Typing a semantic memory for mathematical content

P. Schodl & A. Neumaier

LSFA 2011, August 27, 2011

Outline

(1) The MoSMath project

(2) The semantic memory

(3) The type system

(4) Applications

MoSMath

"A modeling system for mathematics" (MoSMath).

Goal: a modeling system for the specification of models for the numerical work.

Input: a controlled natural language, formulas in a subset of LaTeX.

Output: Description in LaTeX, model-file in AMPL, etc.

Advantages:

No need to learn an algebraic modeling language

Specification is the least error prone, and the most natural

We expect that the framework of the MoSMath project will serve as a first step towards the FMathL project.

`http://www.mat.univie.ac.at/~neum/FMathL.html`

The semantic memory

We want to be able to represent mathematical expressions and mathematical natural language.

We use a directed labeled graph, special case of a semantic network, implementable in the semantic web.

We assume an unrestricted set of **objects**.

Objects may, but need not have **names**.

Objects may, but need not have **external values**.

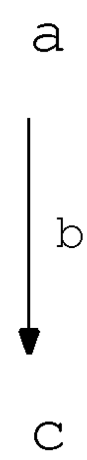
We refer to **unnamed objects** via a string beginning with a dollar-sign ($).

The **semantic memory** (abbreviated SM) stores equations of the form

`a.b=c`

These are called **sems**.

Graphical: a sem `a.b=c` as an edge with label `b` from node `a` to node `c`.



A sem `a.b=c` usually means: "the `b` of `a` is `c`".

In terms of the semantic web: "the `a` has property `b` valued by `c`".

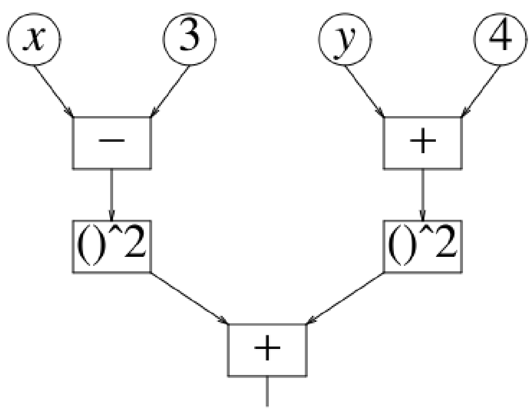
E.g., the sem `formula27.label=CauchySchwarz` would intuitively mean that the `label` of `formula27` is `CauchySchwarz`.

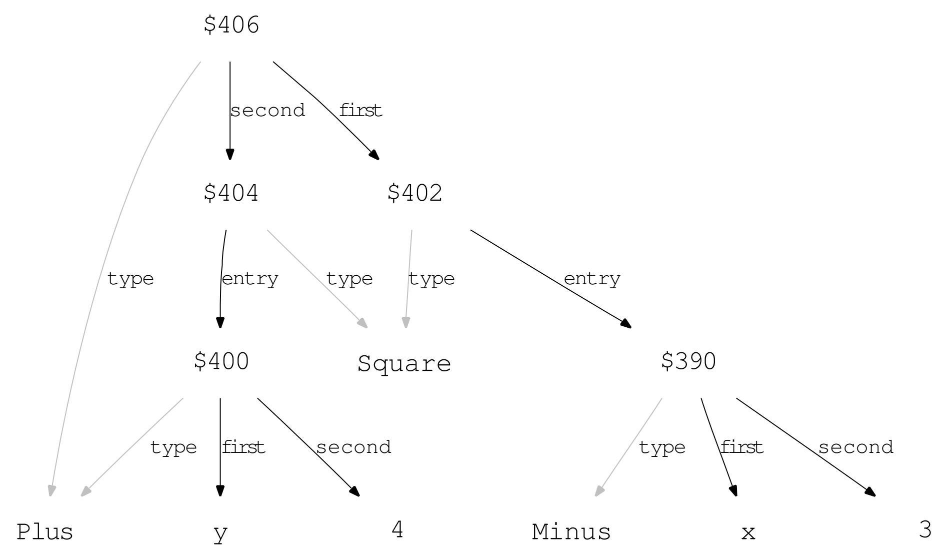
Only restriction: no two arcs beginning at the same node may have the same label.

In particular, cycles are allowed.

