A Logic-Independent IDE

Florian Rabe

Computer Science, Jacobs University Bremen

UITP @ FLoC 2014

MMT+jEdit = Logic-Independent IDE

- MMT
  - prototypical declarative language
  - 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
    - focus on customizable, reusable services
  - written in Scala
- jEdit
  - mature general purpose text editor
  - written in Java
- MMT IDE
  - jEdit plugin that bundles MMT API
  - relatively thin glue layer between MMT and jEdit
    only $\sim$ 1000 loc

Background: MMT

- Attempt at a universal framework for formal knowledge and its semantics
- MMT language
  - prototypical formal declarative language
  - foundation-independent: no commitment to particular logic or type theory
    no built-in operators at all
  - admits concise representations of most formal systems
    logics, specification languages, ontology languages, ...
  - continuous development since 2006 (with Michael Kohlhase)
  - >100 pages of publication
- MMT system
  - API and services
  - continuous development since 2007 (with >10 students)
  - $>30,000$ lines of Scala code
  - >10 papers on individual aspects

Small Example

Logical frameworks in MMT:

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

Logics in MMT/LF:

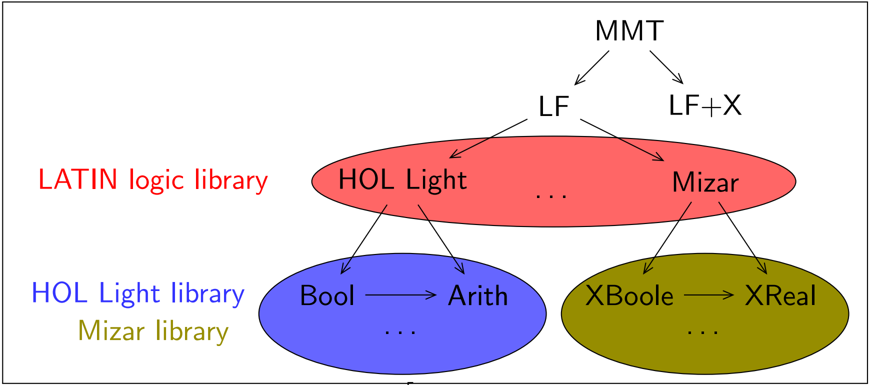
```
theory Logic : LF {o: type, ded : o → type }
theory FOL   : LF {
  include Logic
  u: type. imp: o → o → o, ...
}
```

Algebraic theories in MMT/LF/FOL:

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

Big Picture: The OAF Project

- Open Archive of Formalizations 2014-2017
- Logic formalizations in LF (or variants as necessary)
- Import proof assistant libraries
  joint theory graph for HOL Light, Mizar, Coq, ...
- stepping stone towards library integration



Foundation-Independence

- Practical systems often foundation-specific
  - fixed logical foundational — e.g., CIC
  - fixed kernel implementation for it — e.g., Coq
  - as many features on top as developer community can afford — often a bottleneck
- Effect
  - slow evolution
  - isolated systems
  - hard to get new systems to meaningful scale
- MMT approach
  - foundation-independent wherever possible
  - develop generic solutions at MMT level
  - Very similar to logical framework — but even more general

MMT Design Methodology

1. Choose a typical problem
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

Applied to

- knowledge management features
  e.g., search, querying, browsing, change management
- logical features e.g., module system, type reconstruction
- Here: IDE

Kernel-UI Interface

- Kernel implementation of logic

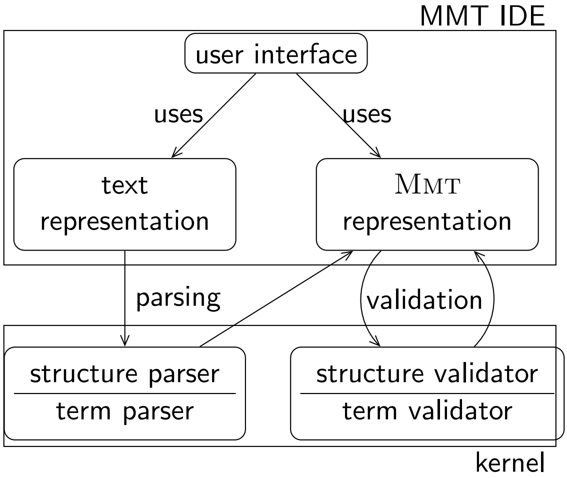
  originally often read-eval-print style loop
- Not good for modern UI standards

  various work towards better kernel-UI integration
- Central idea of MMT IDE
  - use MMT data structures for knowledge representation

    shared by kernel and UI
  - use jEdit as UI framework
  - design abstract interface for kernel functionality

    not a goal to work with any existing kernel

Overview


MMT IDE
user interface
uses
uses
text
representation
MMT
representation
parsing
validation
structure parser
term parser
structure validator
term validator
kernel

Abstract Kernel

2×2 kernel operations	Parsing	Validation
Structure		
Terms		

- Structure parsing
  - parses only outer syntax

    e.g., very fast, e.g., run on every keystroke
  - leaves terms as strings
- Term parsing
  - parsing units generated by structure parser
  - called at the liberty of the UI

