MMT: A UniFormal Approach to Knowledge Representation

Florian Rabe

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

July 2018

About Me

My Background

- Areas
  - theoretical foundations
    logic, programming languages foundations of mathematics
  - formal knowledge representation
    specification, formalized mathematics, ontologies, programming
  - scalable applications
    module systems, libraries, system integration
- Methods
  - survey and abstract — understand fundamental concepts
  - relate and transfer — unify different research areas
  - long-term investment — identify stable ideas, do them right
  - modularity and reuse — maximize sharing across languages, tools

My Vision

UniFormal

a universal framework for the formal representation of knowledge

- integrate all domains
  specification, deduction, compuation, mathematics, ...
- integrate all formal systems
  logics, programming languages, foundations of mathematics, ...
- integrate all applications
  theorem provers, library managers, IDEs, wikis ...

My (evolving, partial) solution: MMT framework

- a uniformal knowledge representation framework
  developed since 2006, $\sim 100,000$ loc, ~ 500 pages of publications
- allows foundation-independent solutions
  module system, type reconstruction, ...
  IDE, search, build system, library, ...

`http://uniformal.github.io/`

Motivation

Logic in Computer Science

- $\sim$ 1930: computer science — vision of mechanizing logic
- Competition between multiple logics
  - axiomatic set theory: ZF(C), GBvN, ...
  - λ-calculus:
    - typed or untyped
    - Church-style or Curry-style
  - new types of logic modal, intuitionistic, paraconsistent ,...
- Diversification into many different logics
  - fine-tuned for diverse problem domains
    far beyond predicate calculus
  - deep automation support
    decision problems, model finding, proof search, ...
  - extensions towards programming languages

Selected Major Successes

Verified mathematical proofs

- 2006–2012: Gonthier et al., Feit-Thompson theorem
  170,000 lines of human-written formal logic
- 2003–2014: Hales et. al., Kepler conjecture (Flyspeck)
  $> 5,000$ processor hours needed to check proof

Software verification

- 2004–2010: Klein et al., L4 micro-kernel operating system
  390,000 lines of human-written formal logic
- since 2005: Leroy et al., C compiler (CompCert)
  almost complete, high performance

Knowledge-based Artificial intelligence

- since 1984: Lenat et al., common knowledge (CyC)
  2 million facts in public version
- since 2000: Pease et. al., foundation ontology (SUMO)
  25,000 concepts

Future Challenges

Huge potential, still mostly unrealized

Applications must reach much larger scales

- software verification successes dwarfed by practical needs
  internet security, safety-critical systems, ...
- automation of math barely taken seriously by mathematicians

Applications must become much cheaper

- mostly research prototypes
- usually require PhD in logic
- tough learning curve
- time-intensive formalization

The Dilemma of Fixed Foundations

Each system fixes a logic and/or programming language

- type theories, set theories, first-order logics, higher-order logics, ...
  ACL2, Coq, HOL, Isabelle/HOL, Matita, Mizar, Nuprl, PVS,...
- functional, imperative, inheritance-oriented, soft typing ...
  Axiom, Sage, GAP, Maple, ...
- Foundation-specific results
  contrast to mathematics: foundation left implicit
- All systems mutually incompatible

Exacerbates the other bottlenecks

- Human resource bottleneck
  - no reuse across systems
  - very slow evolution of systems
- Knowledge management bottleneck
  - retrofitting to fixed foundation systems very difficult
    can be easier to restart from scratch
  - best case scenario: duplicate effort for each system

Two Formidable Bottlenecks

Each system requires $\approx$ 100 person-year investment to

- design the foundational logic
- implement it in a computer system
- build and verify a collection of formal definitions and theorems
  e.g., covering undergraduate mathematics
- apply to practical problems

human resource bottleneck

New scales brought new challenges

- no good search for previous results
  reproving can be faster than finding a theorem
- no change management support
  system updates often break previous work
- no good user interfaces — far behind software engineering IDEs

knowledge management bottleneck

Example Problems

Collaborative QED Project, 1994

- high-profile attempt at building single library of formal mathematics
- failed partially due to disagreement on foundational logic

Voevodsky's Homotopy Type Theory, since 2012

- high-profile mathematician interested in applying logic
- his first result: design of a new foundation

Multiple 100 person-year libraries of mathematics

- developed over the last $\sim$ 30 years
- overlapping but mutually incompatible major duplication of efforts
- translations mostly infeasible

Hales's Kepler Proof

- distributed over two separate implementations of the same logic
- little hope of merging

Vision

UniFormal

a universal framework for the
formal representation of all knowledge and its semantics
in math, logic, and computer science

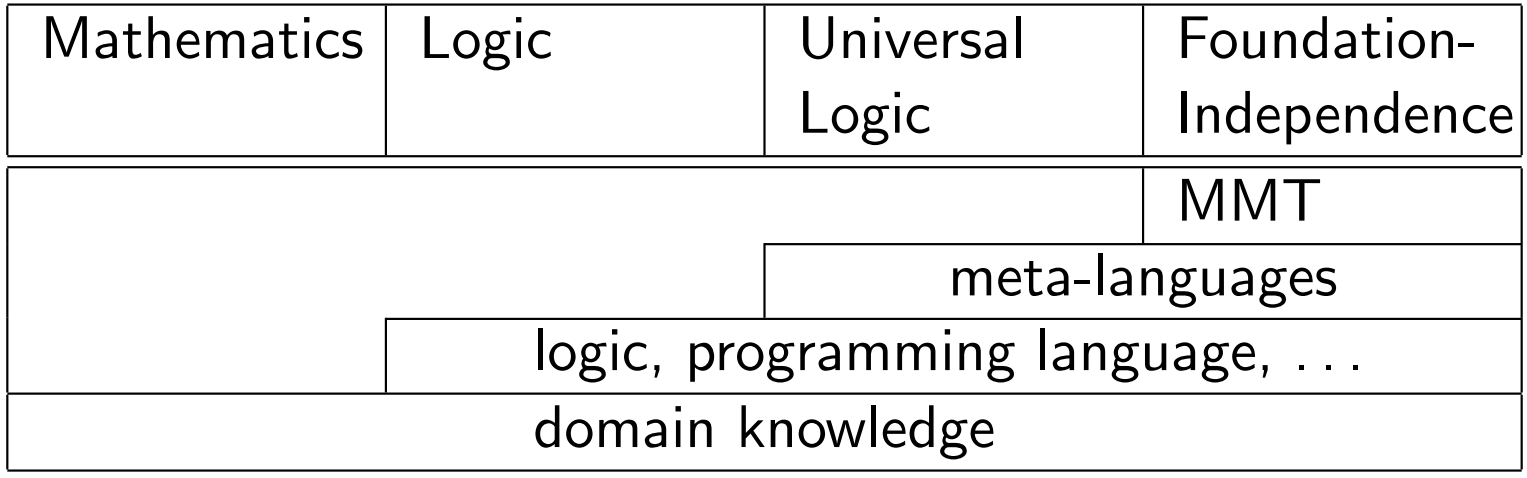
- Avoid fixing languages wherever possible . . .
- . . . and instantiate them for different languages
- Use formal meta-languages in which to define languages . . .
- . . . and avoid fixing even the meta-language
- Obtain foundation-independent results
  - Representation languages
  - Soundness-critical algorithms
  - Knowledge management services
  - User-facing applications

MMT as a UniFormal Framework

MMT = Meta-Meta-Theory/Tool

few primitives . . . that unify different domain concepts

- tiny but universal grammar for expressions
  syntax trees with binding
- standardized semantics of identifiers crucial for interoperability
- high-level definitions derivation, model, soundness, . . .
- no built-in logic or type system all algorithms parametric
- theories and theory morphisms for large scale structure
  translation, interpretation, semantics, . . .



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 ...

