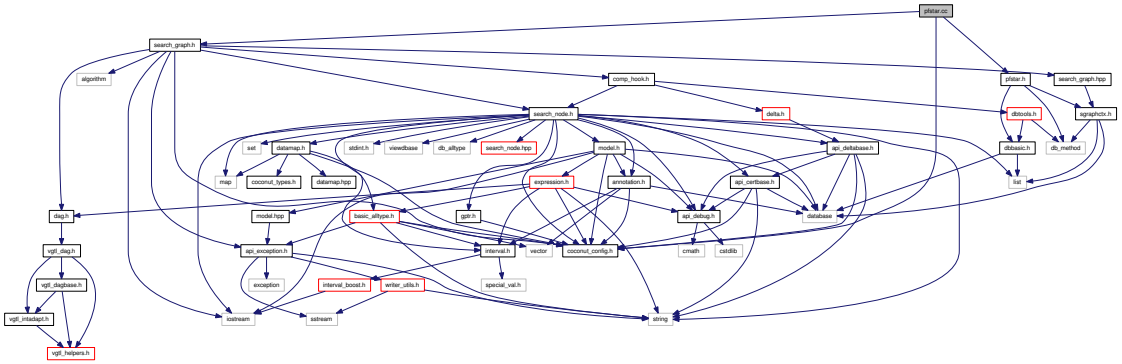
11.112.1 Detailed Description

Definition in file pending_status_hook.h.

11.113 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

11.113.1 Detailed Description

Definition in file pfstar.cc.

11.113.2 Define Documentation

11.113.2.1 #define PFSTAR_DEBUG 0

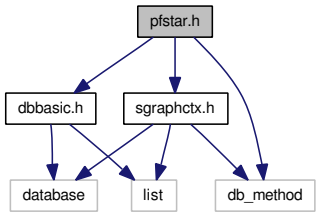
Definition at line 32 of file pfstar.cc.

11.114 pfstar.h File Reference

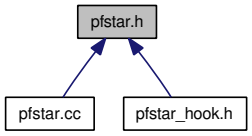
```
#include <dbbasic.h>
```

```
#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

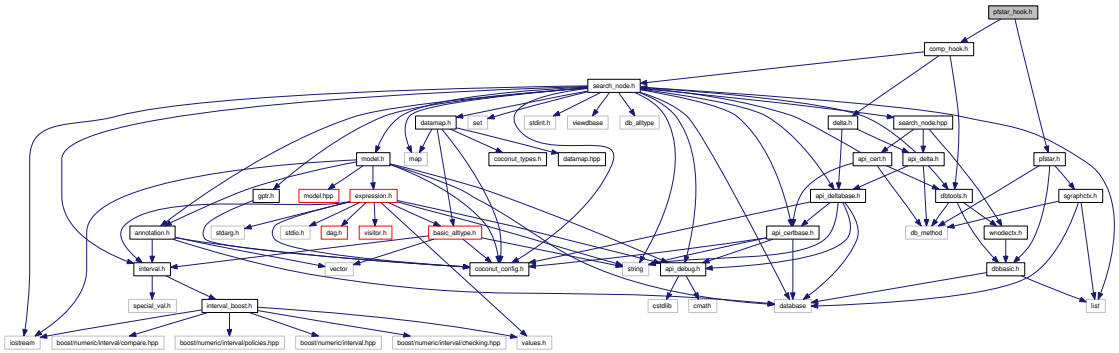
11.114.1 Detailed Description

Definition in file pfstar.h.

11.115 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

11.115.1 Detailed Description

Definition in file pfstar_hook.h.

11.116 point_delta.cc File Reference

```
#include <coconut_config.h>

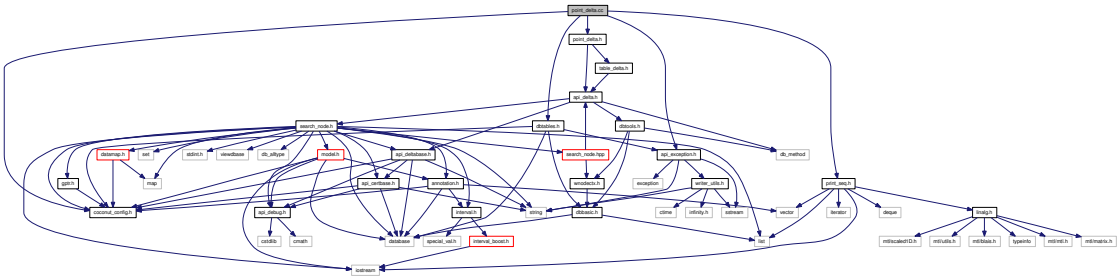
#include <point_delta.h>

#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

11.116.1 Detailed Description

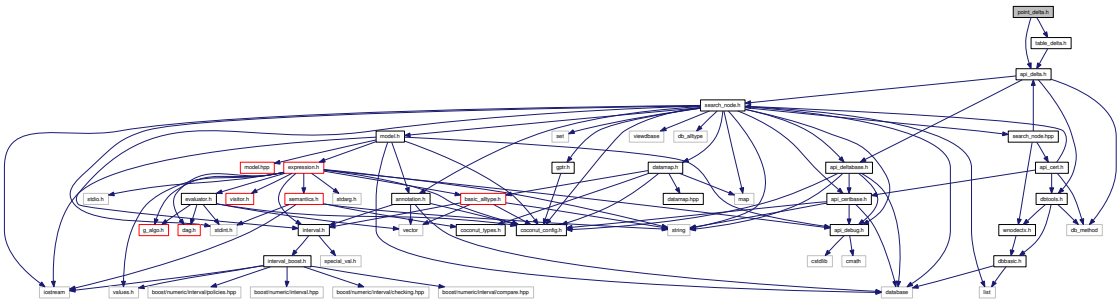
Definition in file point_delta.cc.

11.117 point_delta.h File Reference

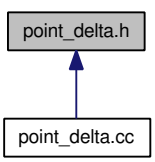
```
#include <api_delta.h>
```

```
#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

11.117.1 Detailed Description

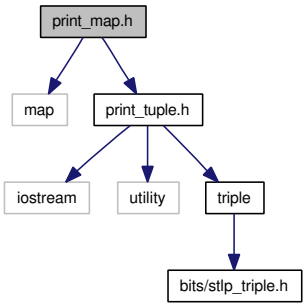
Definition in file point_delta.h.

11.118 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)

11.118.1 Detailed Description

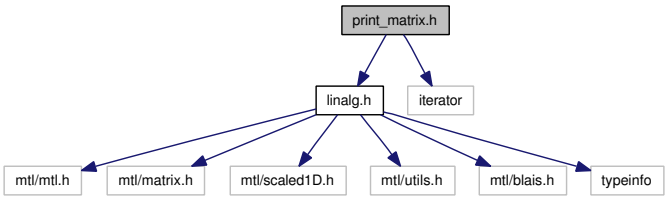
Definition in file print_map.h.

11.119 print_matrix.h File Reference

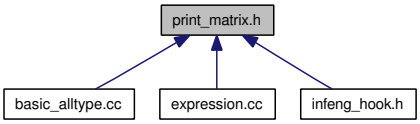
```
#include <linalg.h>
```

```
#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.

Functions

- template<typename _Tp , int NN, int IND_OFFSET>
  ostream & std::operator<< (ostream &o, const mtl::light1D< _Tp, NN, IND_OFFSET > &a)
- template<typename _Tp >
  ostream & std::operator<< (ostream &s, const linalg::matrix< _Tp > &A)
- template<typename _Tp >
  ostream & std::operator<< (ostream &s, const linalg::c_matrix< _Tp > &A)

11.119.1 Detailed Description

Definition in file print_matrix.h.

