The MMT Language and System
The LATIN Logic Atlas

Florian Rabe

Jacobs University Bremen

Deducteam-Kwarc workshop, April 12 2013

MMT Vision

- Universal framework for mathematical-logical content
- Close relatives
  - LF, Isabelle: but more universal, knowledge management, more system integration
  - OMDoc/OpenMath: but formal semantics, automation
- Typical use case
  1. define a logical framework in MMT — e.g., LF
  2. use it to define a logic in MMT — e.g., HOL
  3. optionally: write and register plugins — e.g., type checking
  4. MMT induces a system for that logic

     provides logical and knowledge management services

     handles system integration

Statistics

- MMT language
  - 5 years of development (with Michael)
  - $\sim$ 100 pages write-up
- MMT API
  - 5 years of development (with various students)
  - 30,000 lines of Scala code
  - $\sim$ 10 papers on individual aspects

Example: small scale

- **Little theories**: state every definition/theorem/algorithm in the smallest possible theory/logic/logical framework
- **Theory morphisms**: transport results across theories/logics/logical frameworks

```
theory Types { type }
theory LF {include Types, Π, →, λ, @ }

theory Logic meta LF {o: type, ded : o → type }
theory FOL meta LF {
  include Logic
  u: type. ⇒: o → o → o, ...
}

theory Magma meta FOL { ∘: u → u → u }
⋮
theory Ring meta FOL {
  additive: CommutativeGroup
  multiplicative: Semigroup
  ...
}
```

Example: large scale

- LATIN atlas of logics: highly interconnected network of logic formalizations
- Written in MMT/LF using Twelf
- 4 years, $\sim$ 10 authors, $\sim$ 1000 modules
- Focus on breadth (= many formal systems represented), not so much depth (= theorems in particular systems)
- Each logic root for library of that logic
- Each edge yields library translation functor

Example: large scale

- LATIN atlas of logics: highly interconnected network of logic formalizations
- Written in MMT/LF using Twelf
- 4 years, $\sim$ 10 authors, $\sim$ 1000 modules
- Focus on breadth (= many formal systems represented), not so much depth (= theorems in particular systems)
- Each logic root for library of that logic
- Each edge yields library translation functor

Important meta-result: the logical framework should be flexible

MMT Design Methodology

1. Choose a typical problem

   logical: e.g., type reconstruction, reflection

   MKM: e.g., change management, querying

2. Survey and analyze the existing solutions
3. Differentiate between foundation-specific and foundation-independent definitions/theorems/algorithms
4. Integrate the foundation-independent aspects into MMT

   language and system

5. Define interfaces to supply the logic-specific aspects

   formal and plugin interfaces

6. Repeat

Foundation-Independence

1. We can fix and implement a logical theory e.g., set theory

Foundation-Independence

1. We can fix and implement a logical theory e.g., set theory
2. We can fix and implement a logic
   then define many theories in it e.g., first-order logic

Foundation-Independence

1. We can fix and implement a logical theory e.g., set theory
2. We can fix and implement a logic
   then define many theories in it e.g., first-order logic
3. We can fix and implement a logical framework
   then define many logics in it the foundation, e.g., LF

Foundation-Independence

1. We can fix and implement a logical theory e.g., set theory
2. We can fix and implement a logic
   then define many theories in it e.g., first-order logic
3. We can fix and implement a logical framework
   then define many logics in it the foundation, e.g., LF
4. We can fix and implement meta-framework
   then define many logical frameworks in it
   foundation-independence, e.g., MMT

The Promise and Danger of Abstraction

- Abstraction chain

  $$\textit{theory} \to \textit{logic} \to \textit{foundation} \to \textit{MMT}$$

- Promises: high-level results — generic, reusable!
  - intuitions, documentation, teaching
  - definitions, meta-theorems
  - algorithms, implementations
  - knowledge management
- Dangers: loss of precision — general abstract nonsense?
  - how useful are the abstract results?

    are the deep results foundation-specific?
  - how much work (if any) is needed for specialization?

    hide the framework from the user

MMT Research Hypothesis

abstraction pays off

MMT Research Hypothesis

abstraction pays off

- Representation language Yes!
  - few ontological primitives — MMT language
  - implemented in elegant data structures — MMT system

MMT Research Hypothesis

abstraction pays off

- Representation language Yes!
  - few ontological primitives — MMT language
  - implemented in elegant data structures — MMT system
- Knowledge management services Yes!
  - editing, parsing
  - change management
  - project management, distribution
  - search, querying
  - interactive browsing

