MMT Objects

Florian Rabe

Jacobs University, Bremen, Germany

OpenMath 2014
Overview

- Major **OPENMath**-based experiment/system
- MMT
  - Universal representation language for formal logical content inspired by **OPENMath**, **OMDoc**
  - Implementation with generic support for logical and knowledge management functionality e.g., module system, type reconstruction; presentation, editing
- Object layer uses **OPENMath** as primary data structure
Point of This Talk

- Describe differences between MMT objects and OpenMath objects
- Provide additional information for further development of OpenMath
- Not a
  - position paper
  - standard enhancement proposal

MMT’s deviations may or may not be good for OpenMath
Grammars

c: reference to symbol/constant (OMS)
x: reference to variable (OMV)

**OpenMath**

objects

\[ O ::= \mathcal{I}(i) \mid \mathcal{F}(f) \mid S(s) \mid \mathcal{B}\mathcal{A}(b) \]
\[ \mid c \mid x \]
\[ \mid A(O, O^*) \mid \mathcal{A}\mathcal{T}\mathcal{T}(O; KV^*) \]
\[ \mid B(O; \mathcal{A}\mathcal{T}\mathcal{T}(x; KV^*)^*; O) \mid E(c; O^*) \]

(key-values lists)

\[ KV ::= c \leftrightarrow O \]

**Mmt**

objects

\[ E ::= \mathcal{L}^c(s) \]
\[ \mid c \mid x \]
\[ \mid c(\gamma; \Gamma; E^*) \]

contexts

\[ \Gamma ::= (x[: E][= E])^* \]

substitutions

\[ \gamma ::= (x = E)^* \]
**OpenMath**

- 4 fixed literal types: integers, float, string, byte array
- Concrete syntax fixed by standard
- Side note: OpenMath standard CDs define no operations on strings or byte arrays

**MMT literals** $\mathcal{L}^c(s)$

- Extensible set of literal types like extensible set of symbols
- No individual literal types built-in
- $c$ is symbol whose documentation defines
  - Syntax (string encoding)
  - Semantics (valid values and their meaning)

Of string $s$, which represents the literal value
Attributions

OpenMath
- attributed variables in particular needed for type attributions
- semantically attributed objects does anybody use this?
- ignorable attributions

Mmt: no attributions
- contexts declare variables $x[: E][= E]$ effectively 2 built-in attribution keys
  \[
  \mathcal{ATT}(x; [\text{type} \mapsto T], [\text{def} \mapsto D]) \simeq x[: T][= D]
  \]
- ignorable attributions as extra-linguistic metadata
  somewhat similar to HTML + RDFa
Errors

**OpenMath**
- Explicit error objects

**Mmt**: no errors
- error objects recovered as special case of application objects
Complex Objects

OpenMath

- 4 constructions: attribution of key-value list, error, application, binding
- Note:
  - attribution and binding are purely structural
  - error implies semantic properties
  - application is in between
    - is function application semantics implied or not?

Mmt

- single construction $c(\gamma; \Gamma; \vec{E})$
- purely structural
  - named children $\gamma$
  - bound variables $\Gamma$
  - unnamed children (in scope of bound variables)
- each construction labeled with symbol $c$
- semantics of $c(\gamma; \Gamma; \vec{E})$ defined solely by semantics of $c$
**Complex Objects (2)**

**OpenMath-Mmt correspondence** $O \simeq E$

If

$$O_i \simeq E_i \quad \text{and} \quad V_j \simeq X_j,$$

then for applications:

$$A(c, O_1, \ldots, O_n) \simeq c(\cdot; \cdot; E_1, \ldots, E_n)$$

bindings:

$$B(c; V_1, \ldots, V_m; O_1) \simeq c(\cdot; X_1, \ldots, X_n; E_1)$$

errors:

$$\mathcal{E}(c; O_1, \ldots, O_n) \simeq c(\cdot; \cdot; E_1, \ldots, E_n)$$
What does $\gamma$ do in $c(\gamma; \Gamma; \vec{E})$?

Generalization beyond application and binding objects

Substitution $\gamma$ used for

- named arguments in function application
- records
- list of cases in pattern-match
Conclusion

- **Mmt** grammar uses only 4 productions
  - constants
  - variables
  - literals
  - complex objects
- **OpenMath** uses 10 productions
  - 4 kinds of literals
  - 4 kinds of complex objects
- **Mmt** loses some expressivity, especially for applications
- But gained simplification crucial in **Mmt** implementation