11.120 print_seq.h File Reference

```
#include <iostream>

#include <vector>

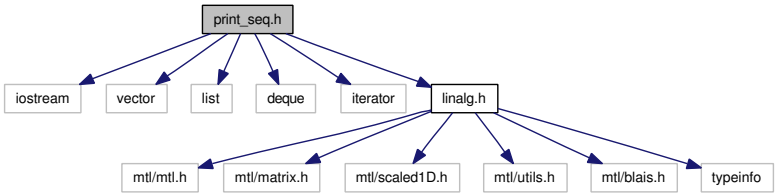
#include <list>

#include <deque>

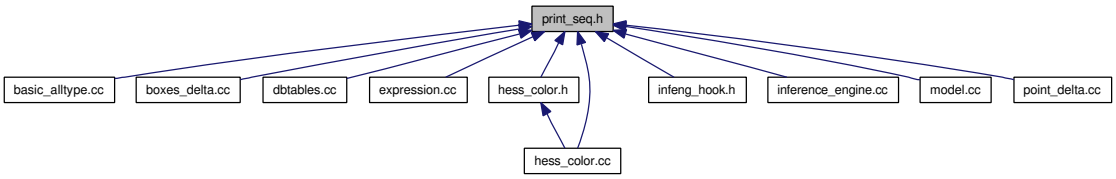
#include <iterator>

#include <linalg.h>
```

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)
- template<typename _Tp >
  ostream & std::operator<< (ostream &s, const linalg::sparse_vector< _Tp > &a)

11.120.1 Detailed Description

Definition in file print_seq.h.

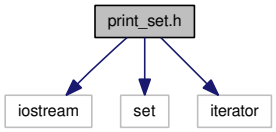
11.121 print_set.h File Reference

```
#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)

11.121.1 Detailed Description

Definition in file print_set.h.

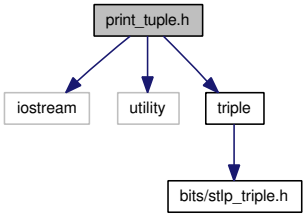
11.122 print_tuple.h File Reference

```
#include <iostream>
```

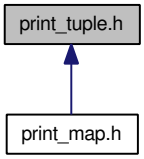
```
#include <utility>
```

```
#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)

11.122.1 Detailed Description

Definition in file print_tuple.h.

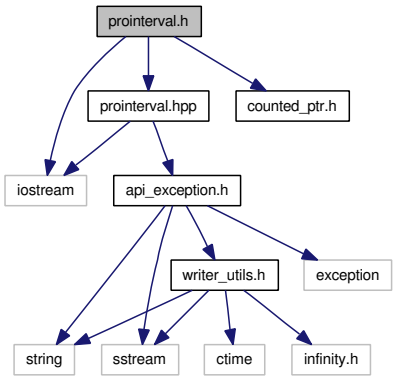
11.123 prointerval.h File Reference

```
#include <iostream>

#include <counted_ptr.h>

#include <prointerval.hpp>
```

Include dependency graph for prointerval.h:



Classes

- class coco::proj_rational
- class coco::projective_interval< I >

Namespaces

- namespace coco

  the main namespace of the COCONUT API

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)

11.123.1 Detailed Description

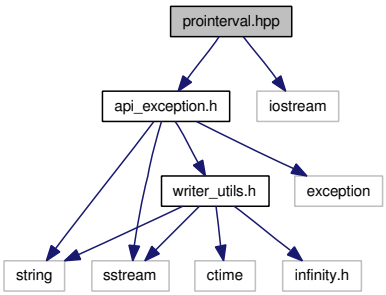
Definition in file prointerval.h.

11.124 prointerval.hpp File Reference

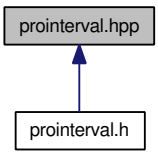
```
#include <api_exception.h>

#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

- namespace std

  The standard namespace.

Functions

- ostream & std::operator<< (ostream &o, const coco::proj_rational &x)
- template<class I >
  ostream & std::operator<< (ostream &o, const coco::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::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::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)

11.124.1 Detailed Description

Definition in file prointerval.hpp.

11.125 report_module.h File Reference

```
#include <iostream>

#include <coconut_config.h>

#include <search_graph.h>

#include <gptr.h>

#include <string>

#include <dbtools.h>
```

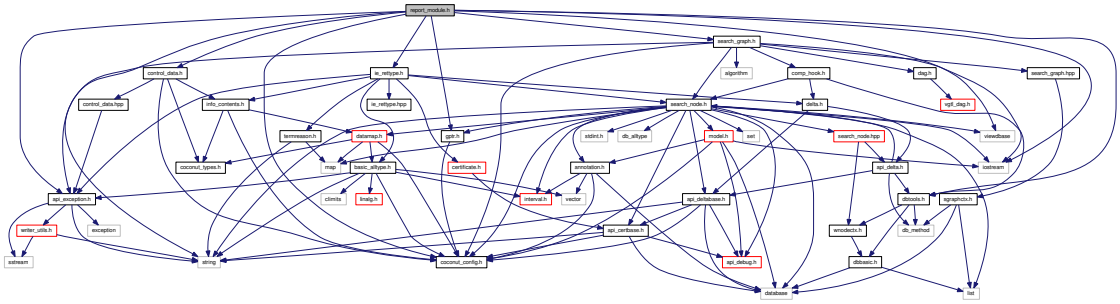
```
#include <viewdbase>

#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

11.125.1 Detailed Description

Definition in file report_module.h.

11.126 search_graph.cc File Reference

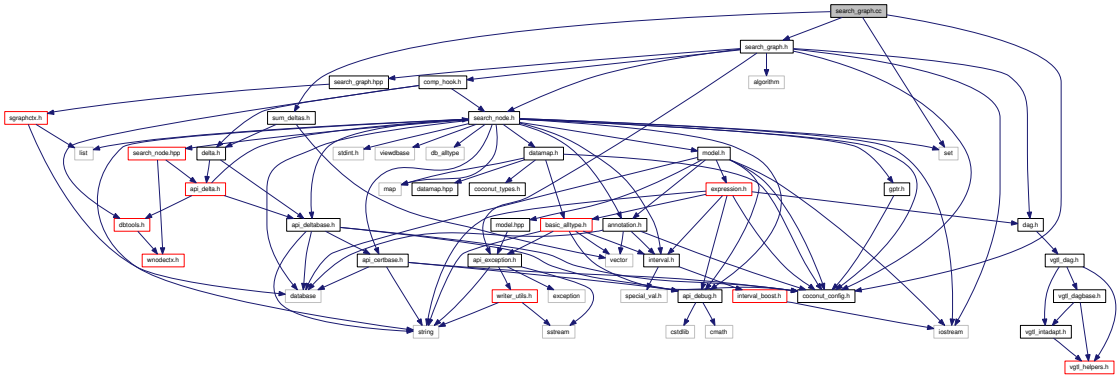
```
#include <coconut_config.h>

#include <search_graph.h>

#include <set>

#include "sum_deltas.h"
```

Include dependency graph for search_graph.cc:



Classes

- 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

Defines

- #define DEBUG_SGRAPH_EXTRACT 0

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.

11.126.1 Detailed Description

Definition in file search_graph.cc.

