Rapid Prototyping Formal Systems in MMT: 5 Case Studies

Dennis Müller and Florian Rabe

Computer Science, University Erlangen-Nürnberg, Germany
LRI, University Paris-Sud, France

June 2019

Motivation

Logical Frameworks

= meta-logic in which syntax and semantics of object logics are defined — Automath, LF, Isabelle

Advantages

- Universal concepts — expressions, substitution, typing, equality, ...
- Meta-reasoning — consistency, logic translations, ...
- Rapid prototyping — type reconstruction, theorem proving, ...
- Generic tools — theorem prover, module system, IDE, ...

Simplicity vs. expressivity

- Meta-logic must be simple to be scalable, trustworthy
- Object logic must be expressive to be practical
- Big challenge for frameworks

Designing Logical Frameworks

Typical approach:

- choose a λ-calculus
- add other features
  - logic programming (λ-Prolog)
  - meta logic (Twelf, Abella)
  - proof assistant for object logic (Isabelle)
  - concurrency (CLF)
  - reasoning about contexts (Beluga)
  - rewriting (Dedukti)
  - external side conditions (LLFP)
  - coupling with proof-assistant support (Hybrid)
  - user-defined unification hints (ELPI)
  - ...

Problems

- Divergence due to choice of other features
- Even hypothetical union not expressive enough for real-life logics

no way to define, e.g., HOL Light, Mizar, PVS

Experimentation with Formal Systems

Customize the system fundamentals

- increasingly complex problem domains
  e.g., mathematics, programming languages
- plain formalization introduces too many artifacts to be human-readable
- therefore: allow users to define how to interpret human input
  e.g., custom parsing, type reconstruction

Examples:

- unification hints (Coq, Matita)
  - extra-logical declarations
  - allow users to guide incomplete algorithms (e.g., unification)
- meta-programming (Idris, Lean)
  - expose internal datatypes to user
  - allow users to program extensions in the language itself

MMT = Meta-Meta-Theory/Tool

Problem:

- logical frameworks not expressive for practical logics
- more system experimentation needed
- trend towards fine-grained user control

Foundation-independence: use logical frameworks without committing to a specific one

Mathematics	Logic	Logical Frameworks	Foundation-Independence
			MMT
		logical frameworks	
	logic, programming language, . . .		
domain knowledge			

The UniFormal Library

Large Scale Example: The LATIN Atlas

- DFG project 2009-2012 (with DFKI Bremen and Jacobs Univ.)
- Highly modular network of little logic formalizations
  - separate theory for each
    - connective/quantifier
    - type operator
    - controversial axioms — e.g., excluded middle, choice, . . .
    - base type
  - reference catalog of standardized logics
  - documentation platform
- Written in MMT/LF
- 4 years, with $\sim$ 10 students, $\sim$ 1000 modules