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
  ...
}
```

The LATIN Atlas

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

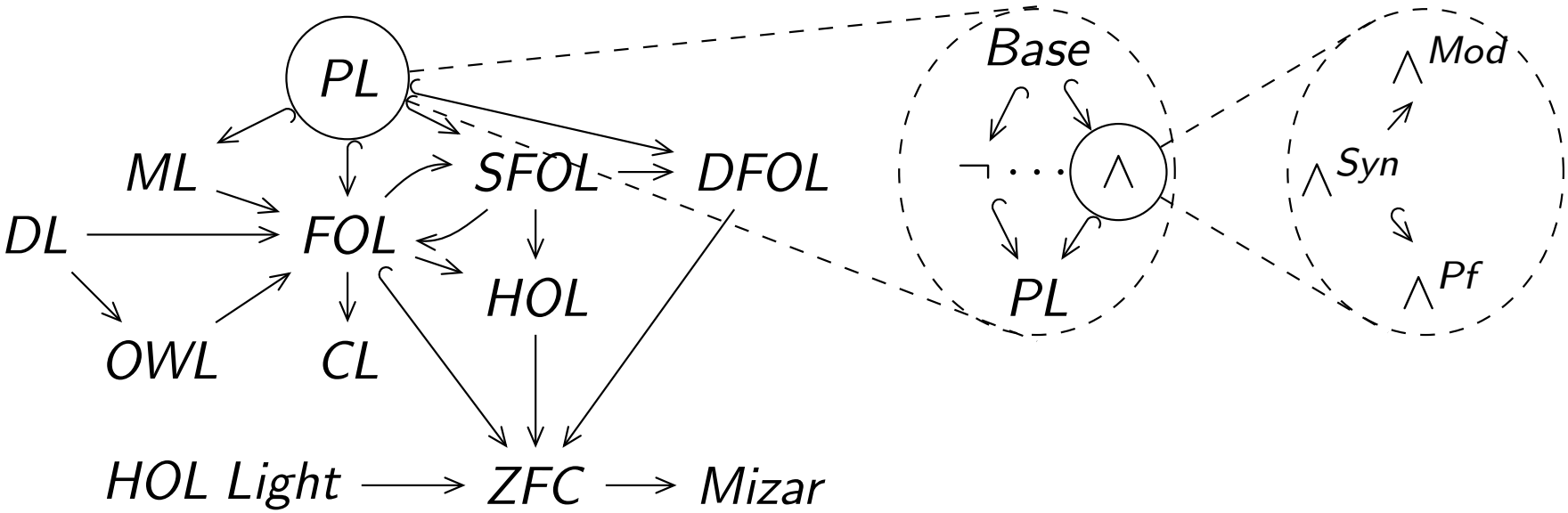
The LATIN Atlas of Logical Systems

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

Discussion

Foundation-Independent Development

Typical workflow

1. choose foundation
   type theories, set theories, first-order logics, higher-order logics, ...
2. implement kernel
3. build support algorithms reconstruction, proving, editor, ...
4. build library

Foundation-independent workflow in MMT

1. MMT provides generic kernel
   no built-in bias towards any foundation
2. build support algorithms on top of MMT
3. choose foundation(s)
4. customize MMT kernel for foundation(s)
5. build foundation-spanning universal library

Advantages

- Avoids segregation into mutually incompatible systems
- Formulate maximally general results
  meta-theorems, algorithms, formalizations
- Rapid prototyping for logic systems
  customize MMT as needed, reuse everyting else
- Separation of concerns between
  - foundation developers
  - support service developers: search, axiom selection, ...
  - application developers: IDE, proof assistant, wiki, ...
- Allows evolving foundation along the way
  design flaws often apparent only much later
- Migrate formalizations when systems die
- Archive formalizations for future rediscovery

Paradigms

- judgments as types, proofs as terms
  unifies expressions and derivations
- higher-order abstract syntax unifies operators and binders
- category of theories and theory morphisms
  - languages as theories
    unifies logical theories, logics, foundations
  - relations as theory morphisms
    unifies modularity, intepretations, representation theorems
- institution-style abstract model theory
  uniform abstract concepts
- models as morphisms (categorical logic)
  unifies models and translations and semantic interpretations

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

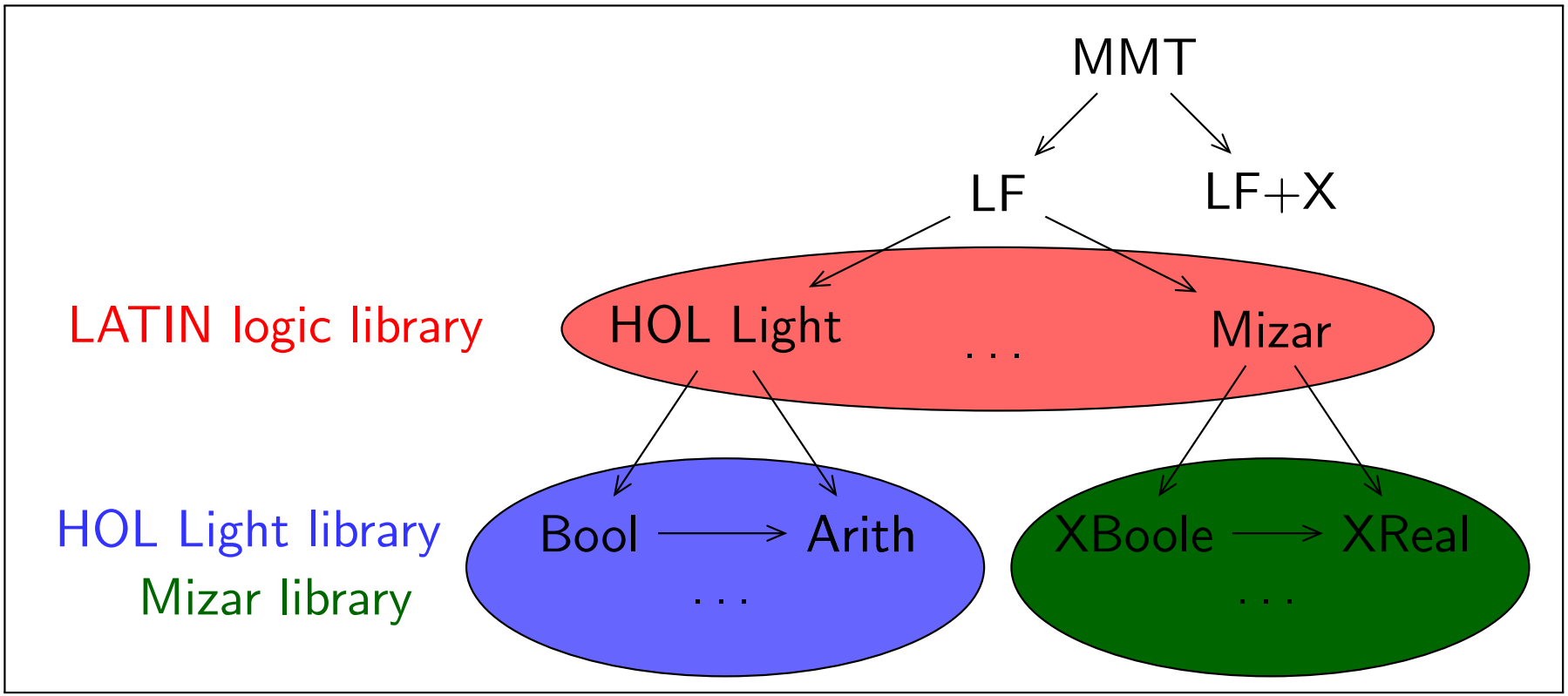
The OAF Archive

Ongoing: Open Archive of Formal Libraries

- Open Archive of Formalizations

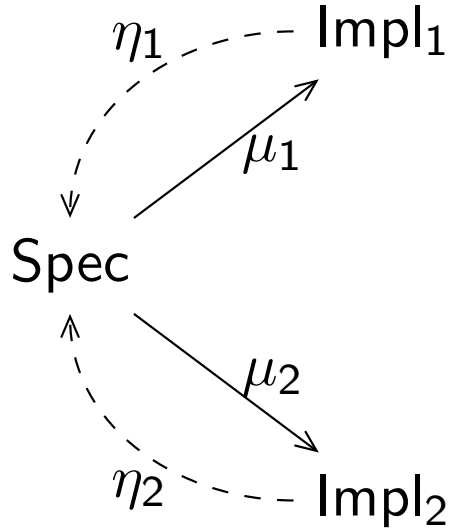
  Michael Kohlhase and myself, 2014-2017
- Goal: archival, comparison, integration of formal libraries

  Mizar, HOL systems, IMPS, Coq/Matita, PVS, ...
- Big, overlapping libraries – that are mutually incompatible



Library Integration

- Spec: mathematical theory axiomatic, foundation-independent
- Impl_i: implementation of Spec in system i
- μ_i: theory morphism describing how Spec is realized
  maps Spec-identifiers to their implementing objects
- η_i: partial inverse of μ_i



Challenge:

- collect initial library of mathematical concepts
- collect alignments with individual libraries

OpenDreamKit

Virtual Research Environments for Mathematics

- OpenDreamKit project 2015-2019 open PhD positions!
  EU project, 15 sites, 25 partners
  http://opendreamkit.org/
- Support full life-cycle
  - exploration, development, and publication
  - archival and sharing of data and computation
  - real mathematicians as target audience
- Key requirements
  - allow using any foundation
    any programming language, database
  - integrate narration, computation, databases
  - uniform user interfaces

Math in the Middle Approach

- Official definitions represented narratively
  MMT embedded into LaTeX to attach types
- Foundations written as MMT formal theories
- Computation: library interfaces exported as MMT theories
- Database schemata: written as formal MMT theories

Example work flow:

1. user input mentions specific object
   e.g., the 13th transitive group with conductor 5
2. Systems X queries MMT for $13a5$
3. MMT retrieves object from connected databases
