MMT: A Foundation-Independent Approach to Declarative Languages

Florian Rabe

Jacobs University Bremen

March 30 2015

My Background

- Areas
  - theoretical foundations
    logic, type theory, foundations of mathematics
  - formal knowledge representation
    specification, formalized mathematics, ontologies
  - scalable applications
    module systems, libraries, system integration
- Vision
  - Develop a universal framework
    for the formal representation of knowledge and its semantics,
  - apply it to the safe and scalable integration of
    math, logic, and computer science.
- 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

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

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

Overview

- Foundation-independent framework
  - avoid fixing foundation wherever possible
  - design and implement generically
  - permit instantiation with different foundations
- MMT language
  - prototypical formal declarative language
  - admits concise representations of most languages
  - continued development since 2006 (with Michael Kohlhase)
  - $>$ 100 pages of publication
- MMT system
  - API and services
  - continued development since 2007 (with $>$ 10 students)
  - $> 30,000$ lines of Scala code
  - $\sim$ 10 papers on individual aspects

The Meta-Meta-Logical Approach

- Foundation-independence = last step in a progression of abstractions
- MMT governs the meta-meta-level
- In retrospect, I define MMT as Meta-Meta-Tool

<table>
<tr><td>Mathematics</td><td>Logic</td><td>Meta-logic</td><td>Foundation-Independence</td></tr>
<tr><td colspan="3"></td><td>MMT</td></tr>
<tr><td colspan="2"></td><td colspan="2">logical framework</td></tr>
<tr><td></td><td colspan="3">logic</td></tr>
<tr><td colspan="4">domain knowledge</td></tr>
</table>

Foundation-Independence

- MMT arises by iterating the following steps
  1. Choose a typical problem
  2. Survey and analyze the existing solutions
  3. Differentiate between foundation-specific and foundation-independent concepts/problems/solutions
  4. Integrate the foundation-independent aspects into MMT
  5. Define interfaces to supply the foundation-specific aspects
- Separation of concerns between
  - foundation developers — focus on logical core
  - service developers — e.g., search
  - application developers — e.g., IDE

yields rapid prototyping for logic systems

Foundation-Independence

- MMT arises by iterating the following steps
  1. Choose a typical problem
  2. Survey and analyze the existing solutions
  3. Differentiate between foundation-specific and foundation-independent concepts/problems/solutions
  4. Integrate the foundation-independent aspects into MMT
  5. Define interfaces to supply the foundation-specific aspects
- Separation of concerns between
  - foundation developers — focus on logical core
  - service developers — e.g., search
  - application developers — e.g., IDE

  yields rapid prototyping for logic systems
- But how much can really be done foundation-independently?

  MMT shows: not everything, but a lot

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, ...
- 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
}
```

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
  ...
}
```

Theories and 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 → Group
    - inheritance — e.g., superclass → subclass
    - semantics — e.g., FOL → ZFC
    - models — e.g., Nat: Monoid → ZFC
    - implementation — e.g., specification → programming language
    - translation — e.g., typed to untyped FOL
    - functors — e.g., output → input interface
  - homomorphic translation of expressions
  - preserve typing (and thus truth)

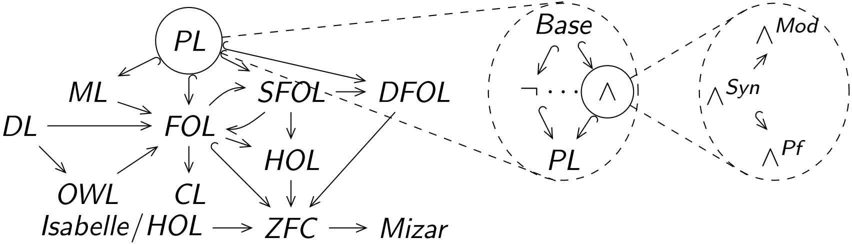
Large Scale Example: The LATIN Atlas

- 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

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

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

Logical Results Obtained at the MMT Level

- 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? very recent, ongoing
- Code generation? most likely

Abstract Syntax

Key ideas

- no predefined constants
- very general term constructor
- $c(\Gamma; \vec{E})$ binds variables and takes arguments
  - non-binding operators: Γ empty e.g., $\texttt{apply}(\cdot; f, a)$ for LF's $f\ a$
  - typical binders: Γ and $\vec{E}$ have length 1
    e.g., $\texttt{lambda}(x:A; t)$ for LF's $\lambda x:A.t$

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

MMT implements foundation-independent data structures for theories and terms

Concrete Syntax

- Notations of c introduce connect concrete syntax for $c(\Gamma; \vec{E})$

e.g., for LF

production	MMT 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`$(\cdot; E_1, E_2)$