MMT Research Hypothesis

abstraction pays off

- Representation language — Yes!
  - few ontological primitives — MMT language
  - implemented in elegant data structures — MMT system
- Knowledge management services — Yes!
  - editing, parsing
  - change management
  - project management, distribution
  - search, querying
  - interactive browsing
- Logical services?
  - module system — Yes!
  - type reconstruction — Yes?
  - computation — current work
  - theorem proving — future work

Features of MMT

few primitives . . . that unify different domain concepts

JaT judgments as types, proofs as terms

unifies expressions and derivations

Features of MMT

few primitives ... that unify different domain concepts

JaT judgments as types, proofs as terms

unifies expressions and derivations

HOAS higher-order abstract syntax unifies operators and binders

Features of MMT

few primitives ... that unify different domain concepts

JaT judgments as types, proofs as terms
unifies expressions and derivations

HOAS higher-order abstract syntax unifies operators and binders

CoT category of theories unifies logical theories, logics, foundations

- languages as theories
- relations as theory morphisms

Features of MMT

few primitives ... that unify different domain concepts

JaT judgments as types, proofs as terms

unifies expressions and derivations

HOAS higher-order abstract syntax unifies operators and binders

CoT category of theories unifies logical theories, logics, foundations

- languages as theories
- relations as theory morphisms

MS module system (little theories)

unifies inheritance and representation theorems

Features of MMT

few primitives . . . that unify different domain concepts

JaT judgments as types, proofs as terms
unifies expressions and derivations

HOAS higher-order abstract syntax unifies operators and binders

CoT category of theories unifies logical theories, logics, foundations

- languages as theories
- relations as theory morphisms

MS module system (little theories)
unifies inheritance and representation theorems

MaM models as morphisms (categorical logic)
unifies syntactical translations and semantic interpretations

Features of MMT (2)

- current work: declaration patterns
  unifies declarations and extension principles
- current work: induction, coinduction
  unifies multiple constructions/reasoning principles
- current work: reflection
  unifies meta- and object level

  for example, module system
  - meta-level: MMT theories
  - object-level: record types

The MMT System

Application-independence

1. data structures
2. logical and knowledge management services
3. individual applications

The MMT System

Application-independence

1. data structures
2. logical and knowledge management services
3. individual applications

Advantages

- flexibility
- no compromises, hacks
- high code reuse

Disadvantages

- no running system bad for talks like this one
- starting MMT gives only an empty environment

  no single name defined

Implementing Services in MMT

- Isolate functionality into services
- Integrate interfaces with core
- Then do 2 implementation approaches
  - plugin interfaces — arbitrary implementations
  - generic implementation — parametrized as declaratively as possible

Logical Services Example: Type reconstruction

- type reconstruction
  - input: judgment with unknown variables

$$\lambda_{n:?}\lambda_{l:?}\texttt{cons}\ ?\ c\ l \qquad \Leftarrow \qquad \Pi_{n:?}\mathit{list}\ n \to ?$$

  - output: derivation of judgment and solution for variables

Logical Services Example: Type reconstruction

- type reconstruction
  - input: judgment with unknown variables

$$\lambda_{n:?}\lambda_{l:?}\texttt{cons}\ ?\ c\ l \quad \Leftarrow \quad \Pi_{n:?} list\ n \to ?$$

  - output: derivation of judgment and solution for variables
- plugin implementation
  - Twelf does type reconstruction for an LF file, exports as MMT
  - MMT module system added to Twelf

1 month full time work

Logical Services Example: Type reconstruction

- type reconstruction
  - input: judgment with unknown variables

$$\lambda_{n:?}\lambda_{l:?}\texttt{cons}\ ?\ c\ l \quad \Leftarrow \quad \Pi_{n:?} list\ n \to ?$$

  - output: derivation of judgment and solution for variables
- plugin implementation
  - Twelf does type reconstruction for an LF file, exports as MMT
  - MMT module system added to Twelf

1 month full time work

- generic implementation
  - parametrized by sets of rules — ~ 8 rule types
  - origin of rules up to plugins