   e.g., LMFDb (L-functions and modular forms)
4. Database schema defines type and encoding of object
5. MMT builds $13a5$ in high-level foundation
6. MMT exports $13a5$ in input syntax of system X

MMT-Based Foundation-Independent Results

Logical Result: Representation Language

- MMT theories uniformly represent
  - logics, set theories, type theories, algebraic theories, ontologies, ...
  - module system: state every result in smallest possible theory

    Bourbaki style applied to logic
- MMT theory morphisms uniformly represent
  - extension and inheritance
  - semantics and models
  - logic translations
- MMT objects uniformly represent
  - functions/predicates, axioms/theorems, inference rules, ...
  - expressions, types, formulas, proofs, ...
- Reuse principle: theorems preserved along morphisms

Logical Result: Concepts

MMT allows coherent formal definitions of essential concepts

- Logics are MMT theories
- Foundations are MMT theories e.g., ZFC set theory
- Semantics is an MMT theory morphism
  e.g., from FOL to ZFC
- Logic translations are MMT theory morphisms
- Logic combinations are MMT colimits

Logical Results: Algorithms

- 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?

Modular Framework Definitions in MMT

Individual features given by set of symbols, notations, rules

- $\lambda\Pi$
- Rewriting
- Polymorphism
- Subtyping (ongoing)
- ...

Language definitions are modular themselves

e.g., Dedukti = LF + rewriting

Knowledge Management Results

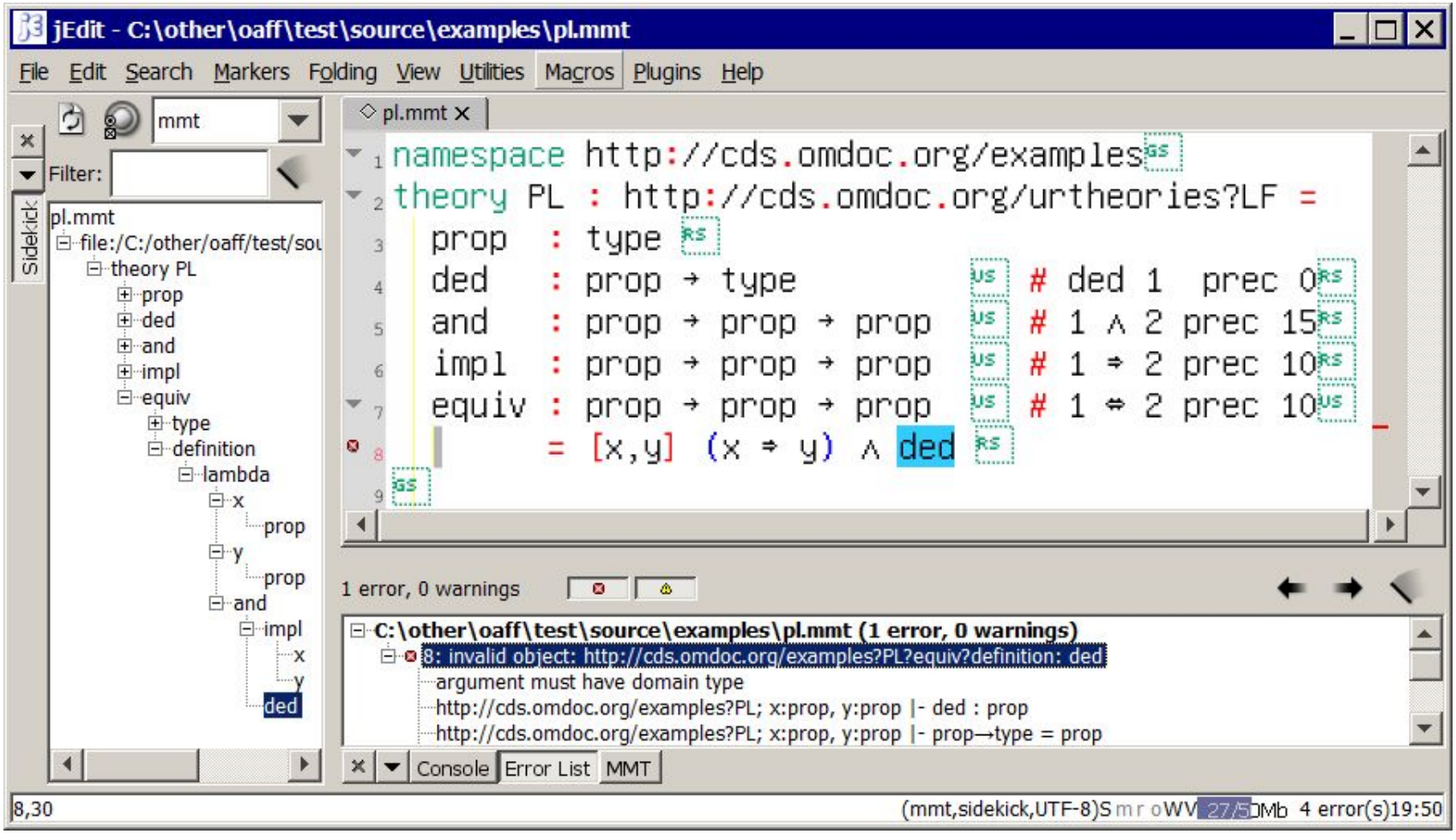
- Change management — recheck only if affected
- Project management — indexing, building
- Extensible export infrastructure — Scala, SVG graphs, LaTeX, HTML, ...
- Search, querying — substitution-tree and relational index
- Browser — interactive web browser
- Editing — IDE-like graphical interface

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

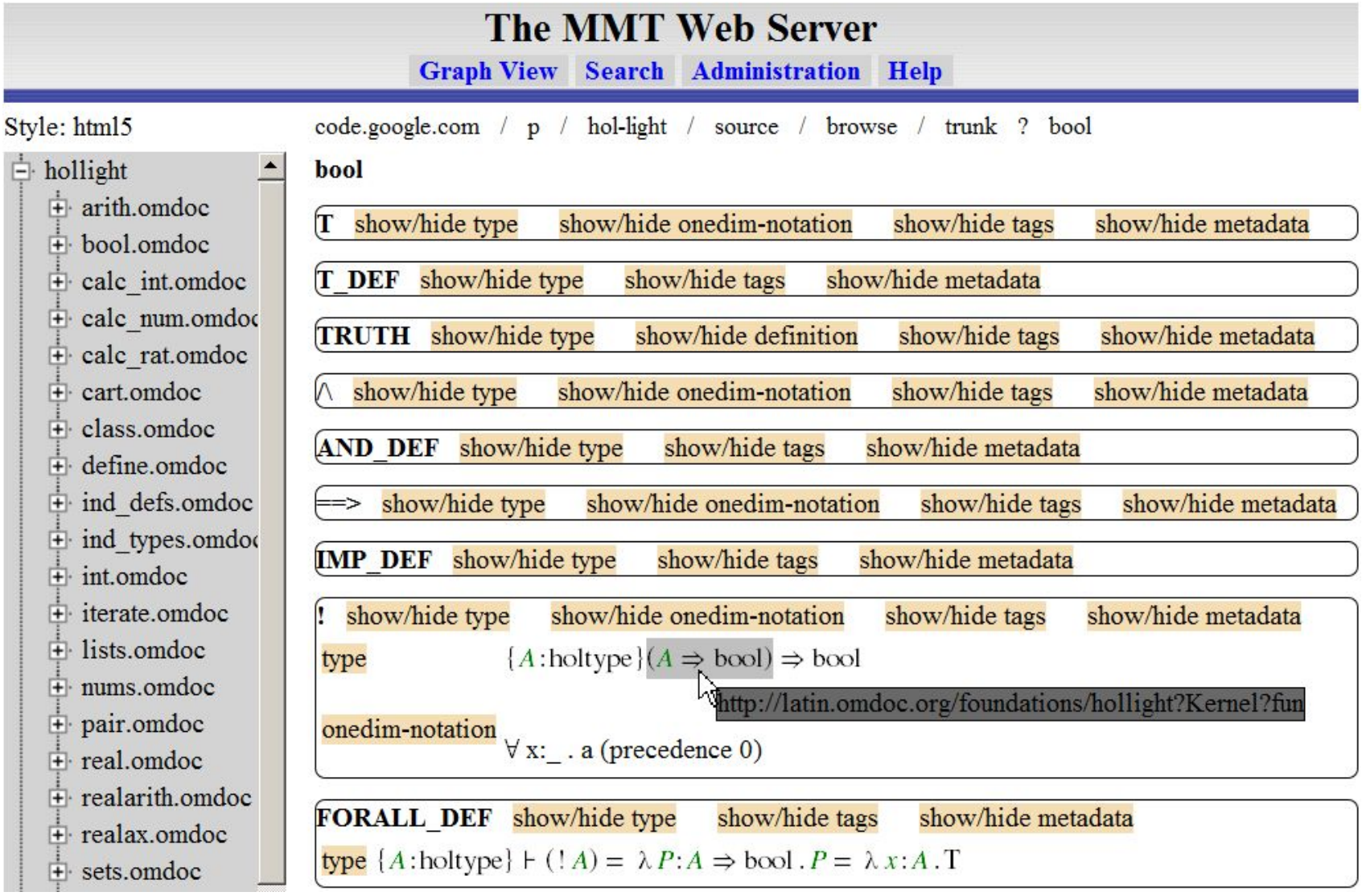
IDE: Example View



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

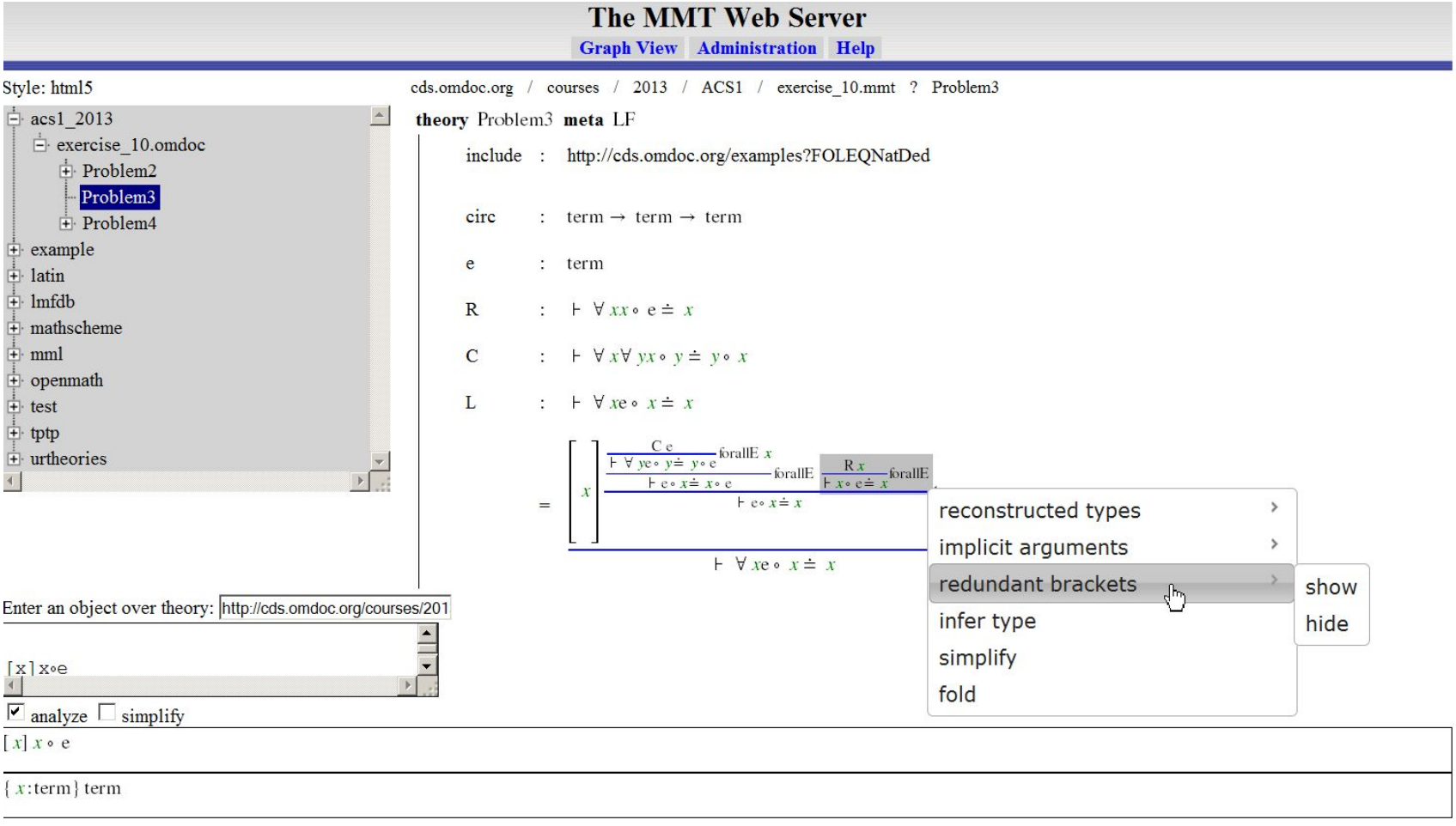


Browser Features: 2-dimensional Notations

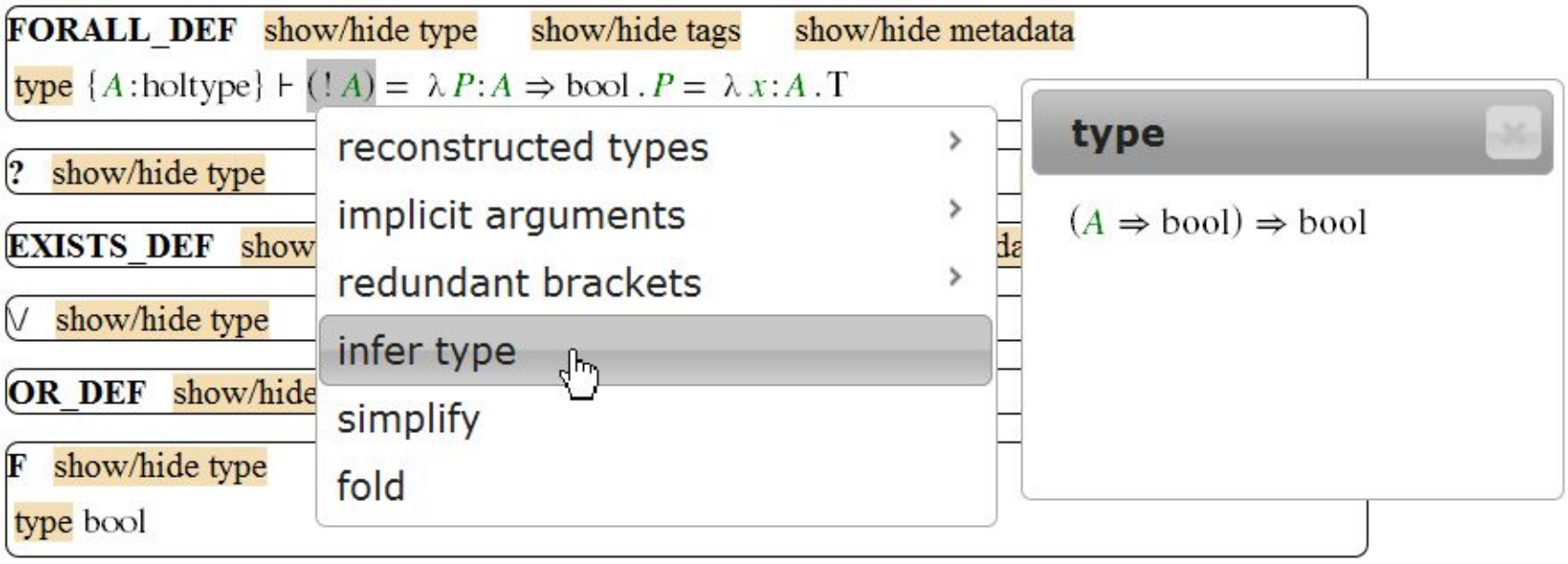
REAL_POW_DIV show/hide type show/hide definition

type $\vdash \forall x{:}\mathrm{real} . \forall y{:}\mathrm{real} . \forall n{:}\mathrm{num} . \left(\frac{x}{y}\right)^n = \frac{x^n}{y^n}$

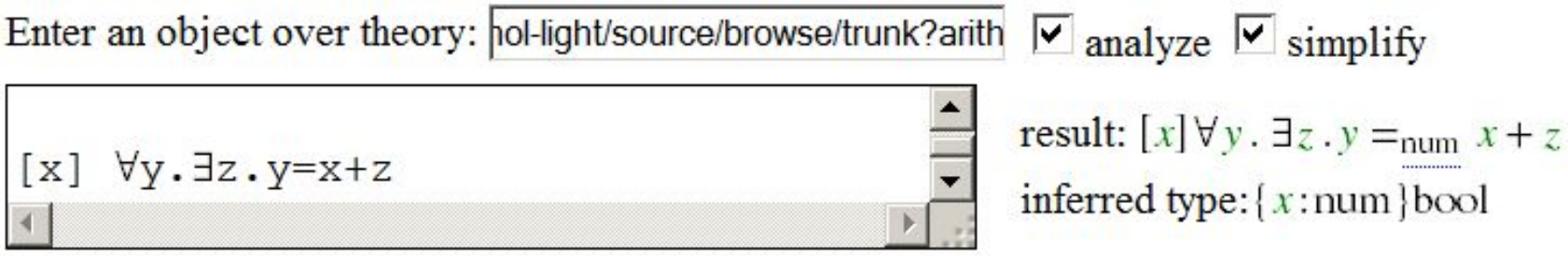
Browser Features: Proof Trees



Browser Features: Type Inferece



Browser Features: Parsing



Example Service: Search

Enter Java regular expressions to filter based on the URI of a declaration

Namespace

Theory

Name

Enter an expression over theory http://code.google.com/p/hol-light/source/browse/trunk

$x,y,p: x MOD p = y MOD p

Use $x,y,z:query to enter unification variables.

Search

type of MOD_EQ

$$\vdash \forall m{:}\text{num} . \forall n{:}\text{num} . \forall p{:}\text{num} . \forall q{:}\text{num} . m = n + q * p \Longrightarrow m \,\text{MOD}\, p = n \,\text{MOD}\, p$$

type of MOD_MULT_ADD

$$\vdash \forall m{:}\text{num} . \forall n{:}\text{num} . \forall p{:}\text{num} . (m * n + p) \,\text{MOD}\, n = p \,\text{MOD}\, n$$

Envisioned: A Generic Theorem Prover

- Theorem proving currently highly logic-specific
- But many successful designs actually logic-independent, e.g.,
  - Declarative proof languages
  - Tactic languages
  - Integration of decision procedures
  - Axiom selection
  - Modularity

Claim

- Logic-specific implementations a historical accident
- Possible to build MMT level proving technology

Envisioned: Computational Knowledge

- So far: MMT focuses on declarative formal languages
- Goal: extend to programming languages
  - understand key concepts foundation-independently
    state/effects, recursion, inheritance, ...
  - represent features modularly
