Formal Languages for Mathematics

Jonas Betzendahl
jonas.betzendahl@fau.de

2018 – 08 – 13

Motivation (1)

The topic of developing languages for formalising mathematics has gathered a lot of attention over the last few years. A few observations:

Motivation (1)

The topic of developing languages for formalising mathematics has gathered a lot of attention over the last few years. A few observations:

- Most formalisms fail to capture the flexible nature of in-the-wild chalk-and-blackboard mathematics.

Motivation (1)

The topic of developing languages for formalising mathematics has gathered a lot of attention over the last few years. A few observations:

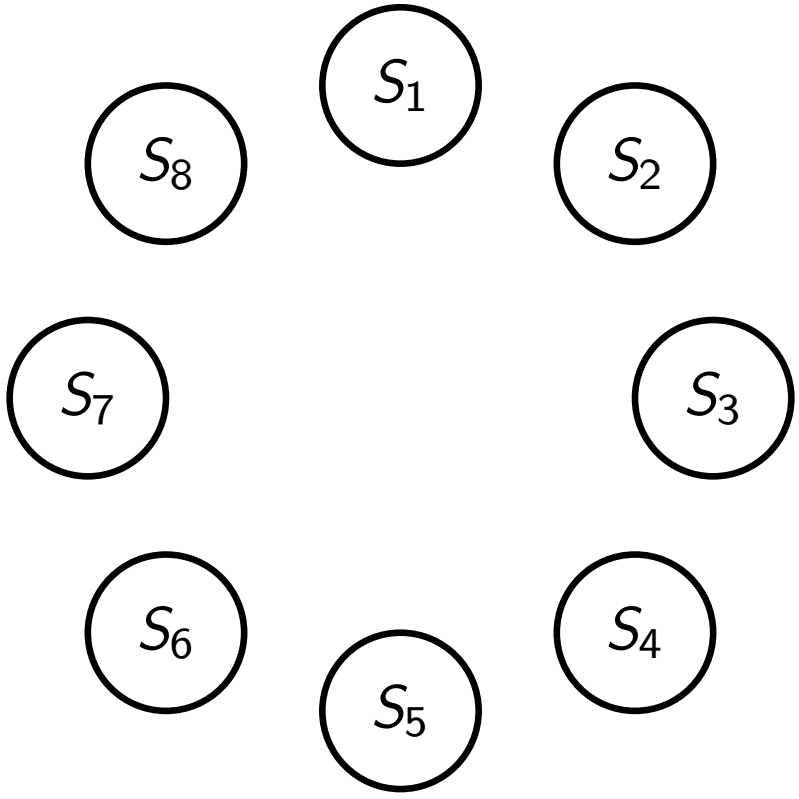
- Most formalisms fail to capture the flexible nature of in-the-wild chalk-and-blackboard mathematics.
- Type systems are ubiquitous, but often lack key features to keep them decidable

Motivation (1)

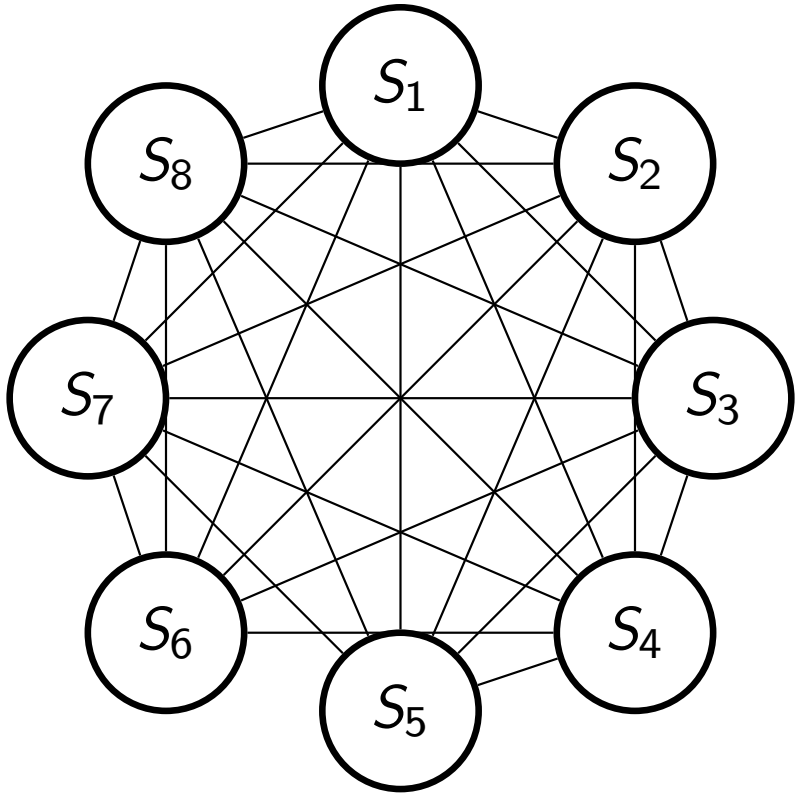
The topic of developing languages for formalising mathematics has gathered a lot of attention over the last few years. A few observations:

- Most formalisms fail to capture the flexible nature of in-the-wild chalk-and-blackboard mathematics.
- Type systems are ubiquitous, but often lack key features to keep them decidable
- Many systems are also either flexible *or* have good tool, support, not both.

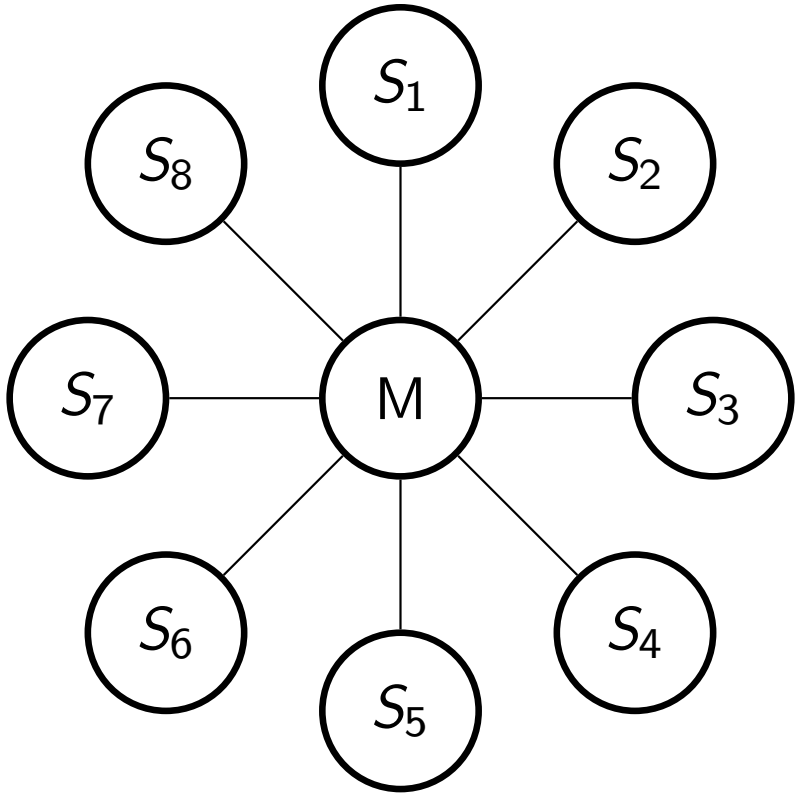
Motivation (2)



Motivation (2)



Motivation (2)



Context & Preliminaries

IMPS (1)

IMPS (short for "Interactive Mathematical Proof System") is an interactive theorem prover developed by William Farmer, Joshua Guttmann and Javier Thayer from 1990 to 1993. It was one of the influential systems in the era of automated reasoning.

IMPS (1)

IMPS (short for "Interactive Mathematical Proof System") is an interactive theorem prover developed by William Farmer, Joshua Guttmann and Javier Thayer from 1990 to 1993. It was one of the influential systems in the era of automated reasoning.

One of the goals in developing IMPS was to create a mathematical system that gave computational support to mathematical techniques common among actual mathematicians.