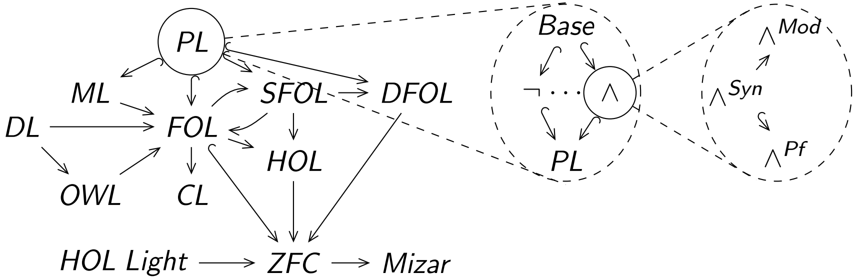
The LATIN Atlas of Logical Systems

The LATIN Atlas is huge: That's me pointing at the theory for first-order logic

Logic Diagrams in LATIN

An example fragment of the LATIN logic diagram

- nodes: MMT/LF theories
- edges: MMT/LF theory morphisms

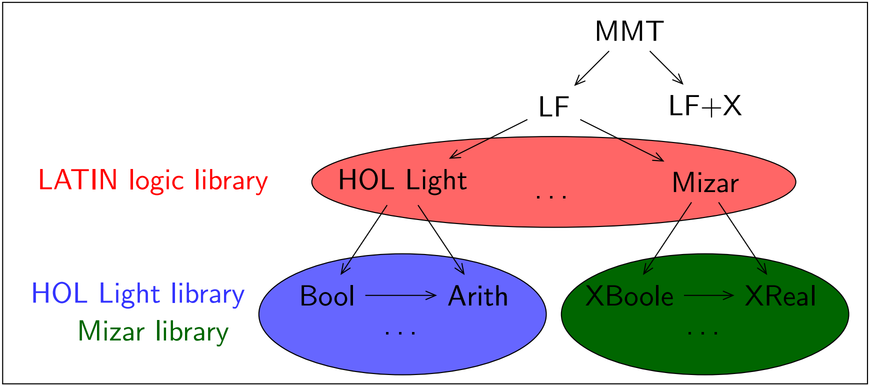


- each node is root for library of that logic
- each edge yields library translation functor

library *integration* very difficult though

OAF: Integration of Proof Assistant Libraries

- DFG project, 2014–2020, 15 contributors
- Big, overlapping libraries joined in MMT as the uniform representation language > 100 GB XML in total
  Mizar, HOL systems, IMPS, Coq, PVS, Isabelle...
- enables archival, comparison, integration



OpenDreamKit: Virtual Math Research Environments

- EU project, 2015-2019, 15 sites, 25 partners
  http://opendreamkit.org/
- MMT as mediator system
  - system-independent formalization of math > 200 theories
    no proofs, no algorithms
  - integration of math computation systems
    SageMath, GAP, Singular: services interfaces defined in MMT
  - ...and math databases
    LMFDB, OEIS: database schemas defined in MMT

Example: dynamic retrieval

- SageMath user needs 13th transitive group with conductor 5
- SageMath queries MMT
- MMT retrieves it from LMFDB, translates it to SageMath syntax

MathHub

GitHub-like but for MMT projects https://gl.mathhub.info

- 251 Repositories
- 187 Users
- 28.5 GB in March, probably doubled by now

For example:

Language	Library	Modules	Declarations
MMT	Math-in-the-Middle	220	826
LF	LATIN	529	2,824
PVS	Prelude+NASA	974	24,084
Isabelle	Distribution+AFP	9553	1,472,280
HOL Light	Basic	189	22,830
Coq	$>$ 50 in total	1,979	167,797
Mizar	MML	1,194	69,710
SageMath	Distribution	1,399	
GAP	Library		9,050

MMT Overview

Basic Concepts

Design principle

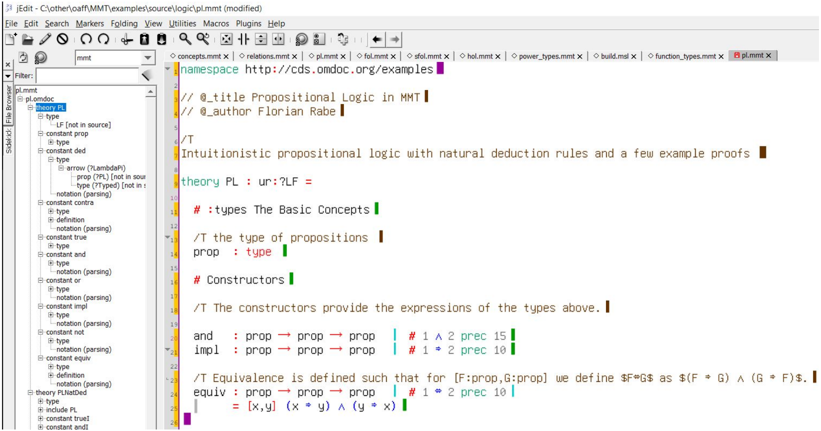
- few orthogonal concepts
- uniform representations of diverse languages