  - freely mix logic and computation share declarative aspects
- Syntax is (kind of) easy:
  - programming languages are MMT theories
  - classes/modules are MMT theories
  - programs are expressions
- Semantics is open question: What is the general judgment?

Big Challenges

Exporting System Libraries

Major interoperability obstacle

- Communities very focused on
  - honing their system
  - curating their library
- Little effort for making libraries accessible
- Slowly improving
  - more people interested in sharing and generic tools
  - OAF a driving force

MMT good candidate for truly universal standard

Universal Library of Elementary Knowledge

Requirements

- language-independent
- system-independent
- easy to write for practitioners

Vision

- Central library uses modular library of language features
- Focus on names, types, properties — no definitions, no proofs
- Individual contributions piece together minimal needed language
- All existing systems
  - remain in use — e.g., for optimized proof support
  - allow refactoring results for submission to central library

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, browser, IDE, ...
- MMT quite mature now, ready for larger applications

  about to break even
- Serious contender for
  - universal library
  - generic applications/services
  - system integration/combination

Further Resources

Web sites

- MMT: `http://uniformal.github.io`
- OAF web server (experimental): `http://mathhub.info:8080/`

Selected publications all available from `https://kwarc.info/people/frabe`

- the primary paper on the MMT language (I&C 2013, with M. Kohlhase):
  A Scalable Module System
- a more recent paper on the MMT approach to logic (JLC 2014):
  How to Identify, Translate, and Combine Logics?
- Foundations in LATIN: (MSCS 2011, with M. Iancu)
  Formalizing Foundations of Mathematics
- Modular logics in LATIN (TCS 2011, with F. Horozal):
  Representing Model Theory in a Type-Theoretical Logical Framework
- the primary paper on OAF (under review 2015, with M. Kohlhase)
  QED Reloaded
- Mizar in OAF (JAR 2013, with M. Iancu, M. Kohlhase, J. Urban):
  The Mizar Mathematical Library in OMDoc: Translation and Applications
- HOL Light in OAF: (CICM 2014, with C. Kaliszyk)
  Towards Knowledge Management for HOL Light

Details: Foundations

Foundations

- Foundation = the most primitive formalism on which everything else is built
  set theories, type theories, logics, category theory, ...
- We can fix the foundation once and for all — but which one?

- In math: usually implicit and arbitrary foundation
  - can be seen as avoiding subtle questions
  - but also as a strength: it's more general
- In CS: each system fixes its own foundational language
  e.g., a variant of type theory or HOL
- Programming languages foundations as well
  but representation of state in MMT still open problem

Fixed Foundations

- Fixing foundation the first step of most implementations
  often foundation and implementation have the same name
- No two implementations for the exact same foundation
  even reimplementations diverge quickly
- Negative effects
  - isolated, mutually incompatible systems
    no sharing of results, e.g., between proof assistants
  - no large scale libraries
    each system's library starts from scratch
  - no library archival
    libraries die with the system
  - comparison of systems difficult
    no common problem set
  - slow evolution
    evaluating a new idea can take years

Details: Logical Frameworks

Logical Frameworks

= meta-logic in which syntax and semantics of object logics are defined

Advantages

- Universal concepts — expressions, substitution, typing, equality, ...
- Meta-reasoning — consistency, logic translations, ...