11.126.2 Define Documentation

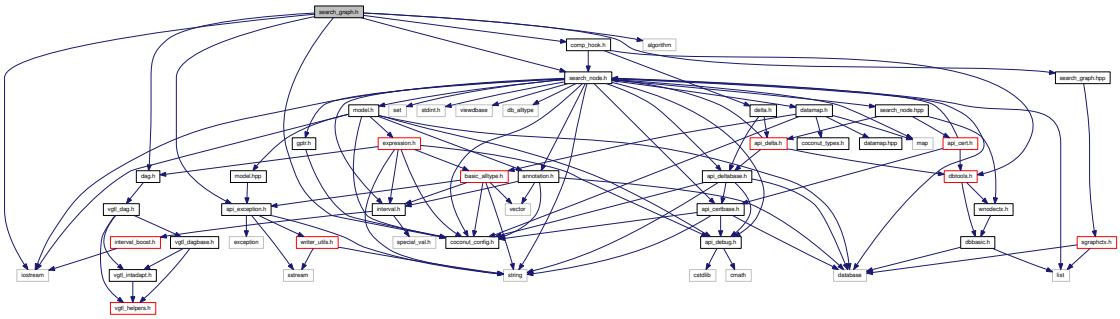
11.126.2.1 #define DEBUG_SGRAPH_EXTRACT 0

Definition at line 39 of file search_graph.cc.

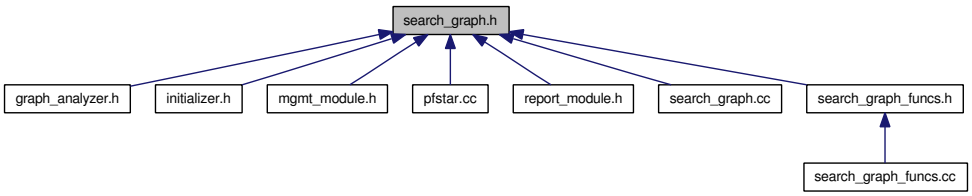
11.127 search_graph.h File Reference

```
#include <iostream>
#include <algorithm>
#include <dag.h>
#include <coconut_config.h>
#include <search_node.h>
#include <comp_hook.h>
#include <api_exception.h>
#include <search_graph.hpp>
```

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.

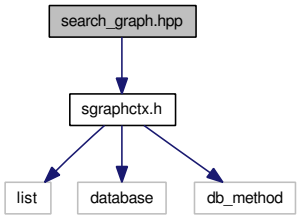
11.127.1 Detailed Description

Definition in file search_graph.h.

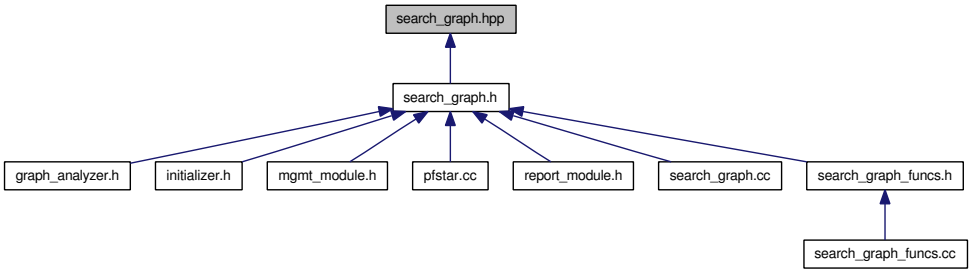
11.128 search_graph.hpp File Reference

```
#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

11.128.1 Detailed Description

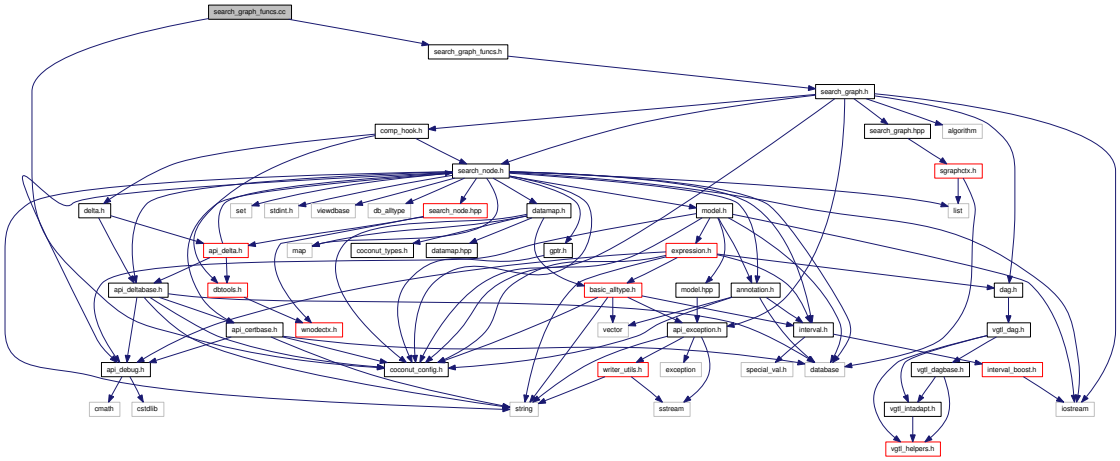
Definition in file search_graph.hpp.

11.129 search_graph_funcs.cc File Reference

```
#include <coconut_config.h>

#include <search_graph_funcs.h>
```

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)

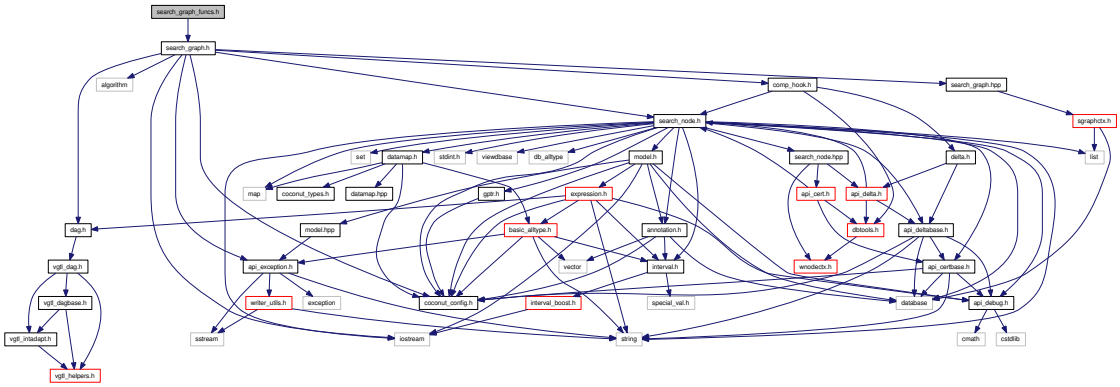
11.129.1 Detailed Description

Definition in file search_graph_funcs.cc.

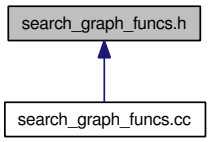
11.130 search_graph_funcs.h File Reference

```
#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)

11.130.1 Detailed Description

Definition in file search_graph_funcs.h.

11.131 search_node.cc File Reference

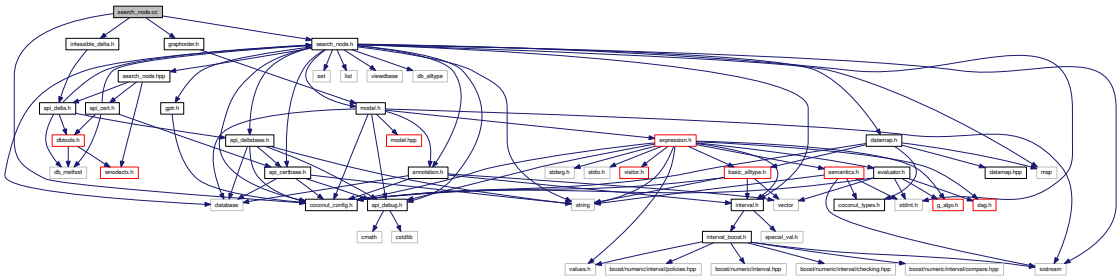
```
#include <coconut_config.h>

#include <search_node.h>

#include <infeasible_delta.h>

#include <graphorder.h>
```

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.