sweet spot in the expressivity-simplicity trade off

Concepts

- theory = named set of declarations
  - foundations, logics, type theories, classes, specifications, ...
- theory morphism = compositional translation
  - inclusions, translations, models, katamorphisms, ...
- constant = named atomic declaration
  - function symbols, theorems, rules, ...
  - may have type, definition, notation
- term = unnamed complex entity, formed from constants
  - expressions, types, formulas, proofs, ...
- typing $\vdash_T s : t$ between terms relative to a theory
  - well-formedness, truth, consequence ...

Example: Propositional Logic in the MMT IDE



Small Scale Example (1)

Logical frameworks in MMT

```
theory LF {
    type
    Pi        # Π V1 . 2                 name[:type][#notation]
    arrow     # 1 → 2
    lambda    # λ V1 . 2
    apply     # 1 2
}
```

Logics in MMT/LF

```
theory Logic: LF {
    prop : type
    ded  : prop → type # ⊢ 1            judgments-as-types
}
theory FOL: LF {
   include Logic
   term        : type                   higher-order abstract syntax
   forall      : (term → prop) → prop #  ∀ V1 . 2
}
```

Small Scale Example (2)

FOL from previous slide:

```
theory FOL: LF {
  include Logic
  term         : type
  forall       : (term → prop) → prop  #  ∀ V1 . 2
}
```

Proof-theoretical semantics of FOL

```
theory FOLPF: LF {
  include FOL
                                                    rules are constants
  forallIntro  : ΠF:term→prop.
                    (Πx:term.⊢ (F x)) → ⊢ ∀(λx:term.F x)
  forallElim   : ΠF:term→prop.
                    ⊢ ∀(λx:term.F x) → Πx:term.⊢ (F x)
}
```

Small Scale Example (3)

FOL from previous slide:

```
theory FOL: LF {
  include Logic
  term       : type
  forall     : (term → prop) → prop  #  ∀ V1 . 2
}
```

Algebraic theories in MMT/LF/FOL:

```
theory Magma : FOL {
  comp : term → term → term # 1 ∘ 2
}
theory SemiGroup          : FOL {include Magma, ...}
theory CommutativeGroup : FOL {include SemiGroup, ...}
theory Ring : FOL {
  additive: CommutativeGroup
  multiplicative: Semigroup
  ...
}
```

Abstract Syntax of Terms

Key ideas

- no predefined constants
- single general syntax tree constructor $c(\Gamma; \vec{E})$
- $c(\Gamma; \vec{E})$ binds variables and takes arguments
  - non-binding operators: Γ empty   e.g., $\mathtt{apply}(\cdot; f, a)$ for $(f\ a)$
  - typical binders: Γ and $\vec{E}$ have length 1
    e.g., $\mathtt{lambda}(x:A; t)$ for $\lambda x:A.t$

contexts	Γ	$::=$	$(x[:E][=E])^*$
terms	E	$::=$	
constants		$\mid$	c
variables		$\mid$	x
complex terms		$\mid$	$c(\Gamma; E^*)$

Terms are relative to theory T that declares the constants c

Concrete Syntax of Terms

- Theories may attach notation(s) to each constant declaration
- Notations of c introduce concrete syntax for $c(\Gamma; \vec{E})$

e.g., for type theory

concrete syntax	constant declaration	abstract syntax
E ::=		
`type`	`type` #	`type`
$\Pi x : E_1 . E_2$	`Pi` # Π V1 . 2	`Pi`$(x : E_1; E_2)$
$E_1 \to E_2$	`arrow` # 1 → 2	`arrow`$(\cdot; E_1, E_2)$
$\lambda x : E_1 . E_2$	`lambda` # λ V1 . 2	`lambda`$(x : E_1; E_2)$
$E_1\ E_2$	`apply` # 1 2	`apply`$(\cdot; E_1, E_2)$