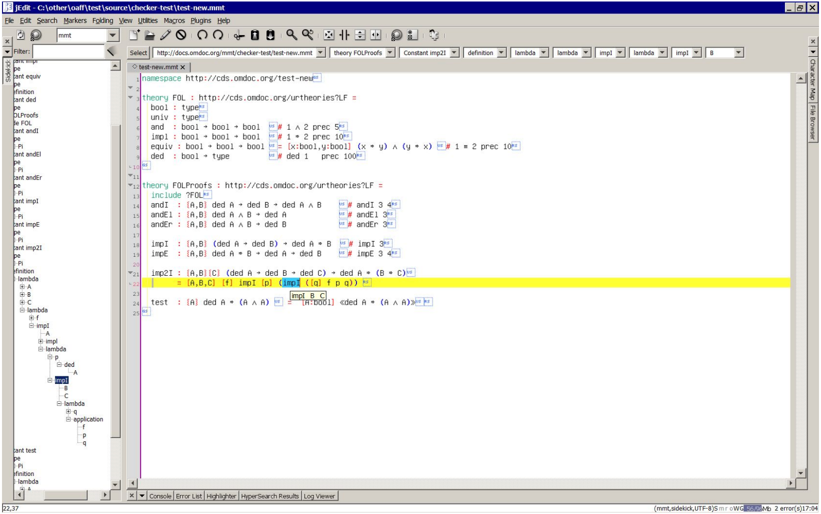
LF plugins provides ~ 10 rules, each a few lines of code

Application Example: Editing

- IDE like based on MMT projects
- jEdit text editor with MMT as plugin
- Generic default implementations for parsing
  - outer syntax: extensible through keyword handlers
  - inner syntax: extensible through notation language
- Cross-references MMT data structures $\leftrightarrow$ source locations
  - outline view
  - hyperlinks (= click on operator, jump to declaration/definition)
  - context-sensitive auto-completion: show identifiers that
    - are in scope
    - have the right type

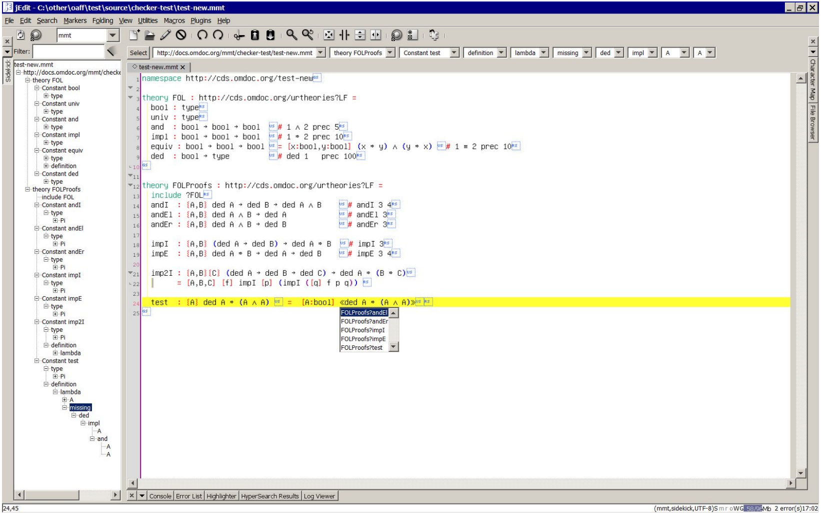
Application Example: Editing

Example feature: pop up shows reconstructed arguments



Application Example: Editing

Example feature: auto-completion shows only identifiers that are in scope and have the right type



Application Example: LaTeX Integration

- Unified document format LaTeX + MMT
- Processed by LaTeX
- MMT-relevant aspects represented in special macros
  sent to MMT via HTTP during compilation
- LaTeX queries MMT at run time via HTTP
  1. parse
  2. type reconstruct
  3. generate high-quality LaTeX cross-references, tooltips

Application Example: LaTeX Integration

Example feature

- upper part: LaTeXsource 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

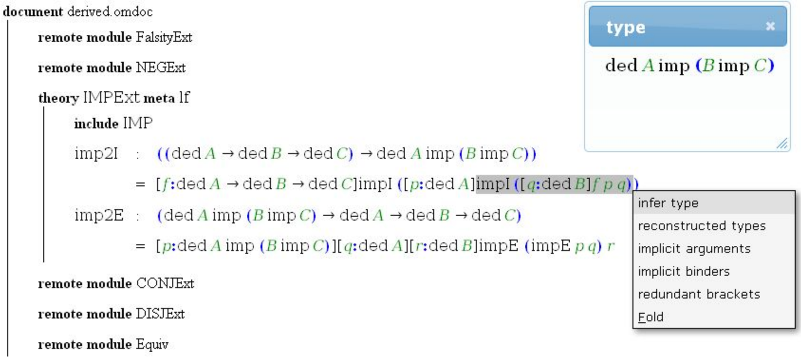
- M is a sort, called the universe.
- $\circ$ is a binary function on M.
- e is a distinguished element of M, the unit.

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 $e \circ x =_M x$.