- Rapid prototyping — type reconstruction, theorem proving, ...

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 a meta-logic
- add other features
  - second level for induction on syntax (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)
  - ...

Problems

- Recent trend: 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

Customizable Formal Systems

Parallel trend in formal system design

- 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

Details: MMT Syntax

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., $\texttt{apply}(\cdot; f, a)$ for $(f\ a)$
  - typical binders: Γ and $\vec{E}$ have length 1
    e.g., $\texttt{lambda}(x : A; t)$ 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`	$\texttt{Pi}(x : E_1; E_2)$
$E_1 \to E_2$	`arrow # 1 → 2`	$\texttt{arrow}(\cdot; E_1, E_2)$
$\lambda x : E_1.E_2$	`lambda # λ V1 . 2`	$\texttt{lambda}(x : E_1; E_2)$
$E_1\ E_2$	`apply # 1 2`	$\texttt{apply}(\cdot; E_1, E_2)$

Notations

MMT implements parsing and rendering foundation-independently
relative to notations declared in current theory

Notations			$(ARG \mid VAR \mid DELIM)^*[PREC]$
Bound variable	VAR	$::=$	Vn for $n \in \mathbb{N}$
Argument	ARG	$::=$	n for $n \in \mathbb{N}$
Delimiter	$DELIM$	$::=$	Unicode string
Precedence	$PREC$	$::=$	integer

Bound variabes	`forall`	#	$\forall V1.2$
Mixfix	`setComprehension`	#	$\{V1 \in 2 \mid 3\}$
Argument sequences	`plus`	#	$1 + \ldots$
Variable sequences	`forall`	#	$\forall V1, \ldots 2$
Implicit arguments	`functionComposition`	#	$4 \circ 5$

Abstract Syntax of Theories

- Theories are named lists of declarations
- Theory names yield globally unique identifiers for all constants
- Module system: Previously defined theories can be included/instantiated

theory declaration		$T = \{Dec^*\}$
	Dec ::=	
constant declaration	|	$c[: E][= E][\#Notation]$
theory inclusion	|	`include` T
theory instantiation	|	`structure` $c : T$ `where` $\{Dec^*\}$

Flattening: Every theory is semantically equivalent to one without inclusions/instantiations intuition: theories are named contexts

Details: Typing

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

- MMT defines some rules once and for all

  foundation-independent rules
- Foundation-independent declared in theories

  rule for c defines when $c(\Gamma; E^*)$ well-typed

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 $T = \{\Sigma\}$

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

- Rules for well-formed contexts similar to theories

Foundation-Specific Rules

- Declared in theories as constants
- Module system allows composing foundations

Two options to give rules

1. For a few dedicated meta-logics
   - rules are constants without type, definiens, or notation
   - meaning provided externally — on paper or as code snippet
   - necessary to get off the ground
   - Examples:
     - logical framework LF: ~ 10 rules
     - shallow polymorphism: 1 rule
     - modulo rewriting: 1 rule family
2. For any other language
   - include a meta-logic
   - use meta-logic to give rules as typed constants
   - example: modus ponens using meta-logic LF

$$\begin{array}{lcl} o & : & \texttt{type} \\ \Rightarrow & : & o \to o \to o \\ \texttt{ded} & : & o \to \texttt{type} \\ \texttt{mp} & : & \Pi_{A,B} \texttt{ded}\,(A \Rightarrow B) \to \texttt{ded}\,A \to \texttt{ded}\,B \end{array}$$

Type Reconstruction

Type checking:

- input: judgement, e.g., $\Gamma \vdash_T t : A$
- output: true/false, error information

Type reconstruction

- input judgment with unknown meta-variables
  - implicit arguments, type parameters
  - omitted types of bound variables
- output: unique solution of meta-variables that makes judgement true
- much harder than type checking

MMT implements foundation-independent type reconstruction