IMPS (2)

IMPS pioneered mechanisations of many interesting features for reasoning systems, such as:

IMPS (2)

IMPS pioneered mechanisations of many interesting features for reasoning systems, such as:

- The *small theories* approach to axiomatic mathematics (which gives rise to *theory graphs*)

IMPS (2)

IMPS pioneered mechanisations of many interesting features for reasoning systems, such as:

- The *small theories* approach to axiomatic mathematics (which gives rise to *theory graphs*)
- A rigorous treatment of undefinedness (Partial valuation for terms, total valuation for formulas)

IMPS (2)

IMPS pioneered mechanisations of many interesting features for reasoning systems, such as:

- The *small theories* approach to axiomatic mathematics (which gives rise to *theory graphs*)
- A rigorous treatment of undefinedness (Partial valuation for terms, total valuation for formulas)
- Partial functions, subsorts, definite description operator

OMDoc

OMDoc (short for **O**pen **M**athematical **D**ocuments) is a semantics-oriented markup format for STEM-related documents extending OpenMath.

OMDoc

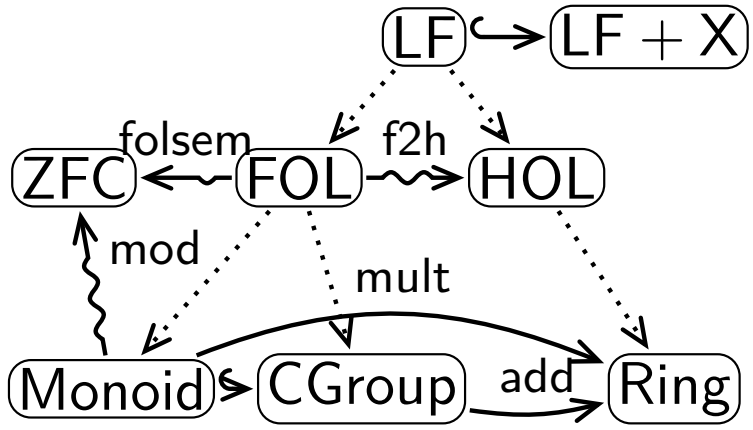
OMDoc (short for **O**pen **M**athematical **D**ocuments) is a semantics-oriented markup format for STEM-related documents extending OpenMath.