Judgments

- MMT terms subsume terms of specific languages
- Type systems singles out the well-typed terms

For any theory Σ:

$\vdash \Sigma$	$T = \{\Sigma\}$ is a valid theory definition
$\vdash_T \Gamma$	Γ is a valid context
$\Gamma \vdash_T t : A$	t has type A
$\Gamma \vdash_T E = E'$	E and E' are equal
$\Gamma \vdash_T \_ : A$	A is inhabitable

Two kinds of rules:

- MMT defines some global rules once and for all
  foundation-independent rules
- declared in MMT theories, subject to scoping
  foundation-specific rules

Foundation-Independent Rules

- Lookup rules for atomic terms over a theory $T = \{\Sigma\}$

$$\frac{c : A \text{ in } \Sigma}{\vdash_T c : A} \qquad \frac{c = t \text{ in } \Sigma}{\vdash_T c = t}$$

- Equivalence and congruence rules for equality
- Rules for well-formed theories/contexts

$$\frac{}{\vdash \cdot} \qquad \frac{\vdash \Sigma \quad [\vdash_\Sigma \_ : A] \quad [\vdash_T t : A]}{\vdash \Sigma,\ c[: A][= t]}$$

Foundation-Specific Rules

- Declared in theories as constants
- Carry reference to self-contained Scala object implementing a rule interface

$\sim$ 10 rule interfaces, in particular one for each algorithm:

- simplification: $\Gamma \vdash_T E =?$
- equality checking: $\Gamma \vdash_T E = E'$?
- type inference: $\Gamma \vdash_T t :?$
- type checking: $\Gamma \vdash_T t : A$?
- proving: $\Gamma \vdash_T ? : A$ experimental

E.g., λ-inference rule

- applicable to $\Gamma \vdash_T t :?$ whenever $t = \lambda x : A.s$
- recursively infers type of s, returns Π-type
- reports errors and trace messages

MMT Tool

Mature implementation

- API for representation language — foundation-independent
- Collection of reusable algorithms — no commitment to particular application
- Extensible wherever reasonable — storage backends, file formats, user interfaces, . . . operators and rules, language features, checkers, . . .

Separation of concerns between

- Foundation developers — e.g., language primitives, rules
- Service developers — e.g., search, theorem prover
- Application developers — e.g., IDE, proof assistant

Yields rapid prototyping for logic systems

MMT Tool

Mature implementation

- API for representation language — foundation-independent
- Collection of reusable algorithms — no commitment to particular application
- Extensible wherever reasonable — storage backends, file formats, user interfaces, ... operators and rules, language features, checkers, ...

Separation of concerns between

- Foundation developers — e.g., language primitives, rules
- Service developers — e.g., search, theorem prover
- Application developers — e.g., IDE, proof assistant

Yields rapid prototyping for logic systems

But how much can really be done foundation-independently?

MMT shows: not everything, but a lot

Logical Results

- Module system
  modularity transparent to foundation developer
- Concrete/abstract syntax
  notation-based parsing/presentation
- Interpreted symbols, literals
  external model/implementation reflected into MMT
- Type reconstruction
  foundation plugin supplies only core rules
- Simplification
  rule-based, integrated with type reconstruction

- Theorem proving?
- Code generation? Computation?