11.131.1 Detailed Description

Definition in file search_node.cc.

11.131.2 Define Documentation

11.131.2.1 #define MAX_VOL_COMP 1.e12

Definition at line 268 of file search_node.cc.

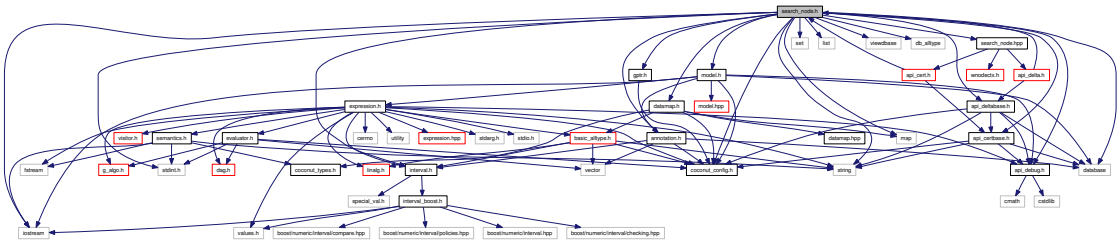
11.131.2.2 #define MIN_VOL_WIDTH DBL_MIN

Definition at line 269 of file search_node.cc.

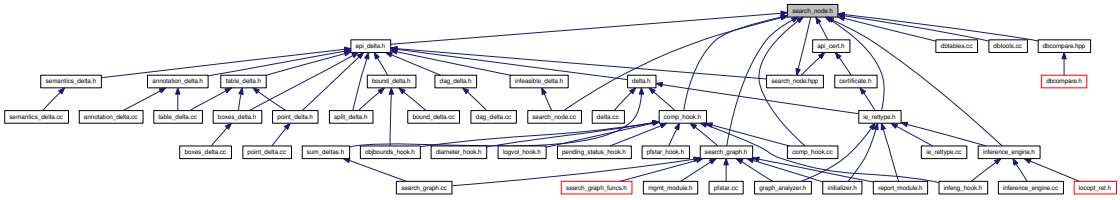
11.132 search_node.h File Reference

```
#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 <api_deltabase.h>
#include <api_certbase.h>
#include <search_node.hpp>
```

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::work_node::constraint_iterator_base< _TW, _TV, _VR, _TP, _TR, _TI >

  The base class for work_node::constraint_iterator and work_node::constraint_const_iterator.

Namespaces

- namespace coco

  the main namespace of the COCONUT API

Typedefs

- typedef uint32_t coco::search_node_id

  Type of the unique search node identifier.

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.

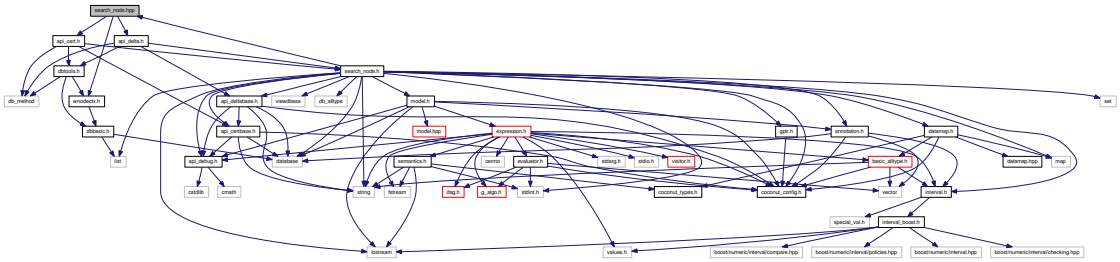
11.132.1 Detailed Description

Definition in file search_node.h.

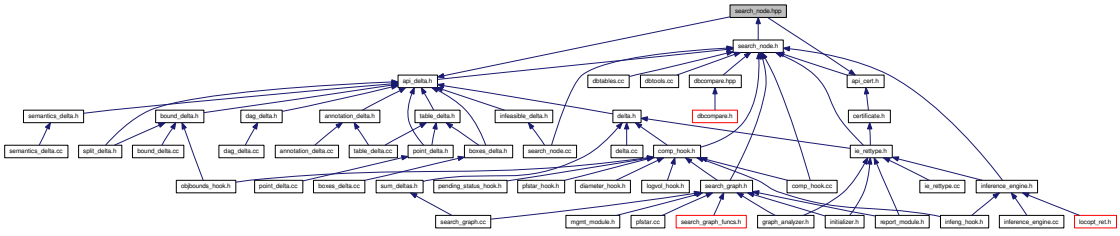
11.133 search_node.hpp File Reference

```
#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.

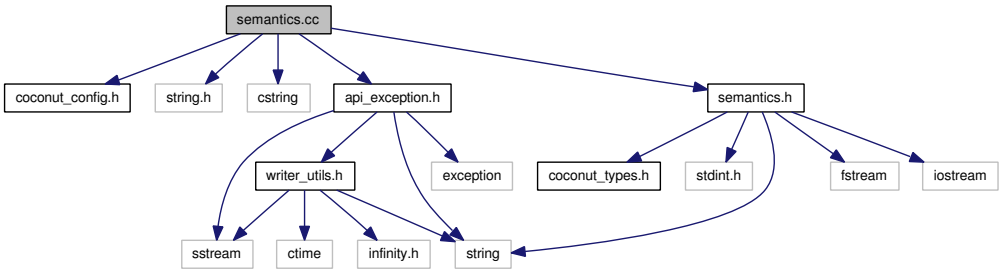
11.133.1 Detailed Description

Definition in file search_node.hpp.

11.134 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.

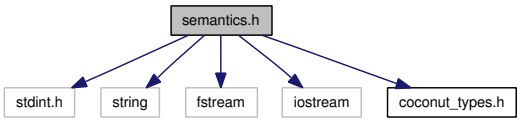
11.134.1 Detailed Description

Definition in file semantics.cc.

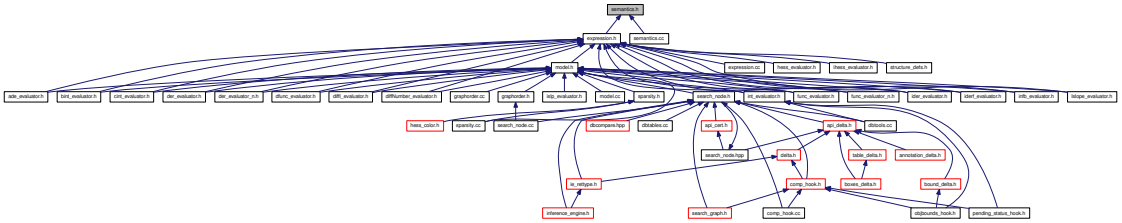
11.135 semantics.h File Reference

