Work-in-progress: An MMT-Based User-Interface

Mihnea Iancu and Florian Rabe

Jacobs University Bremen

UITP 2012

MMT https://trac.kwarc.info/MMT

- Prototypical declarative language

  theories, morphisms, declarations, expressions

  module system
- OMDoc/OpenMath-based XML syntax with Scala-based API
- Foundation-independent
  - no commitment to particular logic or logical framework

    both represented as MMT theories themselves
  - concise and natural representations of wide variety of formal systems virtually all of them

Example: small scale

- **Little theories**: state every definition/theorem/algorithm in the smallest possible theory
- Extended to **little logics** and **little logical frameworks**

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

sig Logic { meta LF, form: type, ded : form → type }
sig FOL { meta LF,
  include Logic,
  term: type. ∧: form → form → form, ...
}

sig Magma { meta FOL, ∘: term → term → term }
⋮
sig Ring {meta FOL,
  additive: CGroup,
  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 in the graph serves as root for theory graph in that logic

demo

MMT Vision

- Universal framework for mathematical-logical content
- Close relatives
  - LF, Isabelle: but more universal, more MKM support, more system integration
  - OMDoc/OpenMath: but formal semantics, more automation support
- 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 MKM services

     handles system integration

Applications

- No competitor yet for dedicated “first-tier” systems

  Isabelle, Mizar, Coq
- For the community
  - experimental languages
  - new languages
  - small communities
  - “systems where an emacs mode is the state of the art”
- For me
  - logic and even logical framework in flux
  - need to experiment
  - want to evolve logic and UI independently

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 API
5. Define plugin interfaces to supply the logic-specific aspects
6. Repeat

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 API
5. Define plugin interfaces to supply the logic-specific aspects
6. Repeat

So far no theorem prover (except humans!)

So what are we doing at UITP?

- UI and TP notoriously hard to integrate
- Strength of MMT in the intersection: the data structures
- Implicit claim in MMT project:

  > *Investment in getting the data structures right eventually benefits*
  >
  > - *MKM services*
  > - *logical services*
  > - *user interfaces*

- Evaluation long-term endeavor
- So far
  - MKM services: very positive results
  - logical services, user interfaces: promising outlook

MMT Design, so far

- Foundation-independent MKM aspects
  - abstract syntax for theories, declarations, expressions
  - module system, canonical identifiers
  - notation-based presentation MKM 2008
  - interactive browsing MKM 2009
  - database MKM 2010
  - archival, project management MKM 2011
  - foundations of system integration Calculemus 2011
  - change management Friday, AISC 2012
  - querying MKM 2012
  - extension principles MKM 2012
- Foundation-specific interfaces
  - parsing of files or expressions (e.g., Twelf, TPTP, Mizar, OWL)
  - type checking of abstract syntax (e.g., LF)

MMT Design, current/future work

So far: MMT as a background and MKM system

- content developed using dedicated foundations
- foundation-specific plugins treated as black boxes
- plugins often wrappers around external tools

decent support in user interface

Next: open up black boxes

- generic parser customized by notations
- generic type-checker customized by rules
- generic computation engine customized by rules or code snippets
- generic theorem prover customized by plugins

much better support in user interface

MMT Design, current/future work

So far: MMT as a background and MKM system

- content developed using dedicated foundations
- foundation-specific plugins treated as black boxes
- plugins often wrappers around external tools

decent support in user interface

Next: open up black boxes

- generic parser customized by notations
- generic type-checker customized by rules
- generic computation engine customized by rules or code snippets
- generic theorem prover customized by plugins

much better support in user interface

MMT Design, current/future work

So far: MMT as a background and MKM system

- content developed using dedicated foundations
- foundation-specific plugins treated as black boxes
- plugins often wrappers around external tools

decent support in user interface

Next: open up black boxes

- generic parser customized by notations
- generic type-checker customized by rules
- generic computation engine customized by rules or code snippets
- generic theorem prover customized by plugins

much better support in user interface

MMT Design, current/future work

So far: MMT as a background and MKM system

- content developed using dedicated foundations
- foundation-specific plugins treated as black boxes
- plugins often wrappers around external tools

decent support in user interface

Next: open up black boxes

- generic parser customized by notations
- generic type-checker customized by rules
- generic computation engine customized by rules or code snippets
- generic theorem prover customized by plugins