Knowledge Management Results

- Change management — recheck only if affected
- Project management — indexing, building
- Extensible export infrastructure — Scala, SVG graphs, LaTeX, HTML, ...
- Search, querying — substitution-tree and relational index
- Browser — interactive web browser, 2D/3D theory graph viewer
- Editing — IDE-like graphical interface, LaTeX integration

Type Reconstruction

Algorithm

- MMT implements foundation-independent rules
- visible foundation-specific rules collected from current context
- algorithm delegates to foundation-specific rules as needed

General algorithm takes care of

- unknown meta-variables
- delaying constraints
- definition expansion
- module system

transparent to foundation-specific rules

Case Studies

Rewriting

Add rewriting to any language already defined in MMT

Inspiration: LF modulo (e.g., Dedukti)

- Key idea: annotate
  - a binary judgment as a rewrite predicate
  - axioms for that judgment as rewrite rules
- Implement MMT plugin that dynamically generates new simplification rules for each annotated axioms
  - implemented via Change Listener interface
  - Termination of rewrite system remain user's responsibility
- Needed work: a few 100 hundred loc for the rule generator

External Side Conditions

Monadic type operator for calling external side conditions

$LLF_{\mathcal{P}}$ paper by Honsell, Liquori, Maksimovic, Scagnetto

- Special expressions that represent keys
  - normal MMT expressions
  - but not part of the type system
- New rules for keys that run external side condition
- Variables typed by keys indicate which locks can be opened
  - declared wheneve traversing into the monad
  - automatically ignored by all other typing rules
- Add new rules for lock types that look for keys in the context
  - if found, monad can be inspected
  - otherwise, discharge external side condition by calling rule for key
- Needed work: $\sim$ 100 loc for 7 rules

One evening at Dagstuhl with Ivan

Linear Logic

Inspiration: resource semantics to represent linear logic in LF-like structural framework MMT context lookup obeys structural rules

- Already done in LF
  - Kripke style model, worlds represent available resources
  - monoid of worlds to represent empty world, union
  - additional laws represent structural rules

    e.g., commutativity for exchange
- Problem in LF: requires explicit reasoning in the monoid

  awkward, inadequate
- Solution: add new rules for equality reasoning
- Needed work: depends on desired reasoning strength, < 100 loc for simple version

Conclusion

Summary

- MMT: foundation-independent framework for declarative languages
  - representation language
  - implementation
- Easy to instantiate with specific foundations

  rapid prototyping logic systems
- Deep foundation-independent results
  - logical: parsing, type reconstruction, module system, ...
  - knowledge management: search, browsers, IDE, ...
- Serious contender for
  - experimenting with new system ideas
  - generic applications/services
  - universal library
  - system integration platform

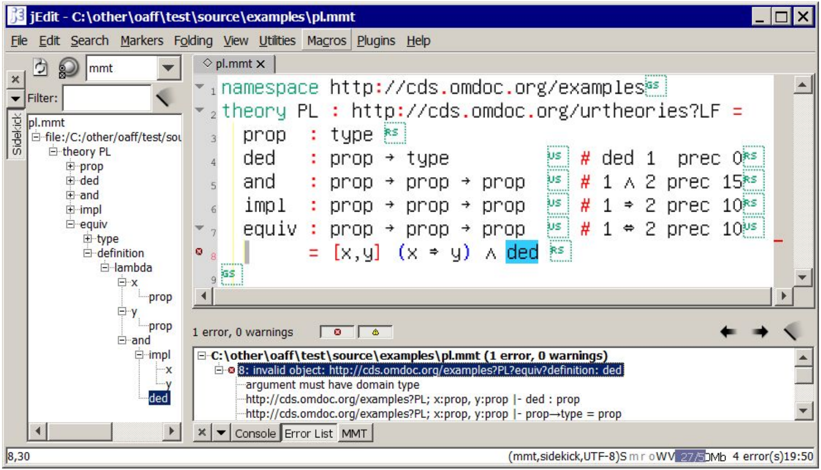
MMT-Based Foundation-Independent Results

IDE

- Inspired by programming language IDEs
- Components
  - jEdit text editor (in Java): graphical interface
  - MMT API (in Scala)
  - jEdit plugin to tie them together

  only $\sim$ 1000 lines of glue code
- Features
  - outline view
  - error list
  - display of inferred information
  - type inference of subterms
  - hyperlinks: jump to definition
  - search interface
  - context-sensitive auto-completion: show identifiers that