General idea for representing formulas: automatic differentiation.

$$(x-3)^2+(y+4)^2$$

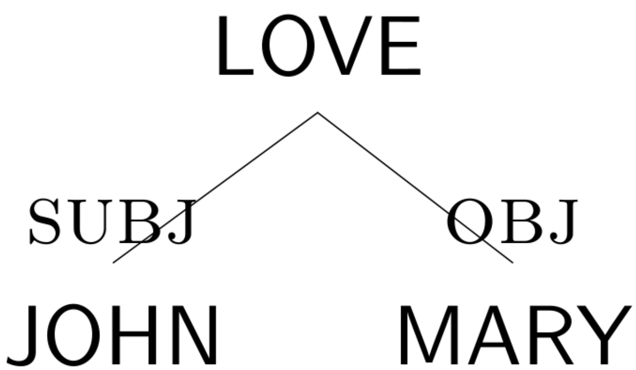


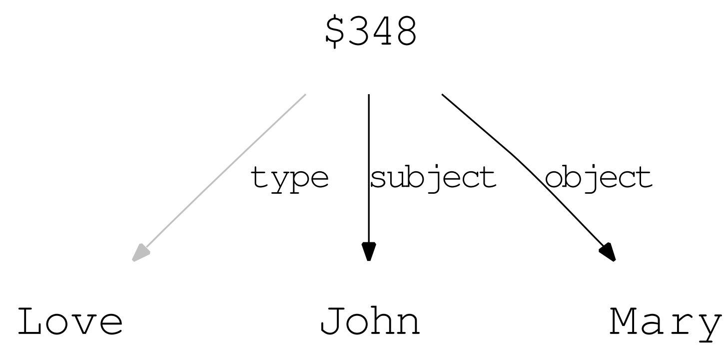


Source: D. Gay, Using Expression Graphs In Optimization Algorithms

General idea for representing natural language: dependency grammar.

"John loves Mary."





The semantic virtual machine:

A virtual machine that operates on the SM.

The algorithms for the semantic virtual machine are also graphs in the semantic memory.

Typing

An essential step that brings formal structure into the semantic memory is the introduction of **types**.

The set of sems reachable from object o are called the **record** with handle o.

We want to give a criterion when a record is “well-formed”.

The type system

Categories are objects, ordered by a transitive partial relation `<`. (`C1<C2` means `C1` is **contained** in `C2`).

A category is called a **type** or an **atomic** if it is minimal in the ordering `<`, and a **union** otherwise.

Atomics do not have any children, they have semantic meaning in itself, types pose requirements on records.

If `x.type=y` then the requirements of `y` apply to `x`.

Categories are defined in text documents called **type sheets**.

Example: we define types `BinaryRel` and `Integer`,
atomics `LessEq`, `Equal` and `Less`
and the union `RelationSym`.

```
RelationSym:
  atomic> LessEq, Equal, Less

BinaryRel:
  allOf> lhs = Integer
         rhs = Integer
         relation = RelationSym

Integer:
  nothingElse
```

All information about the categories is represented in the semantic memory
$\rightarrow$ type checking is an algorithm on the semantic memory.

operator	usage
allOf	requirements to all of a set of fields
oneOf	requirements to exactly of a set of fields
someOf	requirements to at least one of a set of fields
optional	requirements to a set of fields, if nonempty
fixed	requires not types but certain objects
someOfType	requirements to all fields of a type
itself	requires a set of fields which are also the entries
nothingElse	forbids non-required fields
nothing	defines an atomic type
union	defines a union
complete	closes a union

The type definitions are themselves typed, i.e., the creation of a type sheet that allows to type the representation of types in the semantic matrix, which gives the type of types.

The type sheet for types has only 40 lines.

Meta-schema for RelaxNG schema has over 300 lines, for Meta-DTD over 700 lines.

Application 1: The OR Library

A significant fraction of the OR Library was represented manually.

Algorithms produce LaTeX and AMPL.

Example: multi-dimensional knapsack problem.

Let the integer N be the number of contracts, let the integer M be the number of budgets. Let c_j be the contract volume of project j for $j = 1, \dots, N$, let $A_{i,j}$ be the estimated cost of budget i for project j for $i = 1, \dots, M$ and $j = 1, \dots, N$, and let B_i be the available amount of budget i for $i = 1, \dots, M$. For $j = 1, \dots, N$, let $x_j = 1$ if project j is selected, and let $x_j = 0$ otherwise.

Problem : Given integers N and M, vector c, matrix A and vector B, find the binary vector x such that

$$\sum_{j=1}^{N} c_j \, x_j$$

is maximal under the constraint $\sum_{j=1}^{N} A_{i,j} \, x_j \leq B_i$ for $i = 1, \dots, M$.

Represented in the semantic memory using 449 sems.

Automatically generated AMPL-output:

```
param N ;
param M ;
param c{j in 1..N} ;
param A{i in 1..M , j in 1..N} ;
param B{i in 1..M} ;
var x{j in 1..N} binary ;

maximize target :  sum{j in 1..N}(c[j] * x[j]);
subject to constraint_1{i in 1..M} :  sum{j in 1..N}(A[i ,
j] * x[j]) <= B[i];
```

Application 2: A collection of formulas

- mathematical formulas were extracted from lecture notes
- manually fed into the semantic memory
- automatic LaTeX-output

Application 3: The TPTP Library

- Automatically import formulas from the TPTP library ("Thousands of Problems for Theorem Provers").
- formulas from different branches of mathematics
- more than 10.000 problem files.

Thank you for your attention!

http://www.mat.univie.ac.at/~neum/FMathL.html