much better support in user interface

MMT Design, current/future work

So far: MMT as a background and MKM system

- content developed using dedicated foundations
- foundation-specific plugins treated as black boxes
- plugins often wrappers around external tools

decent support in user interface

Next: open up black boxes

- generic parser customized by notations
- generic type-checker customized by rules
- generic computation engine customized by rules or code snippets
- generic theorem prover customized by plugins

much better support in user interface

MMT Design, current/future work

So far: MMT as a background and MKM system

- content developed using dedicated foundations
- foundation-specific plugins treated as black boxes
- plugins often wrappers around external tools

decent support in user interface

Next: open up black boxes

- generic parser customized by notations
- generic type-checker customized by rules
- generic computation engine customized by rules or code snippets
- generic theorem prover customized by plugins

much better support in user interface

Two User Interfaces

Editing

- author-oriented
- local text editor (jEdit)
- jEdit plugin based on MMT API

Browsing

- reader-oriented
- MMT API acts as web server
- interaction through browser via Javascript, Ajax

Side remark: Do we need both? Should they be integrated? How?

Editing: Envisioned Architecture

Pipeline

1. structure parsing (outer syntax)

   abstract syntax with some unparsed strings
2. refine by object parsing: generic parser using notations

   result may be ill-typed
3. refine further: type reconstruction, computation, theorem proving

Principles

- unified internal representation
  - cross-linked to source locations
  - exposed to plugins, user interface
- separate compilation (module system), change management
- internal representation
- provenance tracking for refinement operations

Editing: Envisioned Architecture

Pipeline

1. structure parsing (outer syntax)

   abstract syntax with some unparsed strings
2. refine by object parsing: generic parser using notations

   result may be ill-typed
3. refine further: type reconstruction, computation, theorem proving

Principles

- unified internal representation
  - cross-linked to source locations
  - exposed to plugins, user interface
- separate compilation (module system), change management
- internal representation
- provenance tracking for refinement operations

Editing: Current State

Structure Parsing: done

- Fast, (essentially) never fails, local (no loading of other files)
- Produces valid MMT data structures
- Sufficient for
  - outline view
  - context-sensitive auto-completion (suggest only identifiers that are in scope
  - tool tips (hover over operator, see (e.g.) qualified and origin
  - hyperlinks (= click on operator, jump to declaration/definition)
  - file and theory level dependency management

Current/ongoing work

- Term parsing, type reconstruction, computation: going well
- Theorem proving: still in surveying phase

Editing: Current State

Structure Parsing: done

- Fast, (essentially) never fails, local (no loading of other files)
- Produces valid MMT data structures
- Sufficient for
  - outline view
  - context-sensitive auto-completion (suggest only identifiers that are in scope
  - tool tips (hover over operator, see (e.g.) qualified and origin
  - hyperlinks (= click on operator, jump to declaration/definition)
  - file and theory level dependency management

Current/ongoing work

- Term parsing, type reconstruction, computation: going well
- Theorem proving: still in surveying phase

Editing: Current State

Structure Parsing: done

- Fast, (essentially) never fails, local (no loading of other files)
- Produces valid MMT data structures
- Sufficient for
    - outline view
    - context-sensitive auto-completion (suggest only identifiers that are in scope
    - tool tips (hover over operator, see (e.g.) qualified and origin
    - hyperlinks (= click on operator, jump to declaration/definition)
    - file and theory level dependency management

Current/ongoing work

- Term parsing, type reconstruction, computation: going well
- Theorem proving: still in surveying phase

Editing: Current State

Structure Parsing: done

- Fast, (essentially) never fails, local (no loading of other files)
- Produces valid MMT data structures
- Sufficient for
  - outline view
  - context-sensitive auto-completion (suggest only identifiers that are in scope
  - tool tips (hover over operator, see (e.g.) qualified and origin
  - hyperlinks (= click on operator, jump to declaration/definition)
  - file and theory level dependency management

Current/ongoing work

- Term parsing, type reconstruction, computation: going well
- Theorem proving: still in surveying phase

demo

Browser Interface

- MMT API exposed through HTTP server
- Javascript/Ajax for interactive browsing of MMT projects

  e.g., dynamic type inference
- Interactive graph view
- Immediate editing ongoing work not totally sure what for

demo

Conclusion

- MMT: rapid prototyping logic systems
- user interface making good progress
- theorem prover still future work but considered in the design
- Interface no competitor of dedicated systems yet
- But interface already good for
  - less well-supported logics
  - new logics
  - changing logics