```
#include <stdint.h>
#include <string>
#include <fstream>
#include <iostream>
#include <coconut_types.h>
```

Include dependency graph for semantics.h:



This graph shows which files directly or indirectly include this file:



Classes

- class coco::convex_e

  Convexity information.

- class coco::semantics

  Expression Semantics.

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 }

  Convexity information enum.

- 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

- 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)
  Equality comparison operator with a convex_info.
- bool coco::operator== (const convex_info &__c, const convex_e &__d)
  Equality comparison operator with a convex_info.
- 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)
  Disequality comparison operator with a convex_info.
- bool coco::operator!= (const convex_info &__c, const convex_e &__d)
  Disequality comparison operator with a convex_info.
- std::ostream & coco::operator<< (std::ostream &o, const convex_e &__s)
  C++ stream output operator for convex_e.
- std::ostream & coco::operator<< (std::ostream &o, const semantics &__s)
  C++ stream output operator for semantics.

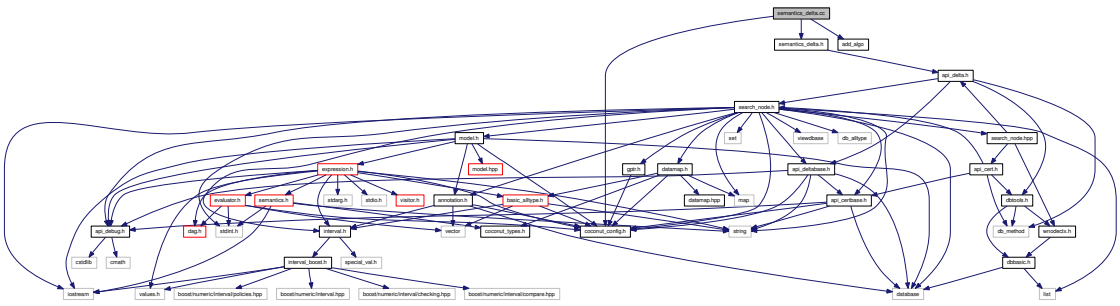
11.135.1 Detailed Description

Definition in file semantics.h.

11.136 semantics_delta.cc File Reference

```
#include <coconut_config.h>
#include <semantics_delta.h>
#include <add_algo>
```

Include dependency graph for semantics_delta.cc:



Namespaces

- namespace coco

  the main namespace of the COCONUT API

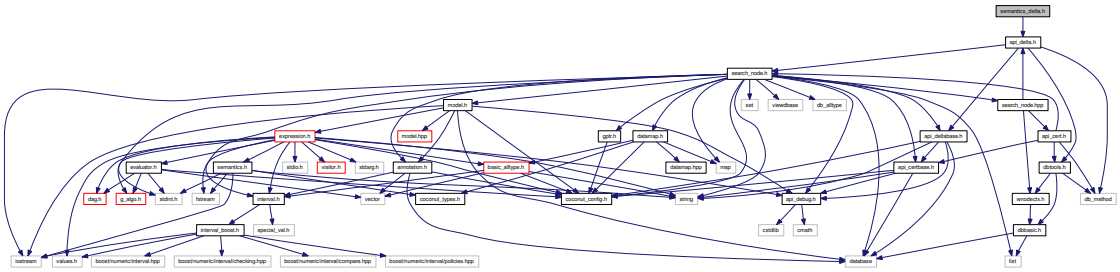
11.136.1 Detailed Description

Definition in file semantics_delta.cc.

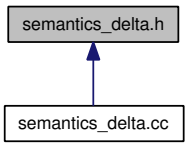
11.137 semantics_delta.h File Reference

```
#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

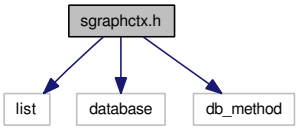
11.137.1 Detailed Description

Definition in file semantics_delta.h.

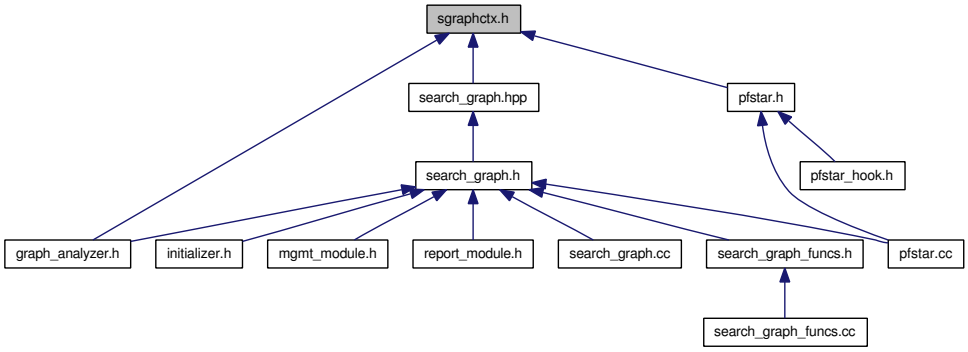
11.138 sgraphctx.h File Reference

```
#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

11.138.1 Detailed Description

Definition in file sgraphctx.h.

11.139 sparsity.cc File Reference

```
#include <coconut_config.h>

#include <dag.h>

#include <model.h>

#include <vector>

#include <algorithm>

#include <g_algo.h>

#include <cmath>

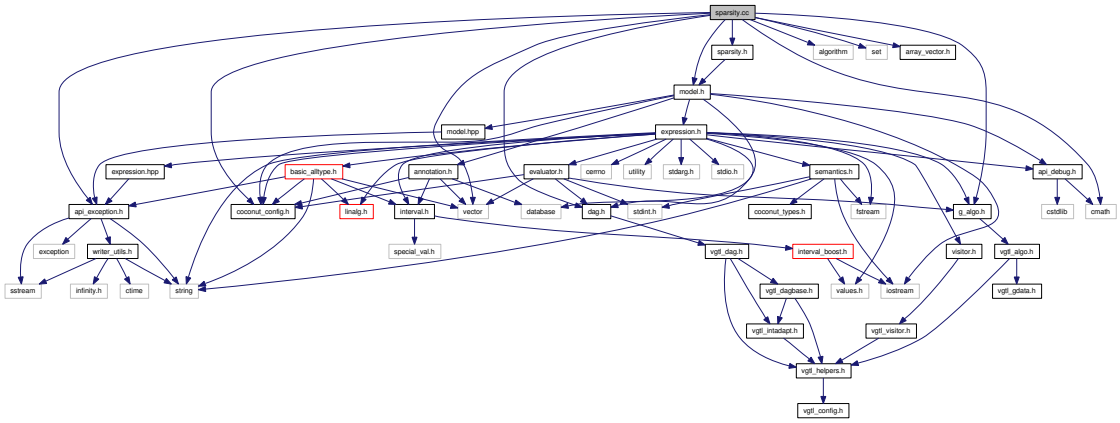
#include <set>

#include <api_exception.h>

#include <array_vector.h>

#include <sparsity.h>
```

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, 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, 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, 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::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, 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_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).

11.139.1 Detailed Description

Definition in file sparsity.cc.

11.139.2 Define Documentation

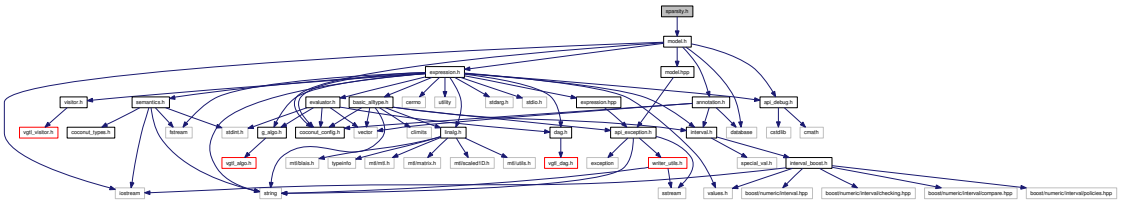
11.139.2.1 #define SP_DBG 0

Definition at line 40 of file sparsity.cc.

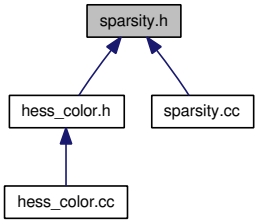
11.140 sparsity.h File Reference

```
#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 }

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, 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::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, 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_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).