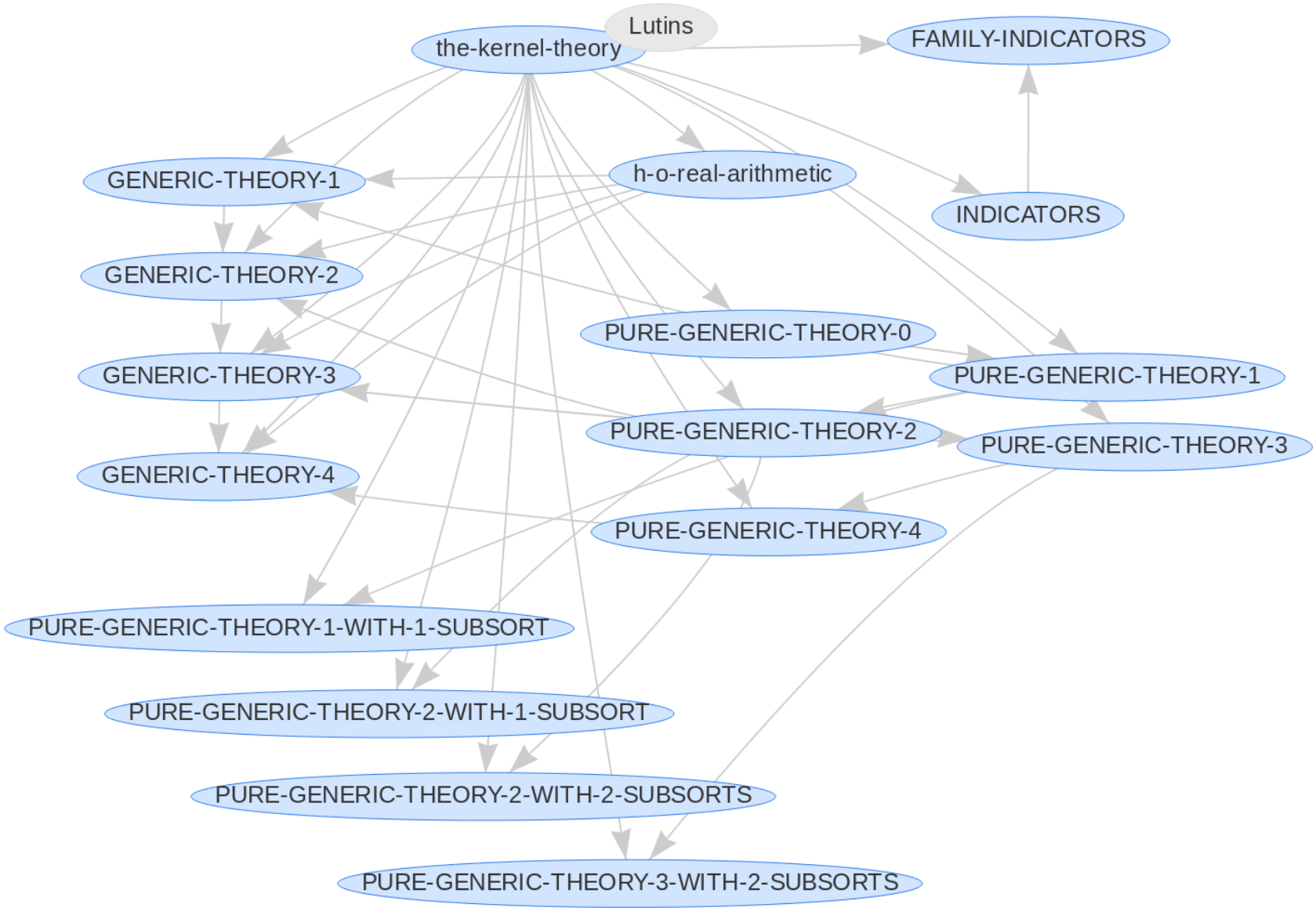
OMDoc/MMT brings with it three distinct levels for expression of (both formal and informal) mathematical knowledge, structurally similar to `IMPS`:

- **Object Level**
  Expressions (e.g. terms and formulae) expressed in OpenMath.
- **Declaration Level**
  Constants (functions, types, judgements) with an optional (object-level) type and/or definition.
- **Module Level**
  Theories and Views; sets of declarations that inhabit a common namespace and context.

MMT

The OMDoc/MMT language is used by the MMT system, which provides an API to handle OMDoc/MMT content and services such as type checking, rewriting of expressions and computation, as well as notation-based presentation of OMDoc/MMT content and a general infrastructure for inspecting and browsing libraries.




Lutins
the-kernel-theory
FAMILY-INDICATORS
GENERIC-THEORY-1
h-o-real-arithmetic
INDICATORS
GENERIC-THEORY-2
PURE-GENERIC-THEORY-0
GENERIC-THEORY-3
PURE-GENERIC-THEORY-1
PURE-GENERIC-THEORY-2
PURE-GENERIC-THEORY-3
GENERIC-THEORY-4
PURE-GENERIC-THEORY-4
PURE-GENERIC-THEORY-1-WITH-1-SUBSORT
PURE-GENERIC-THEORY-2-WITH-1-SUBSORT
PURE-GENERIC-THEORY-2-WITH-2-SUBSORTS
PURE-GENERIC-THEORY-3-WITH-2-SUBSORTS

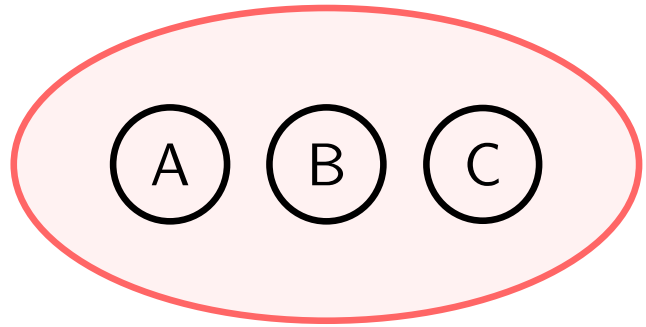
Goals

Goals (1)

Further the shared formalisation for multiple mathematical systems.