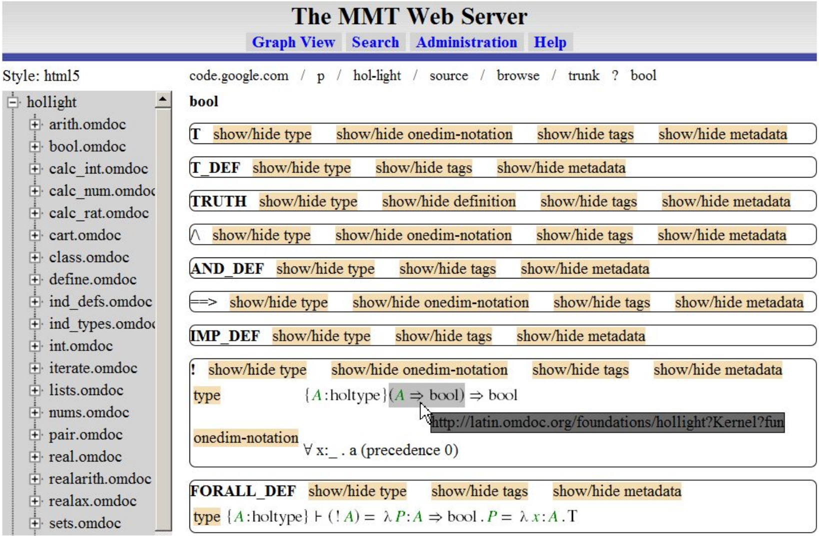
IDE: Example View



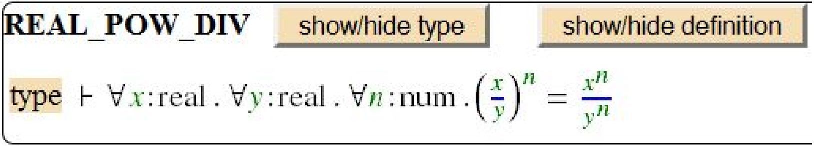
An Interactive Library Browser

- MMT content presented as HTML5+MathML pages
- Dynamic page updates via Ajax
- MMT used through HTTP interface with JavaScript wrapper
- Features
  - interactive display — e.g., inferred types, redundant brackets
  - smart navigation via MMT ontology — can be synchronized with jEdit
  - dynamic computation of content — e.g., definition lookup, type inference
  - graph view: theory diagram as SVG