MMT implements foundation-independent parser and printers

Inference System

For any theory Σ:

$\vdash_\Sigma \Gamma$	Γ is a valid context
$\Gamma \vdash_\Sigma t : A$	t has type A
$\Gamma \vdash_\Sigma E = E'$	E and E' are equal
$\Gamma \vdash_\Sigma \_ : A$	A is inhabitable

MMT define some foundation-independent rules

- congruence rules for equality
- contexts

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

- rules for atomic terms, e.g.

$$\frac{x : A \text{ in } \Gamma}{\Gamma \vdash_\Sigma x : A} \qquad \frac{x = t \text{ in } \Gamma}{\Gamma \vdash_\Sigma x = t}$$

Foundation-specific rules for complex terms are

- declared in theories
- implemented by plugins

Inference System: Implementation

MMT implements foundation-independent parts of type checker

- foundation-independent rules
- lookup in theories, context
- simplification, definition expansion
- error reporting

Foundation-specific rules supplied by plugins

- ~ 8 abstract rules, e.g.,
  - infer type
  - check term at given type
  - check equality of two given terms
  - simplify a term
- each rule can recurse into other judgements
- plugins provide concrete instances
- Example LF: ~ 10 rules for LF, ~ 10 lines of code each

Inference System: Type Reconstruction

Type Reconstruction

- Judgment with unknown meta-variables
  - implicit arguments, type parameters
  - omitted types of bound variables
- Goal: prove judgment and solve meta-variables
- Much harder than type checking requires delaying constraints

MMT implements foundation-independent type reconstruction

- transparent to foundations
- (almost) no extra cost for foundation developer

one additional rule for LF

Application-Independence

- Practical logic-related systems often application-specific
  - fixed functionality for fixed foundation
    often: read-eval-print design
  - many applications shallow, decay quickly
- MMT approach: application-independence
  1. focus on API for MMT language
     data structures and high-level interfaces
  2. encapsulate advanced functionality in reusable services
     e.g., theorem proving, search, ...
  3. allow plugin interfaces for customization
     foundation-specific rules, parsers, change listeners, ...
  4. build lightweight applications on top IDE, library manager, ...