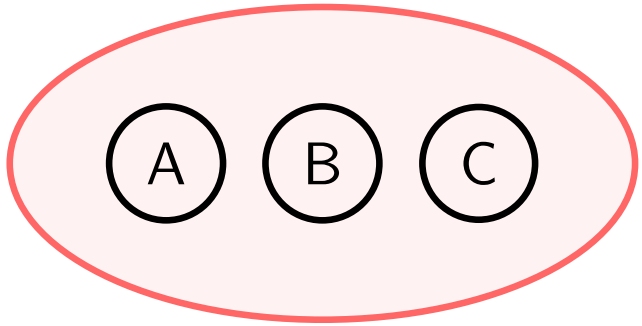
Goals (1)

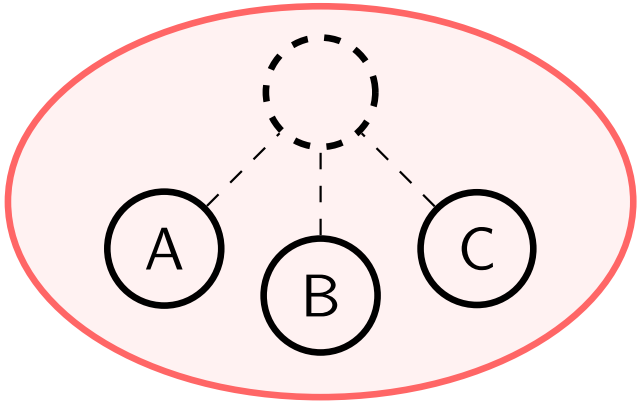
Further the shared formalisation for multiple mathematical systems.



Goals (1)

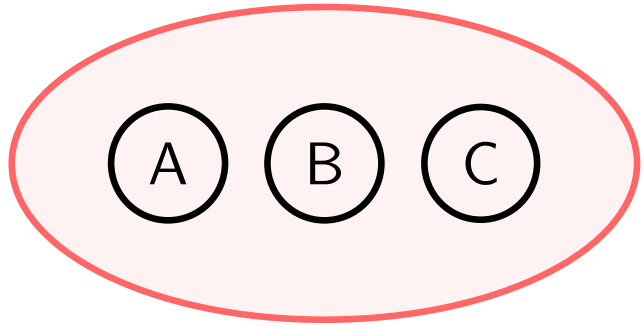
Further the shared formalisation for multiple mathematical systems.

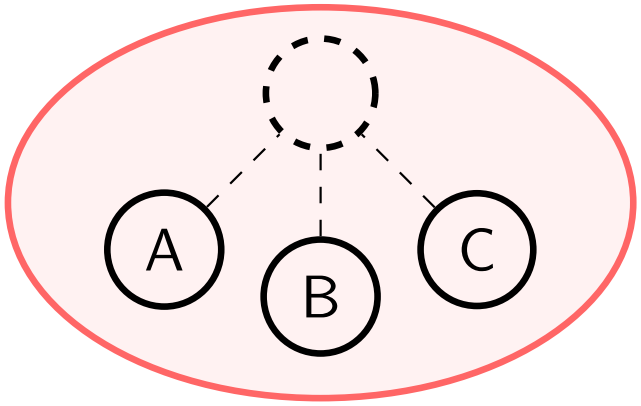


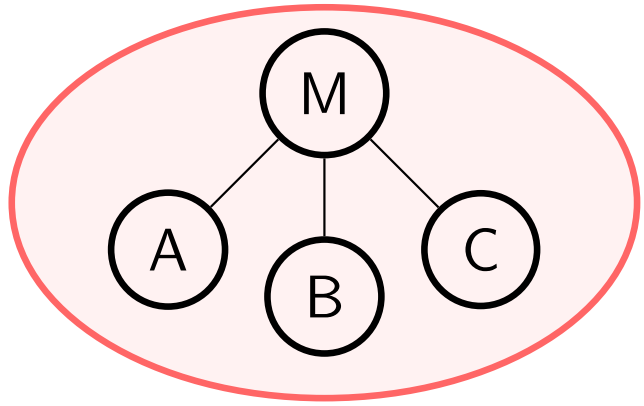


Goals (1)

Further the shared formalisation for multiple mathematical systems.