- transparent to foundations
- no extra cost for foundation developer

Implementation

MMT implements foundation-independent parts of type checker

- foundation-independent rules
- simplification, definition expansion
- error reporting
- abstract interfaces for foundation-specific rules, e.g.,
  - infer type
  - check term at given type
  - check equality of two given terms
  - simplify a term

Foundation-specific plugins add concrete rules

- each rule can recurse into other judgements
- example LF: $\sim$ 10 rules for LF, $\sim$ 10 lines of code each

Details: Theory Morphisms

One More Basic Concept: Theory Morphisms

Theories

- uniform representation of
  - foundations — e.g., logical frameworks, set theories, ...
  - logics, type theories
  - domain theories — e.g., algebra, arithmetic, ...
- little theories: state every result in smallest possible theory
  maximizes reuse

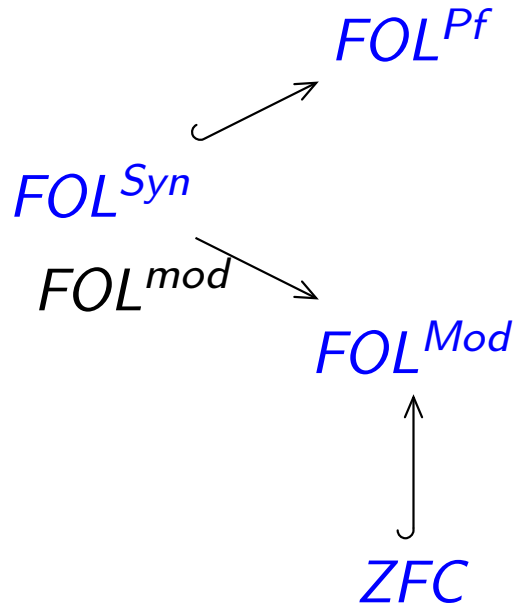
Theory morphisms

- uniform representation of
  - extension — e.g., Monoid $\to$ Group
  - inheritance — e.g., superclass $\to$ subclass
  - semantics — e.g., FOL $\to$ ZFC
  - models — e.g., Nat: Monoid $\to$ ZFC
  - translation — e.g., typed to untyped FOL
- homomorphic translation of expressions
- preserve typing (and thus truth)

Details: LATIN

Logic Example

- FOL^{Syn}: $term : type$, $prop : type$, $\mathtt{ded} : o \to type$, $\neg$, $\wedge$, ...
- FOL^{Pf}: $\neg I$, $\neg E$, $\wedge E_l$, $\wedge E_r$, $\wedge I$, ...
- ZFC: $set : type$, $wff : type$, $thm : wff \to type$, $\varnothing : set$, ...
- FOL^{Mod}: $univ : set$, $nonempty : true\ (univ \neq \varnothing)$
- FOL^{mod}: $term := univ$, $prop := \{0, 1\}$, $\mathtt{ded} := \lambda p.p \doteq 1$

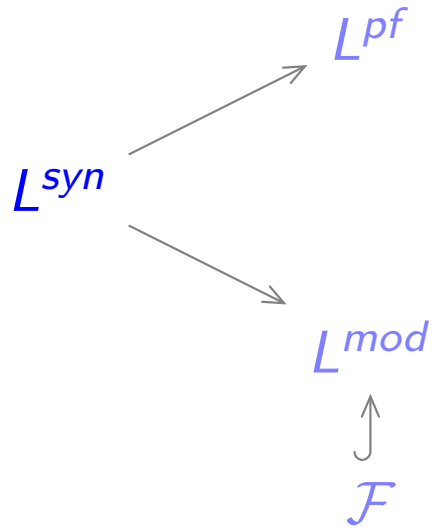


Current State

- Little theories including
  - propositional, common, modal, description, linear logic, unsorted/sorted/dependently-sorted first-order logic, CASL, higher-order logic
  - λ-calculi (λ-cube), product types, union types, ...
  - ZFC set theory, Mizar's set theory, Isabelle/HOL
  - category theory
- Little morphisms including
  - relativization of quantifiers from sorted first-order, modal, and description logics to unsorted first-order logic
  - negative translation from classical to intuitionistic logic
  - translation from type theory to set theory
  - translations between ZFC, Mizar, Isabelle/HOL
  - Curry-Howard correspondence between logic, type theory, and category theory

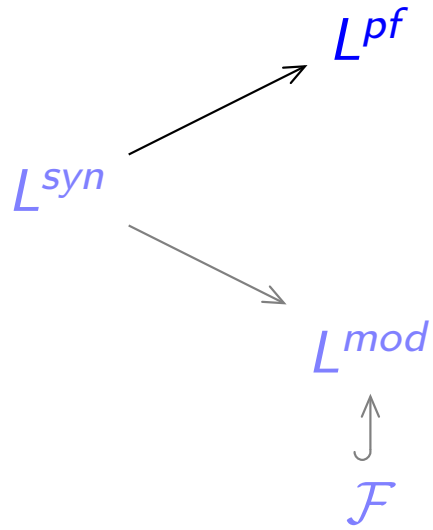
Representing Logics in LATIN

- L^{syn}: Syntax of L: connectives, quantifiers, etc.
  e.g., $\Rightarrow$: $o \to o \to o$
- L^{pf}: Proof theory of L: judgments, proof rules
  e.g., $impE : \mathtt{ded}\,(A \Rightarrow B) \to \mathtt{ded}\,A \to \mathtt{ded}\,B$
- L^{mod}: Model theory of L in terms of foundation $\mathcal{F}$
  e.g., $univ : set$, $nonempty : true\ (univ \neq \varnothing)$



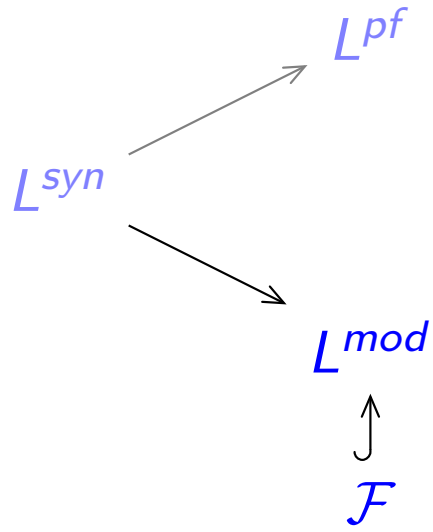
Representing Logics in LATIN

- L^{syn}: Syntax of L: connectives, quantifiers, etc.
  e.g., $\Rightarrow: o \to o \to o$
- L^{pf}: Proof theory of L: judgments, proof rules
  e.g., $impE : \mathtt{ded}\,(A \Rightarrow B) \to \mathtt{ded}\,A \to \mathtt{ded}\,B$
- L^{mod}: Model theory of L in terms of foundation $\mathcal{F}$
  e.g., $univ : set$, $nonempty : true\ (univ \neq \varnothing)$

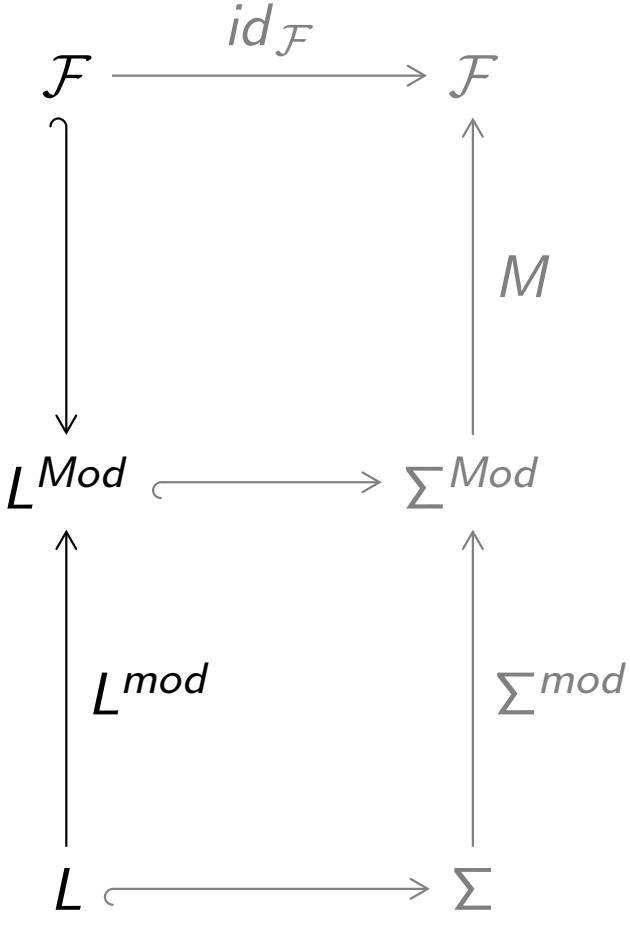


Representing Logics in LATIN

- L^{syn}: Syntax of L: connectives, quantifiers, etc.
  e.g., $\Rightarrow: o \to o \to o$
- L^{pf}: Proof theory of L: judgments, proof rules
  e.g., $impE : \mathtt{ded}\,(A \Rightarrow B) \to \mathtt{ded}\,A \to \mathtt{ded}\,B$
- L^{mod}: Model theory of L in terms of foundation $\mathcal{F}$
  e.g., $univ : set$, $nonempty : true\ (univ \neq \varnothing)$