11.140.1 Detailed Description

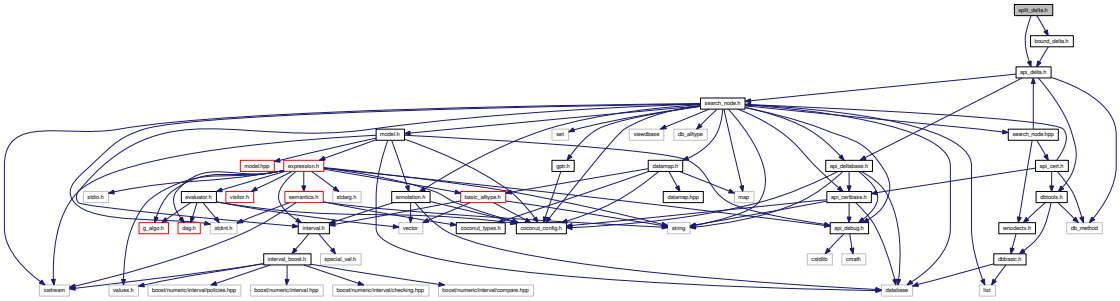
Definition in file sparsity.h.

11.141 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

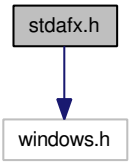
11.141.1 Detailed Description

Definition in file split_delta.h.

11.142 stdafx.h File Reference

```
#include <windows.h>
```

Include dependency graph for stdafx.h:



Defines

- #define WIN32_LEAN_AND_MEAN

11.142.1 Define Documentation

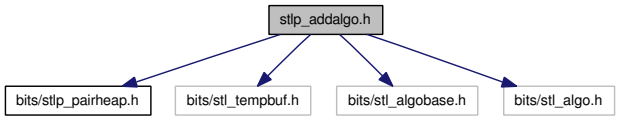
11.142.1.1 #define WIN32_LEAN_AND_MEAN

Definition at line 4 of file stdafx.h.

11.143 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)

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 (_RandomAccessIter __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 (_RandomAccessIter __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, _RandomAccessIterp __last2, _Tpp __val2)
- 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)
- template<typename _RandomAccessIter , typename _RandomAccessIterp >
  void std::__insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)
- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void std::__insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __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, _RandomAccessIterp __firstp, _RandomAccessIterp __lastp)

- template<typename _RandomAccessIter , typename _RandomAccessIterp , typename _Compare >
  void std::__final_insertion_p_sort (_RandomAccessIter __first, _RandomAccessIter __last, _RandomAccessIterp __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.

11.143.1 Detailed Description

Definition in file stlp_addalgo.h.

11.143.2 Define Documentation

11.143.2.1 #define __ISTLGLIBCXX_function_requires(...) __glibcpp_function_requires(__gnu_cxx::__VA_ARGS__)

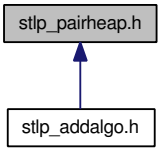
Definition at line 42 of file stlp_addalgo.h.

11.143.2.2 #define __ISTLGLIBCXX_requires_valid_range(A, B)

Definition at line 43 of file stlp_addalgo.h.

11.144 stlp_pairheap.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace std

  The standard namespace.

Defines

- #define __STLP_PAIRHEAP_H 1

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)

11.144.1 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.

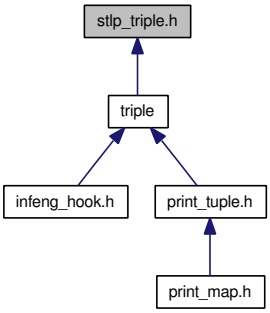
11.144.2 Define Documentation

11.144.2.1 #define __STLP_PAIRHEAP_H 1

Definition at line 38 of file stlp_pairheap.h.

11.145 stlp_triple.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct std::triple< _T1, _T2, _T3 >

  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.*

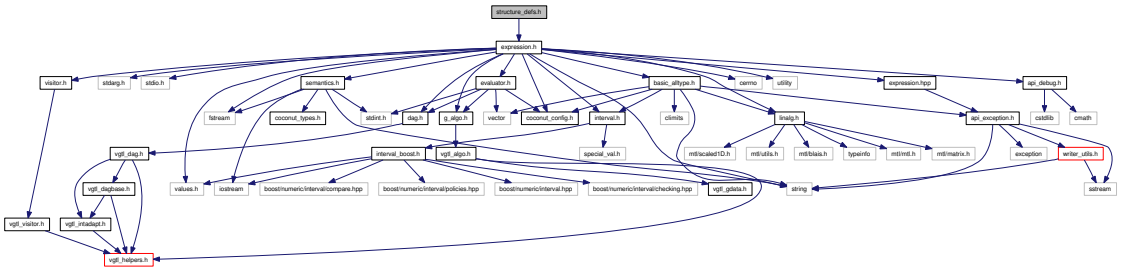
- template<class _T1 , class _T2 , class _T3 >
  triple< _T1, _T2, _T3 > std::make_triple (const _T1 &__x, const _T2 &__y, const _T3 &__z)

  A convenience wrapper for creating a triple from three objects.

11.146 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

11.146.1 Detailed Description

Definition in file structure_defs.h.

11.146.2 Define Documentation

11.146.2.1 #define CTR_CEQU 1

Definition at line 127 of file structure_defs.h.

11.146.2.2 #define CTR_CINEQ 2

Definition at line 129 of file structure_defs.h.

11.146.2.3 #define CTR_CONSTR 0

Definition at line 125 of file structure_defs.h.

11.146.2.4 #define CTR_EBOUND 3

Definition at line 131 of file structure_defs.h.

11.146.2.5 #define CTR_EDEG1 4

Definition at line 133 of file structure_defs.h.

11.146.2.6 #define CTR_EDEG2 5

Definition at line 135 of file structure_defs.h.

11.146.2.7 #define CTR_EDEG3 6

Definition at line 137 of file structure_defs.h.

11.146.2.8 #define CTR_EDGES 0

Definition at line 194 of file structure_defs.h.

11.146.2.9 #define CTR_EDIM1 9

Definition at line 143 of file structure_defs.h.

11.146.2.10 #define CTR_EDIM2 10

Definition at line 145 of file structure_defs.h.

11.146.2.11 #define CTR_EINTEGER 12

Definition at line 149 of file structure_defs.h.

11.146.2.12 #define CTR_EMTDEG3P 7

Definition at line 139 of file structure_defs.h.

11.146.2.13 #define CTR_EMTDIM2 11

Definition at line 147 of file structure_defs.h.

11.146.2.14 #define CTR_ENLN 8

Definition at line 141 of file structure_defs.h.

11.146.2.15 #define CTR_EXISTS 4

Definition at line 48 of file structure_defs.h.

11.146.2.16 #define CTR_EXPRINFO_USERDEFINED (EXPRINFO_NUMOFPREDEF+1)

Definition at line 211 of file structure_defs.h.

11.146.2.17 #define CTR_FORALL 5

Definition at line 50 of file structure_defs.h.

11.146.2.18 #define CTR_FREE 3

Definition at line 46 of file structure_defs.h.

11.146.2.19 #define CTR_IACTIVE 25

Definition at line 175 of file structure_defs.h.

11.146.2.20 #define CTR_IACTIVE_HI 29

Definition at line 183 of file structure_defs.h.

11.146.2.21 #define CTR_IACTIVE_LO 27

Definition at line 179 of file structure_defs.h.

11.146.2.22 #define CTR_IBOUND 13

Definition at line 151 of file structure_defs.h.

11.146.2.23 #define CTR_ICONCAVE 24

Definition at line 173 of file structure_defs.h.

11.146.2.24 #define CTR_ICONVEX 23

Definition at line 171 of file structure_defs.h.

11.146.2.25 #define CTR_IDEG1 14

Definition at line 153 of file structure_defs.h.