Application example: Interactive Browsing

- MMT API exposed through HTTP server
- Javascript/Ajax for interactive browsing of MMT projects
  e.g., definition lookup, dynamic type inference
- Interactive graph view
- Immediate editing ongoing work



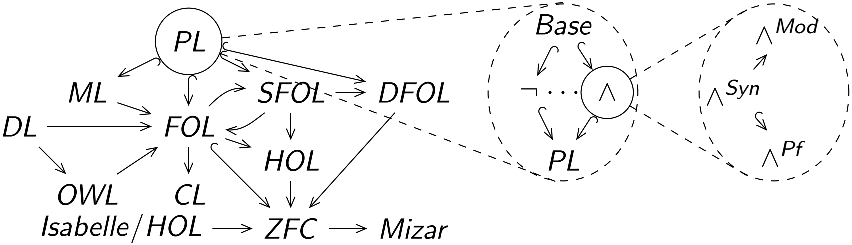
The LATIN Library

- Joint project between DFKI Bremen and Jacobs Univ. Bremen
- Development of an atlas of logics and logic translations
  - reference catalog of standardized logics
  - documentation platform
- All parts of a logic represented in MMT/LF
- Easy part
  - Logical syntax proof theory as MMT/LF theories
  - Judgments as types, higher-order abstract syntax
- Hard part
  - Foundations of mathematics as LF signatures
  - Models as morphisms from the syntax to the foundation

Current State

- 700 little theories including
  - propositional, (unsorted, sorted, dependently-sorted) first-order, higher-order, common, modal, description, linear logic
  - λ-cube, Curry and Church-style type theories
  - ZFC set theory, Mizar's set theory, Isabelle/HOL
  - category theory
- 500 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

Little Theories in LATIN



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., $univ : set$, $nonempty : true\,(univ \neq \emptyset)$

$$\begin{tikzcd}
 & L^{pf} \\
L^{syn} \arrow[ur] \arrow[dr] & \\
 & L^{mod} \\
 & \mathcal{F} \arrow[u, hook]
\end{tikzcd}$$

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., $univ : set$, $nonempty : true\ (univ \neq \emptyset)$

$$\begin{tikzcd}
 & L^{pf} \\
L^{syn} \arrow[ur] \arrow[dr] & \\
 & L^{mod} \\
 & \mathcal{F} \arrow[u, hook]
\end{tikzcd}$$

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., $univ : set$, $nonempty : true\ (univ \neq \emptyset)$

$$
\begin{tikzcd}
 & L^{pf} \\
L^{syn} \arrow[ur] \arrow[dr] & \\
 & L^{mod} \\
 & \mathcal{F} \arrow[u, hook]
\end{tikzcd}
$$

Example

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

$$
\begin{tikzcd}
& FOL^{Pf} \\
FOL^{Syn} \arrow[ur, hook] \arrow[dr, "FOL^{mod}"'] & \\
& FOL^{Mod} \\
& ZFC \arrow[u, hook]
\end{tikzcd}
$$

Representing Logics and Models

$$\begin{tikzcd}
\mathcal{F} \arrow[r, "id_{\mathcal{F}}"] \arrow[d, hook] & \mathcal{F} \\
L^{Mod} \arrow[r, hook] & \Sigma^{Mod} \arrow[u, "M"'] \\
L \arrow[u, "L^{mod}"'] \arrow[r, hook] & \Sigma \arrow[u, "\Sigma^{mod}"']
\end{tikzcd}$$

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

$$
\begin{tikzcd}
\mathcal{F} \arrow[r, "id_{\mathcal{F}}"] \arrow[d, hook] & \mathcal{F} \\
L^{Mod} \arrow[r, hook] & \Sigma^{Mod} \arrow[u, "M"'] \\
L \arrow[u, "L^{mod}"'] \arrow[r, hook] & \Sigma \arrow[u, "\Sigma^{mod}"']
\end{tikzcd}
$$

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

$$
\begin{tikzcd}
\mathcal{F} \arrow[r, "id_{\mathcal{F}}"] \arrow[d, hook] & \mathcal{F} \\
L^{Mod} \arrow[r, hook] & \Sigma^{Mod} \arrow[u, "M"'] \\
L \arrow[u, "L^{mod}"'] \arrow[r, hook] & \Sigma \arrow[u, "\Sigma^{mod}"']
\end{tikzcd}
$$

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 $\mathcal{F}$

Conclusion

- Very general, customizable framework — goal: universal
- Foundation-independent representation language — integrates best primitives
- Interface for logical and knowledge management services
- Rapid prototyping logic systems — scalable
- Interesting for
  - less well-supported logics
  - new, changing logics
  - generic applications/services
  - system integration/combination