MMT Objects

Florian Rabe

Computer Science, Jacobs University, Bremen, Germany

Abstract

MMT is a mathematical knowledge representation language, whose object layer is strongly inspired by OpenMath. In fact, the first version of the MMT system implemented exactly OpenMath objects.

But over time MMT has evolved and deviated from OpenMath in some respects. Some of these deviations are experimental or specific to MMT applications, but others may be interesting for future versions of OpenMath.

This papers presents MMT objects and discusses the differences to OpenMath objects.

1 Introduction

MMT [RK13] is a formal knowledge representation language following the OMDoc approach [Koh06]. It focuses on the foundation-independent and modular representation of formal mathematical content. Like OMDoc, it subsumes OpenMath as the representation language for formal mathematical objects.

The MMT system [Rab13] coherently implements the MMT data structures along with various logical services (e.g., type reconstruction, simplification), knowledge management services (e.g., presentation, search), and applications (e.g., IDE, library browser).

Notably, the MMT language and system are generic in the sense that they can be easily instantiated with different formal languages such as logics or type theories.

Over 7 years of development, the implementation of objects in MMT has deviated from OpenMath in several ways. The main motivation was to reduce the number of productions in the grammar. This greatly simplifies the development of inductive algorithms and is a key enabling factor to keep the implementation manageable.

In this paper, we present and discuss the differences between OpenMath and MMT objects. We start by presenting the two grammars in Sect. 2. Then we discuss individual differences in Sect. 4, 5, 6. MMT also provides a more precise definition of variable scope and substitution than OpenMath, which we describe in Sect. 3.

2 Overview

Symbols and Variables We write c for a symbol reference $(cdbase, cd, name)$. This is a triple of a URI $cdbase$ and two names of a content dictionary cd and a symbol $name$ defined in cd.

We write x for a variable. This is just a name.

OpenMath Objects OpenMath objects O are formed by the following grammar:

$$\begin{array}{rcl} O & ::= & \mathcal{I}(i) \mid \mathcal{F}(f) \mid \mathcal{S}(s) \mid \mathcal{BA}(b) \\ & \mid & c \mid x \\ & \mid & \mathcal{A}(O, O^*) \mid \mathcal{ATT}(O; KV^*) \mid \mathcal{B}(O; \mathcal{ATT}(x; KV^*)^*; O) \mid \mathcal{E}(c; O^*) \\ KV & ::= & c \mapsto O \end{array}$$

where * is (possibly empty) repetition.

There are 10 different productions for objects:

- 4 literals: integers $\mathcal{I}(i)$, floating point numbers $\mathcal{F}(f)$, strings $\mathcal{S}(s)$, and byte arrays $\mathcal{BA}(b)$,
- 2 name references to symbols (named objects in a global namespace) and variables (named objects in the local context),
- 4 complex objects:
  - application $\mathcal{A}(O, O_1, \ldots, O_n)$ of a function to arguments,
  - attribution $\mathcal{ATT}(O; KV_1, \ldots, KV_n)$ of a key-value list to an object,
  - binding $\mathcal{B}(B; V_1, \ldots, V_n; S)$ of a binder B with attributed variables V_i (which are bound in S) and scope S,
  - errors $\mathcal{E}(c; O_1, \ldots, O_n)$ formed from applying a symbol to arguments.

In the grammar above, we omitted foreign objects. These are arbitrary non-OpenMath data, which may occur as the arguments of an error and the values of a key-value pair.

MMT Objects Mmt objects E are formed by the following grammar:

$$\begin{array}{rcl} E & ::= & \mathcal{L}^c(s) \\ & \mid & c \mid x \\ & \mid & c(\gamma; \Gamma; E^*) \\ \Gamma & ::= & (x[: E][= E])^* \\ \gamma & ::= & (x = E)^* \end{array}$$

There are 4 different productions for objects:

- Literals $\mathcal{L}^c(s)$ consists of a symbol c identifying the type and a string encoding s of the value.
- Symbols c and variables x are as for OpenMath.
- Complex objects $c(\gamma; \Gamma; E_1, \ldots, E_n)$ are formed from a head symbol c, a substitution γ, a context Γ (whose variables are bound in the E_i), and a list of arguments E_i.

In addition to objects, MMT defines two additional concepts: contexts and substitutions.