11.146.2.26 #define CTR_IDEG2 15

Definition at line 155 of file structure_defs.h.

11.146.2.27 #define CTR_IDEG3 16

Definition at line 157 of file structure_defs.h.

11.146.2.28 #define CTR_IDIM1 19

Definition at line 163 of file structure_defs.h.

11.146.2.29 #define CTR_IDIM2 20

Definition at line 165 of file structure_defs.h.

11.146.2.30 #define CTR_IINACTIVE 26

Definition at line 177 of file structure_defs.h.

11.146.2.31 #define CTR_IINACTIVE_HI 30

Definition at line 185 of file structure_defs.h.

11.146.2.32 #define CTR_IINACTIVE_LO 28

Definition at line 181 of file structure_defs.h.

11.146.2.33 #define CTR_IINTEGER 22

Definition at line 169 of file structure_defs.h.

11.146.2.34 #define CTR_IMTDEG3P 17

Definition at line 159 of file structure_defs.h.

11.146.2.35 #define CTR_IMTDIM2 21

Definition at line 167 of file structure_defs.h.

11.146.2.36 #define CTR_INLN 18

Definition at line 161 of file structure_defs.h.

11.146.2.37 #define CTR_INTEGER 1

Definition at line 42 of file structure_defs.h.

11.146.2.38 #define CTR_IREDUNDANT 31

Definition at line 187 of file structure_defs.h.

11.146.2.39 #define CTR_IRESTRICT 32

Definition at line 189 of file structure_defs.h.

11.146.2.40 #define CTR_KJNODES 2

Definition at line 44 of file structure_defs.h.

11.146.2.41 #define CTR_LEAVES 1

Definition at line 196 of file structure_defs.h.

11.146.2.42 #define CTR_MIN 19

Definition at line 120 of file structure_defs.h.

11.146.2.43 #define CTR_NODES 0

Definition at line 40 of file structure_defs.h.

11.146.2.44 #define CTR_O1BOUNDED 7

Definition at line 96 of file structure_defs.h.

11.146.2.45 #define CTR_O2BOUNDED 6

Definition at line 94 of file structure_defs.h.

11.146.2.46 #define CTR_OBJ 0

Definition at line 82 of file structure_defs.h.

11.146.2.47 #define CTR_OCONCAVE 18

Definition at line 118 of file structure_defs.h.

11.146.2.48 #define CTR_OCONVEX 17

Definition at line 116 of file structure_defs.h.

11.146.2.49 #define CTR_ODEG1 12

Definition at line 106 of file structure_defs.h.

11.146.2.50 #define CTR_ODEG2 13

Definition at line 108 of file structure_defs.h.

11.146.2.51 #define CTR_ODEG3 14

Definition at line 110 of file structure_defs.h.

11.146.2.52 #define CTR_ODIM1 9

Definition at line 100 of file structure_defs.h.

11.146.2.53 #define CTR_ODIM2 10

Definition at line 102 of file structure_defs.h.

11.146.2.54 #define CTR_OEXISTS 3

Definition at line 88 of file structure_defs.h.

11.146.2.55 #define CTR_OFORALL 4

Definition at line 90 of file structure_defs.h.

11.146.2.56 #define CTR_OFREE 2

Definition at line 86 of file structure_defs.h.

11.146.2.57 #define CTR_OINTEGER 1

Definition at line 84 of file structure_defs.h.

11.146.2.58 #define CTR_OMTDEG3P 15

Definition at line 112 of file structure_defs.h.

11.146.2.59 #define CTR_OMTDIM2 11

Definition at line 104 of file structure_defs.h.

11.146.2.60 #define CTR_ONLN 16

Definition at line 114 of file structure_defs.h.

11.146.2.61 #define CTR_OSTOCHASTIC 5

Definition at line 92 of file structure_defs.h.

11.146.2.62 #define CTR_OUBOUNDED 8

Definition at line 98 of file structure_defs.h.

11.146.2.63 #define CTR_ROOTS 2

Definition at line 198 of file structure_defs.h.

11.146.2.64 #define CTR_SDIFF 2

Definition at line 207 of file structure_defs.h.

11.146.2.65 #define CTR_SNNZH 1

Definition at line 205 of file structure_defs.h.

11.146.2.66 #define CTR_SNNZJ 0

Definition at line 203 of file structure_defs.h.

11.146.2.67 #define CTR_STOCHASTIC 6

Definition at line 52 of file structure_defs.h.

11.146.2.68 #define CTR_V1BOUNDED 9

Definition at line 75 of file structure_defs.h.

11.146.2.69 #define CTR_V2BOUNDED 8

Definition at line 73 of file structure_defs.h.

11.146.2.70 #define CTR_VARS 0

Definition at line 57 of file structure_defs.h.

11.146.2.71 #define CTR_VBINARY 1

Definition at line 59 of file structure_defs.h.

11.146.2.72 #define CTR_VEXISTS 4

Definition at line 65 of file structure_defs.h.

11.146.2.73 #define CTR_VFORALL 5

Definition at line 67 of file structure_defs.h.

11.146.2.74 #define CTR_VFREE 3

Definition at line 63 of file structure_defs.h.

11.146.2.75 #define CTR_VINTEGER 1

Definition at line 61 of file structure_defs.h.

11.146.2.76 #define CTR_VKJ 7

Definition at line 71 of file structure_defs.h.

11.146.2.77 #define CTR_VSTOCHASTIC 6

Definition at line 69 of file structure_defs.h.

11.146.2.78 #define CTR_VUBOUNDED 10

Definition at line 77 of file structure_defs.h.

11.146.2.79 #define NUM_CCTR 33

Definition at line 191 of file structure_defs.h.

11.146.2.80 #define NUM_CTR 7

Definition at line 54 of file structure_defs.h.

11.146.2.81 #define NUM_ECTR 3

Definition at line 200 of file structure_defs.h.

11.146.2.82 #define NUM_OCTR 20

Definition at line 122 of file structure_defs.h.

11.146.2.83 #define NUM_SCTR 3

Definition at line 209 of file structure_defs.h.

11.146.2.84 #define NUM_VCTR 11

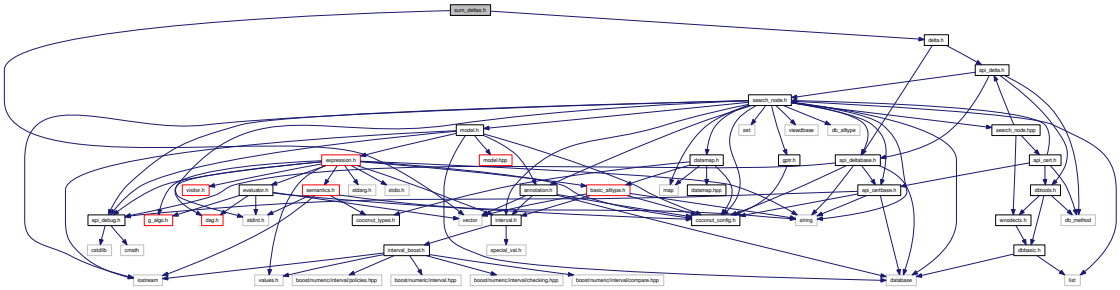
Definition at line 79 of file structure_defs.h.