Representing Logics and Models

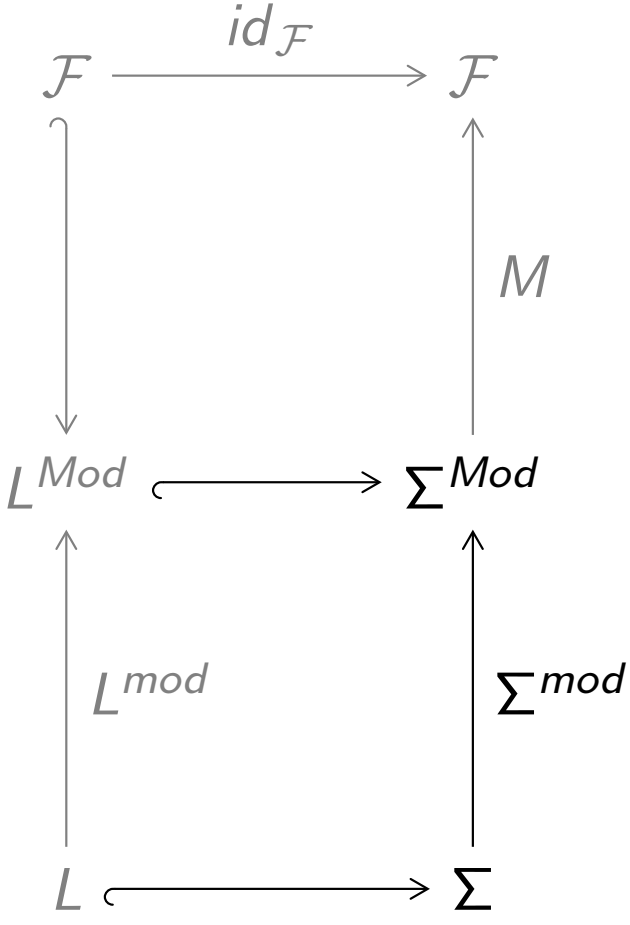


L encodes syntax and proof theory
$\mathcal{F}$ encodes foundation of mathematics
L^{Mod} axiomatizes models
L^{mod} interprets syntax in model

Σ encodes a theory of L,
extends L with functions, axioms, etc.
Σ^{Mod} correspondingly extends L^{Mod}
Σ^{mod} interprets syntax in model

M encodes a model of Σ,
interprets free symbols of L^{Mod} and Σ^{Mod}
in terms of $\mathcal{F}$

Representing Logics and Models

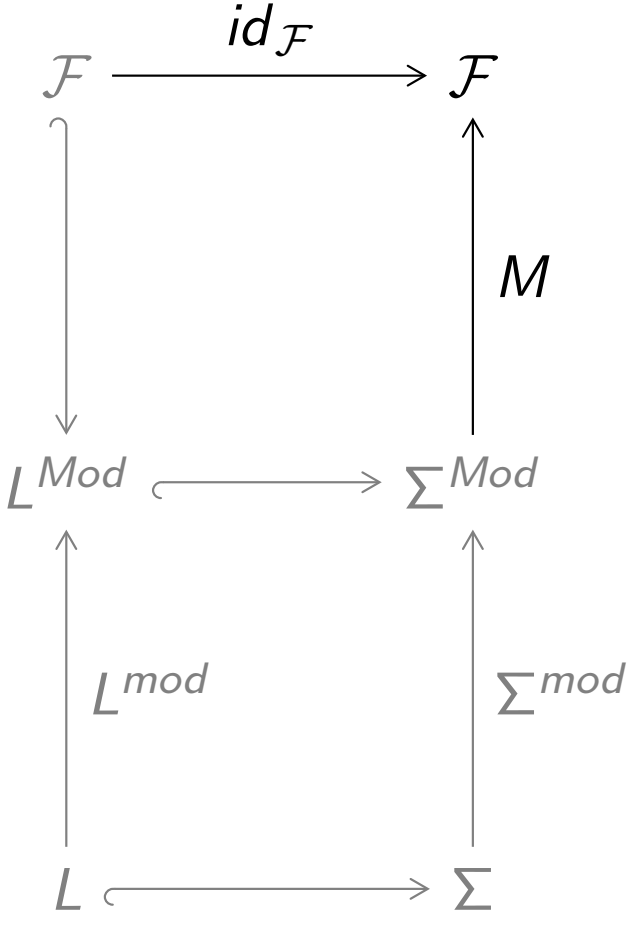


L encodes syntax and proof theory
$\mathcal{F}$ encodes foundation of mathematics
L^{Mod} axiomatizes models
L^{mod} interprets syntax in model

Σ encodes a theory of L,
extends L with functions, axioms, etc.
Σ^{Mod} correspondingly extends L^{Mod}
Σ^{mod} interprets syntax in model

M encodes a model of Σ,
interprets free symbols of L^{Mod} and Σ^{Mod}
in terms of $\mathcal{F}$

Representing Logics and Models



L encodes syntax and proof theory
$\mathcal{F}$ encodes foundation of mathematics
L^{Mod} axiomatizes models
L^{mod} interprets syntax in model

Σ encodes a theory of L,
extends L with functions, axioms, etc.
Σ^{Mod} correspondingly extends L^{Mod}
Σ^{mod} interprets syntax in model

M encodes a model of Σ,
interprets free symbols of L^{Mod} and Σ^{Mod}
in terms of $\mathcal{F}$

Details: Open Archive of Formalizations

Goal: Universal Library Infrastructure

- MMT as representation language
- Repository backend: MathHub
  - based on GitLab – open-source analog of GitHub server
  - GitLab instance hosted at Jacobs University
  - free registration of accounts, creation of repositories
- Generic library management
  - browser
  - inter-library navigation
  - search
  - change management

Goal: Exports from Proof Assistants

- Export major libraries into MMT
- Representative initial targets
  - Mizar: set theoretical initial export done (with Josef Urban)
  - HOL Light: higher-order logic initial export done (with Cezary Kaliszyk)
  - Coq or Matita: type theoretical
  - IMPS: little theories method
  - PVS: rich foundational language
- Major technical difficulty
  - exports must be written as part of proof assistant
  - not all information available

Goal: Towards Library Integration

- Refactor exports to introduce modularity
- 2 options
  - systematically during export
    e.g., one theory for every HOL type definition
  - heuristic or interactive MMT-based refactoring
- Collect correspondences between concepts in different libraries
  heuristically or interactively
- Relate isomorphic theories across languages
- Use partial morphisms to translate libraries

Details: Systems

MMT and Hets

- Hets
  - integration system for specification languages and associated tools
  - developed at DFKI by Till Mossakowski et al.
  - limited foundation-independence
    - multiple foundations possible
    - but must be implemented individually
- Integration
  1. logics defined declaratively in MMT
  2. MMT generates Hets logic definitions
     includes logic-specific abstract data types
  3. new logics dynamically available in Hets
- Generated logics use MMT via HTTP for parsing/type-checking

LaTeX Integration

- MMT declarations spliced into LaTeX documents
  shared MMT-LaTeX knowledge space
- LaTeX macros for MMT-HTTP interface
- Semantic processing of formulas
  - parsing
  - type checking
  - semantic enrichment: cross-references, tooltips
- Design not LaTeX-specific
  e.g., integration with word processors possible

LaTeX Integration: Example

Inferred arguments are inserted during compilation:

- upper part: LaTeX source for the item on associativity
- lower part: pdf after compiling with LaTeX-MMT
- type argument M of equality symbol is inferred and added by MMT