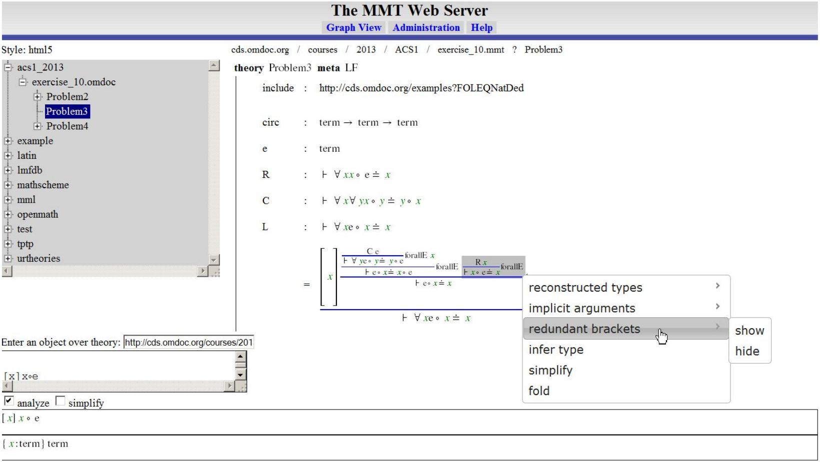
Browser: Example View



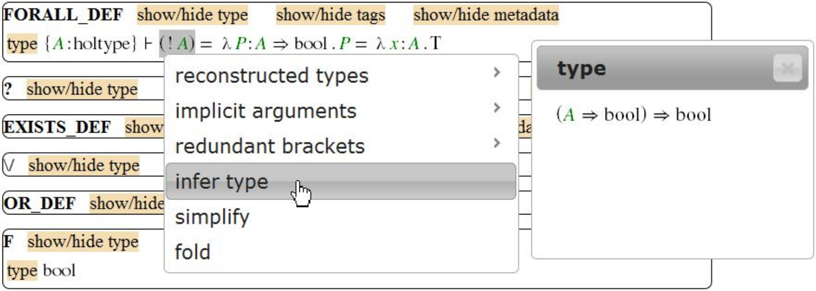
Browser Features: 2-dimensional Notations



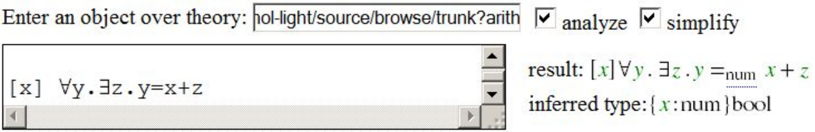
Browser Features: Proof Trees



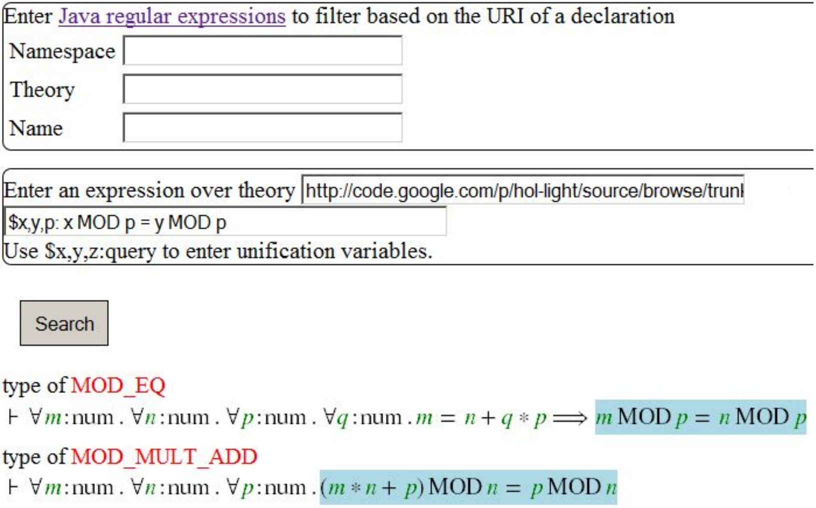
Browser Features: Type Inferece



Browser Features: Parsing



Example Service: Search



Example Service: Theory Graph Viewer

Theory graphs with 1000s of nodes
$\rightarrow$ special visualization tools needed recently even in 3D

demo at `https://www.youtube.com/watch?v=Mx7HSWD5dwg`

LaTeX Integration

- upper part: LaTeX source for the item on associativity
- lower part: pdf after compiling with LaTeX-MMT
- enriched with type inference, cross references, tooltips

e.g., type argument *M* of equality symbol

```
\begin{mmtscope}
 For all \mmtvar{x}{in M},\mmtvar{y}{in M},\mmtvar{z}{in M}
 it holds that !(x * y) * z = x * (y * z)!
\end{mmtscope}
```

A *monoid* is a tuple $(M, \circ, e)$ where

– M is a sort, called the universe.
– $\circ$ is a binary function on M.
– e is a distinguished element of M, the unit.

such that the following axioms hold:

– For all x,y,z it holds that $(x \circ y) \circ z =_M x \circ (y \circ z)$
– For all x it holds that $x \circ e =_M x$ and $e \circ x =_M x$.