11.147 sum_deltas.h File Reference

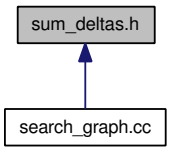
```
#include <vector>
```

```
#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.

11.147.1 Detailed Description

Definition in file sum_deltas.h.

11.148 table_delta.cc File Reference

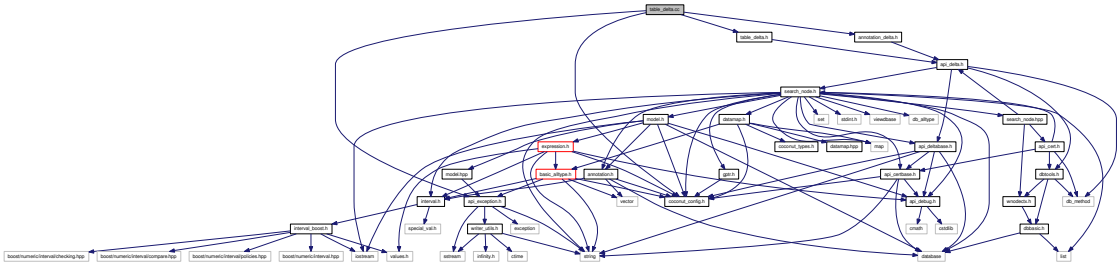
```
#include <coconut_config.h>
```

```
#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

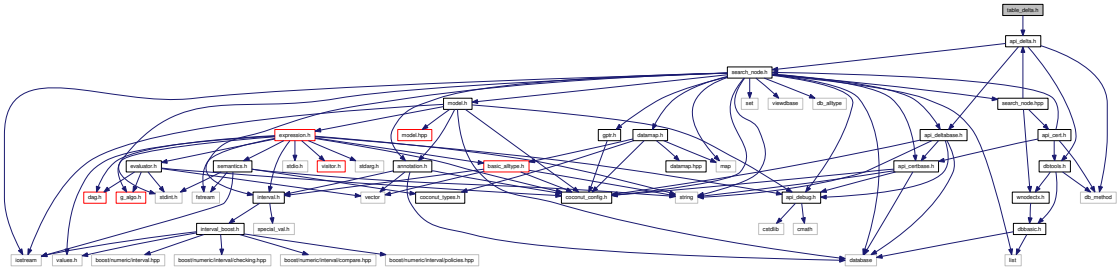
11.148.1 Detailed Description

Definition in file table_delta.cc.

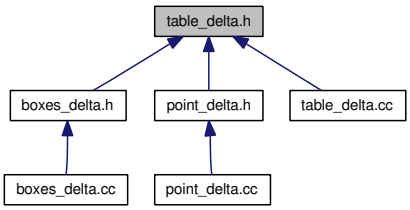
11.149 table_delta.h File Reference

```
#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

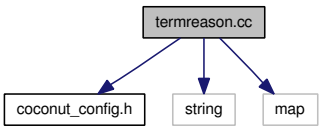
11.149.1 Detailed Description

Definition in file table_delta.h.

11.150 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.

11.150.1 Detailed Description

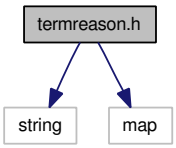
Definition in file termreason.cc.

11.151 termreason.h File Reference

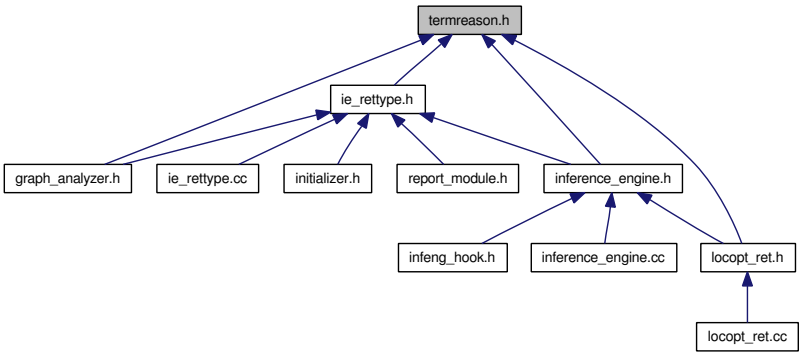
```
#include <string>
```

```
#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.

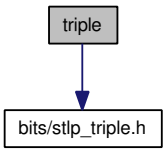
11.151.1 Detailed Description

Definition in file termreason.h.

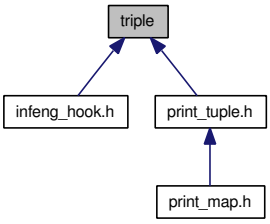
11.152 triple File Reference

```
#include <bits/stlp_triple.h>
```

Include dependency graph for triple:



This graph shows which files directly or indirectly include this file:



11.152.1 Detailed Description

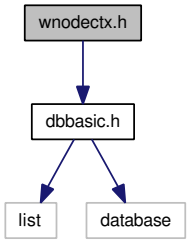
This is the external header file intended for direct use.

Definition in file triple.

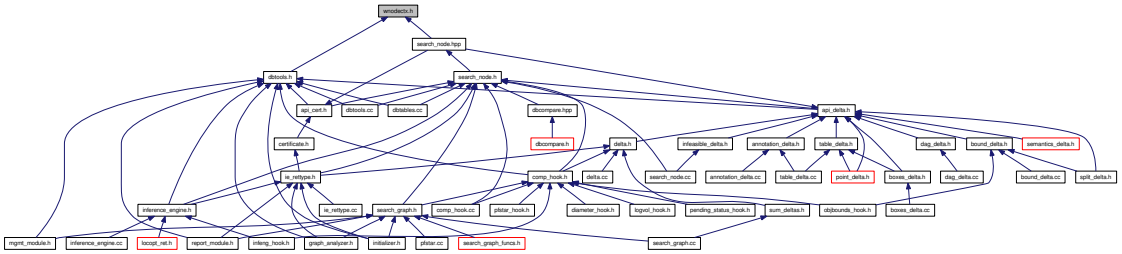
11.153 wnodectx.h File Reference

```
#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

11.153.1 Detailed Description

Definition in file wnodectx.h.

11.154 writer_utils.cc File Reference

```
#include <coconut_config.h>

#include <writer_utils.h>

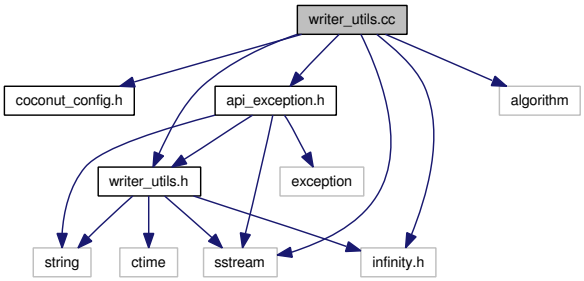
#include <sstream>

#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.

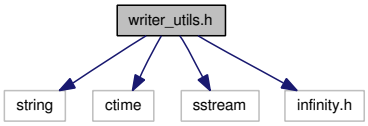
11.154.1 Detailed Description

Definition in file writer_utils.cc.

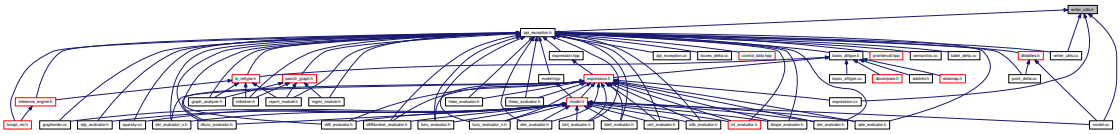
11.155 writer_utils.h File Reference

```
#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)

Functions

- template<class _TH >
  std::string 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.

11.155.1 Detailed Description

Definition in file writer_utils.h.

11.155.2 Define Documentation

11.155.2.1 #define coconut_sscanfdouble(D, S, F, ...) sscanf(S, F " %lf", ## __VA_ARGS__, D)

Definition at line 86 of file writer_utils.h.

11.155.2.2 #define double_out(d) (d)

Definition at line 85 of file writer_utils.h.