```
\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

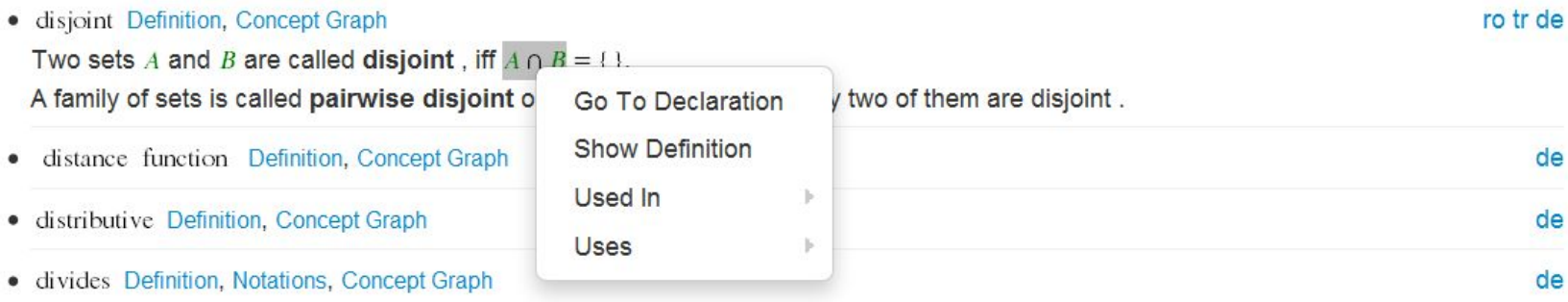
- M is a sort, called the universe.
- $\circ$ is a binary function on M.
- e is a distinguished element of M, the unit.

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 $e \circ x =_M x$.

SMGloM glossary

- Multi-lingual mathematical glossary
  https://mathhub.info/mh/glossary
  Kohlhase and others, 2013–2015
- Includes notations and verbalizations
  . . . but makes no commitment to formal system
- Cross-referenced and searchable
- $\approx$ 1000 entries
- Uses MMT as background representation language
  integrates MMT with natural language



MMT in the Semantic Alliance System

- Semantic Alliance System
  - developed by Michael Kohlhase et al.
  - two DFG projects
    - SiSSi (Hutter, Kohlhase) spreadsheet applications
    - FormalCAD (Kohlhase, Schröder) CAD applications
- Enrich domain-specific applications with semantic services
- Ontology used to
  - formalize background knowledge
  - share knowledge between applications
- MMT used as interface to ontology