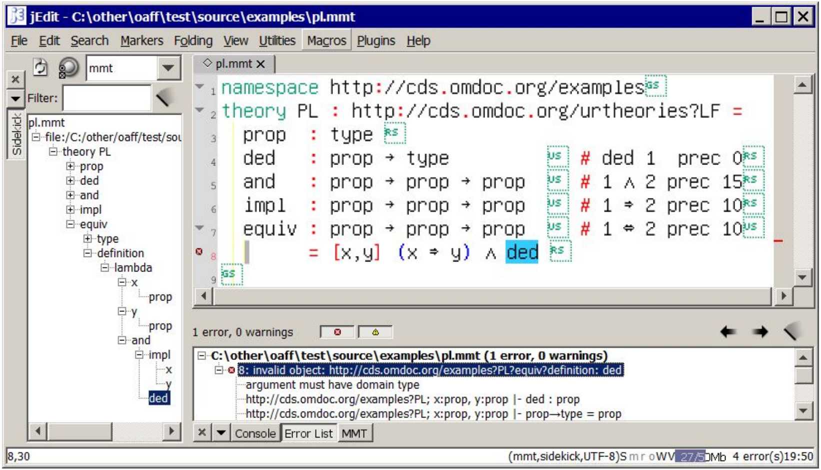
    e.g., change management: only reparse on change
- Structure validation
  - identifier scopes
  - theory graph
- Term validation
  - validation units generated by structure validator
  - type reconstruction, proof obligations, etc.

    change management, parallelization

Content-Presentation Cross-References

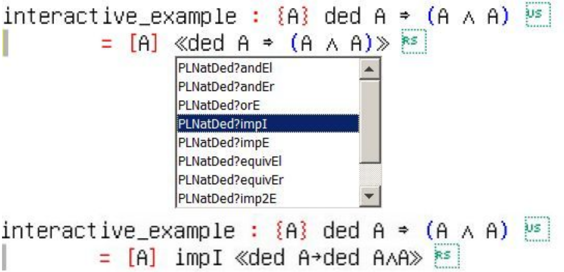
- Structure and term parser should return source regions
  detailed cross-references data structures $\longleftrightarrow$ buffer
- Outline view: side bar shows syntax tree of document
  to the extent parsed/validated
- Joint focus, selection of subterms
- Tool tips show qualified identifiers, implicit arguments, ...
- Hyperlinks CTRL-click on operators

IDE: Example View



Auto-Completion

- Show only identifiers that are in scope
- If needed type is know, show only identifiers whose return type unifies



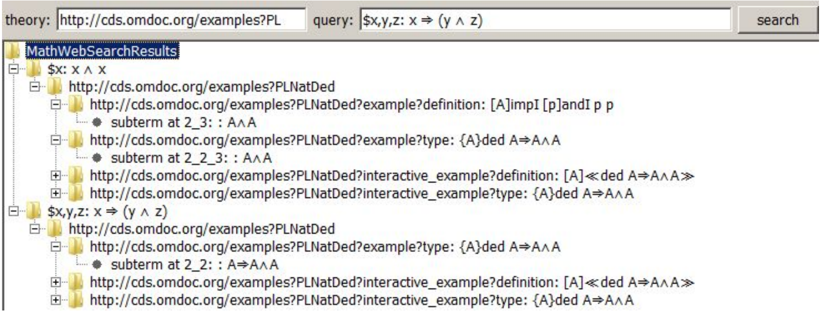
Type Inferece

- Dynamic type inference of selected subterm
- Shown as tool tip

```
equivI : {A,B} (ded A → ded B) → (ded B → ded A) → ded A ⇔ B
       = [A,B,p,q] andI (impI [a] p a) (impI [b] q b)
                         ded A⇒B
```

Search

- Substitution tree index for a whole library
- Hosted on remote server Kohlhase et al., MathWebSearch
- Highly optimized for large libraries
- Index produced by MMT
- Queried from within UI



Change Management

- 2-dimensional dependency relation
  1. for each term, dependency between
     - string
     - parsed
     - validated
  2. between validation units
     - type of any declaration
     - definiens (= proof) of any declaration
- Dependencies tracked by MMT
- Changing a term triggers
  - reparse
  - revalidate
  - revalidate all depending validation units

Structure Parser

- Keyword-based
- ASCII characters 28-31 as delimiters
- Works generically at MMT level
- Further customization possible
  - plugins register individual keywords and handlers

Term Parser

- Notation-based
- Uses MMT notations that are in scope
- Works generically at MMT level
- Adds meta-variables for unknown subterms

  implicit arguments, omitted types
- Customization implied based on notations

Structure Validator

- Implements structural semantics of MMT
- Break declarations into proof obligations
- Example: $c : A = t$ generates
  - validity check of A
  - type check of t against A
- Change management
  - if term validator returns dependencies, JMMT revalidates only when needed
  - t changes much more often than A
  - checking t (= proofs) and A (= assertion) separately splits their dependency

Term Validator

- Rule-based
- Type reconstruction
  - solves unknown meta-variables inserted by term parser
  - returns dependencies
- Customized by inference rules provided by plugins
- Several LF-based instances
  - module system
  - shallow polymorphism
  - literals
  - modulo

Conclusion

- MMT: rapid prototyping logic systems
- Generic IDE making good progress
- Currently, no competitor yet for dedicated "first-tier" systems
  - no native theorem proving support in MMT
  - no connection of abstract kernel interface and existing proof assistant

  should be tried, but not on my personal critical path
- Promising for less well supported systems
  - experimental languages
  - new languages
  - small communities