Goals (2)

Finding a treatment of undefinedness that integrates well the MMT system.

Goals (2)

Finding a treatment of undefinedness that integrates well the MMT system.

Partial valuation for terms, total valuation for formulas and the divide into two concrete kinds of all mathematical expressions seem a bit inelegant.

Goals (3)

A proof system similar to that of IMPS, for MMT. Probably a tall order for the time frame of one PhD.

Goals (3)

A proof system similar to that of IMPS, for MMT. Probably a tall order for the time frame of one PhD.

Focus for now: automatic handling / solving of small proof obligations in the context of type checking.

Goals (3)

A proof system similar to that of IMPS, for MMT. Probably a tall order for the time frame of one PhD.

Focus for now: automatic handling / solving of small proof obligations in the context of type checking.

$$\Gamma \vdash t = t' \qquad \Gamma \vdash t : A$$

$$\Gamma \vdash t \downarrow \qquad \Gamma \vdash ? : A$$

Efforts
(planned)

Efforts (1)

Survey of related systems:
(with focus on treatment of undefined values and possibly small proof obligations)

Efforts (1)

Survey of related systems:
(with focus on treatment of undefined values and possibly small proof obligations)

- Logical Frameworks
  (e.g. **LF**, Isabelle) *Typically thin tool support, narrow focus.*

Efforts (1)

Survey of related systems:
(with focus on treatment of undefined values and possibly small proof obligations)

- Logical Frameworks
  (e.g. **LF**, Isabelle) *Typically thin tool support, narrow focus.*
- Proof Assistants
  (e.g. Coq, Agda, HOL Light) *Mostly limited to decidable TT.*

Efforts (1)

Survey of related systems:
(with focus on treatment of undefined values and possibly small proof obligations)

- Logical Frameworks
  (e.g. **LF**, Isabelle) *Typically thin tool support, narrow focus.*
- Proof Assistants
  (e.g. Coq, Agda, HOL Light) *Mostly limited to decidable TT.*
- Fully Automated Provers
  (e.g. E, Vampire) *Usually limited to first-order logic.*

Efforts (1)

Survey of related systems:
(with focus on treatment of undefined values and possibly small proof obligations)

- Logical Frameworks
  (e.g. **LF**, Isabelle) *Typically thin tool support, narrow focus.*
- Proof Assistants
  (e.g. Coq, Agda, HOL Light) *Mostly limited to decidable TT.*
- Fully Automated Provers
  (e.g. E, Vampire) *Usually limited to first-order logic.*
- Computation Systems
  (e.g. GAP, Sage) *Insufficient treatment of undefinedness.*

Efforts (2)

Import/Export (to and from MMT) for HOL Light.

Efforts (3)

Work towards mechanised dismissal of tiny proof obligations during type-checking.

Efforts (3)

Work towards mechanised dismissal of tiny proof obligations during type-checking.

Concretely: MMT as of today has support for annotating declarations of equalities so that the simplifier is able to use them as additional rules at runtime.

This process needs to be extended to also allow the introduction of rules with *premises* and the automated checking of the same.

Efforts (4)

Develop a softly typed set theory (i.e. types are added only after the fact via unary predicates) in MMT.

Efforts (4)

Develop a softly typed set theory (i.e. types are added only after the fact via unary predicates) in MMT.

Working in this context brings up a lot of the tiny proof obligations mentioned earlier and hence could serve as a testing environment.

Efforts (4)

Develop a softly typed set theory (i.e. types are added only after the fact via unary predicates) in MMT.

Working in this context brings up a lot of the tiny proof obligations mentioned earlier and hence could serve as a testing environment.

It also might lead to some insights if an approach like this, with the flexibility it brings to the table, is suitable for formalisation.