Knowledge Management Results at the MMT Levels

- 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

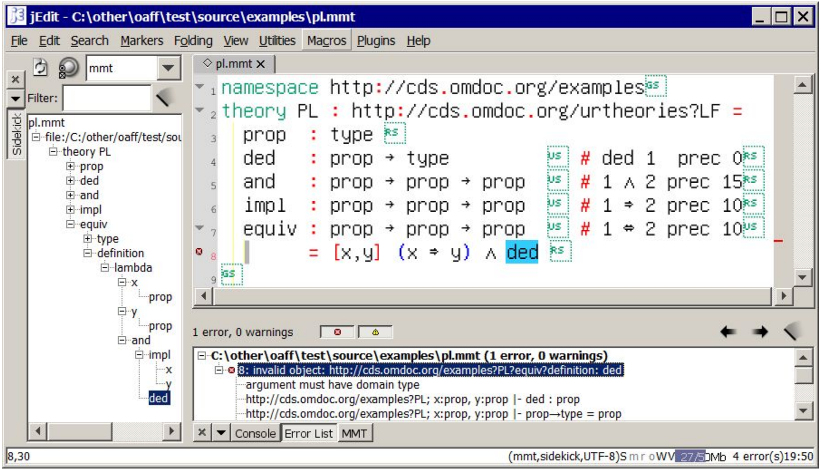
IDE

- 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

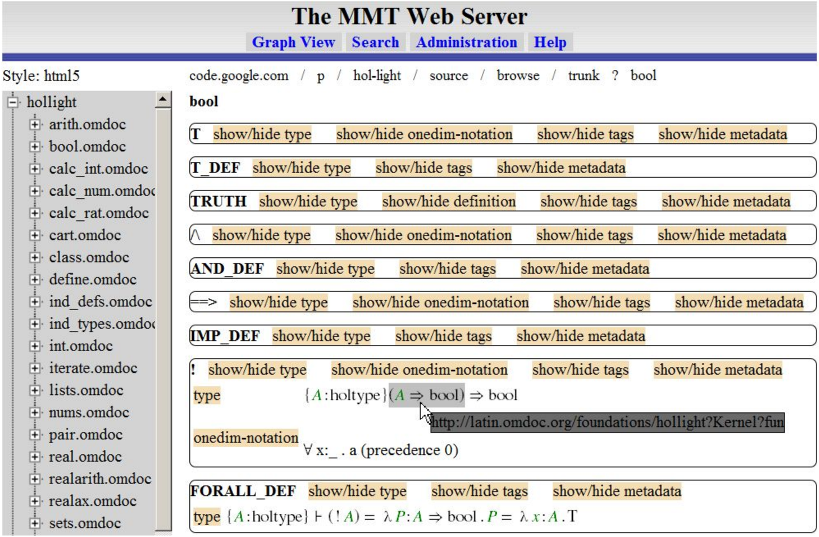
IDE: Example View



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

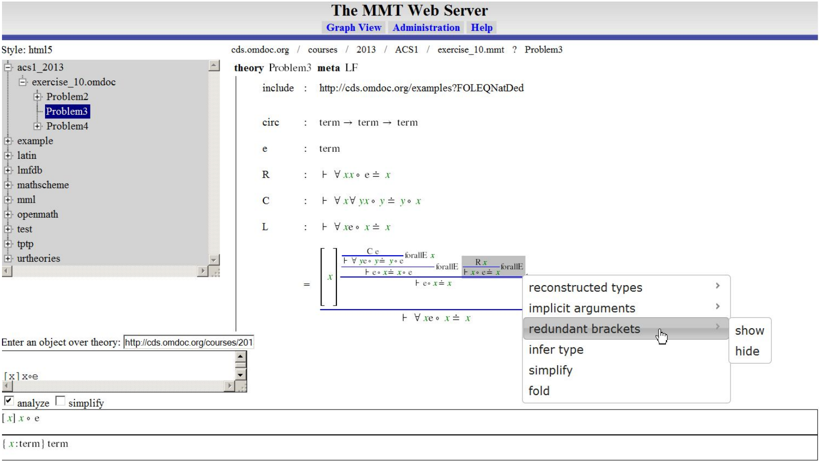


Browser Features: 2-dimensional Notations

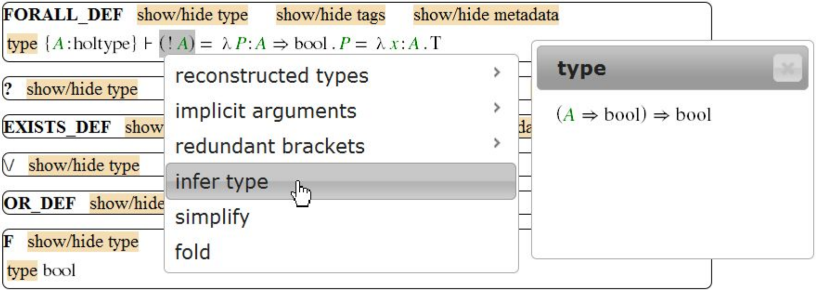
REAL_POW_DIV show/hide type show/hide definition

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

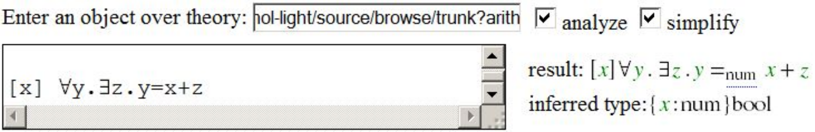
Browser Features: Proof Trees



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

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

type of MOD_MULT_ADD

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

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

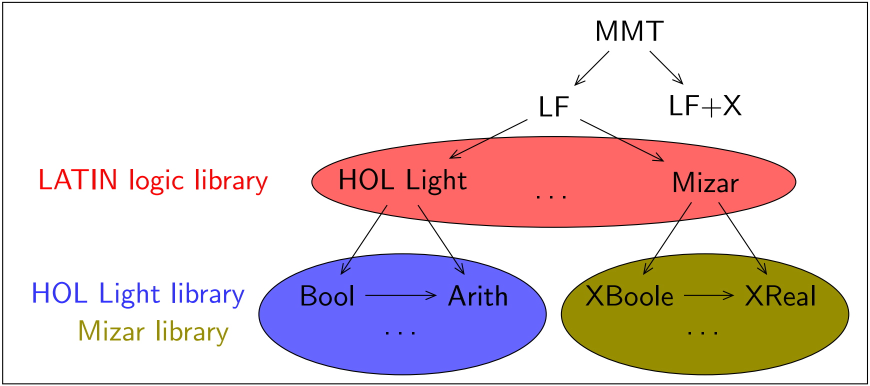
- M is a sort, called the universe.
- $\circ$ is a binary function on M.
- e is a distinguished element of M, the unit.

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 $e \circ x =_M x$.

Current Work: Library Integration

- Open Archive of Formalizations open PhD position!
  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



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

Conclusion

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

  rapid prototyping logic systems
- Multiple 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
- Interesting for
  - new, changing foundations
  - generic applications/services
  - system integration/combination

Further Resources

Web sites

- MMT: https://svn.kwarc.info/repos/MMT/doc/html/index.html
- 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*
- 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*