Contexts Γ are lists of variable declarations $x[: T][= D]$ where the type T and the definiens D are optional. Substitutions γ are lists of assignments $x = E$.

3 Scoping

We will use the notation X_i for an MMT variable declaration $x_i[: T_i][= D_i]$

Objects in Context OPENMATH does not clarify whether bound variables may occur in attributions of other bound variables of the same binding.

MMT formalizes this as follows. A variable x is **free** in E if it occurs in E without being bound. And in $c(\gamma; X_1, \ldots, X_m; E_1, \ldots, E_n)$, any occurrence of x_i in $T_{i+1}, D_{i+1}, \ldots, T_m, D_m, E_1, \ldots, E_n$ is **bound**.

In particular, MMT allows every bound variable to occur in the type and definiens of later variables of the same binding. Note that this includes the degenerate case of a context $x, x : x$, which declares two variables of the same name with the first one occurring in the type of the second one.

A context $\Gamma = X_1, \ldots, X_m$ is **well-formed** if every T_i and D_i (if given) is well-formed in context $X_1, \ldots, X_{i-1}$. And an object E is **well-formed** in context Γ if all free variables of E are declared in Γ.

We further define that if $x_i = x_j = x$ for $i < j$, i.e., multiple variables in Γ have the same name, then all free occurrences of x in E are occurrences of x_j. Intuitively, the declaration of x_j shadows the one x_i. This is irrelevant for well-formedness but matters when looking up the type or definiens of x in Γ.

Substitution Well-formedness relative to a context permits a formal definition of substitution. γ of the form $x_1 = E_1, \ldots, x_m = E_m$ is a **well-formed** substitution from $\Gamma = X_1, \ldots, X_m$ to Γ' if every E_i is well-formed in context Γ'.

In that case, if E is well-formed in context Γ, we write $E[\gamma]$ for the object arising from replacing every free occurrence of x_i in E with E_i. Then $E[\gamma]$ is well-formed in context Γ'.

As usual, we assume substitution to be capture-avoiding. This, however, means that $E[\gamma]$ is under-specified because the necessary α-renaming of the bound variables of E is not specified. The MMT implementation α-renames bound variables only when necessary and then does so by appending distinguished characters to the name.

4 Attributions

Attributed Variables MMT contexts can be seen as a list of attributed variables restricted to 2 distinguished keys for type and definiens.

Let `type` and `def` be special symbols. If we have the correspondence

$$O \quad \simeq \quad E \quad \text{and} \quad O' \quad \simeq \quad E'$$

between OpenMath and Mmt objects, then

$$\mathcal{ATT}(x; [\texttt{type} \mapsto O], [\texttt{def} \mapsto O']) \quad \simeq \quad x[: E][= E']$$

and accordingly for lists of attributed OpenMath variables corresponding to Mmt contexts.

Effectively, Mmt allows only 2 distinguished keys when attributing variables. This is a major limitation compared to OpenMath, which allows any symbol as a key. However, to the author's knowledge, type and definiens attributions account for the vast majority of variable attributions in practice.

At the same time, building type and definiens into the grammar explicitly tremendously simplifies the grammar and arguably makes the treatment of bound variables more intuitive.

Attribution Objects Mmt does not allow for attribution objects other than the limited variable declarations from above.

However, OpenMath objects like $\mathcal{ATT}(O; k \mapsto V)$ for a semantic attribution key k can often be represented as $\mathcal{A}(k, O, V)$. Therefore, this might not be a critical limitation in practice.

Ignorable Attributions The above does not cover non-semantic (i.e., ignorable) attributions.

However, the Mmt implementation anyway permits attaching metadata to any object (including subobjects). Mmt metadata is orthogonal to the syntax of objects and includes URI-object pairs (which subsumes OpenMath attributions) as well as RDF-style typed links and tags.

Such extra-linguistic attributions may be preferable in practice because they make it much easier for applications to ignore the ignorable attributions (e.g., during pattern-matching).

5 Complex Objects

Mmt uses only one constructor for complex objects. The basic idea is that every inner node of an Mmt syntax tree is labeled with a symbol, which defines the meaning of the subtree.

In particular, the syntactic constraints and intended semantics of a complex Mmt object are completely relegated to the head symbol (i.e., specified in the content dictionary in which the head symbol is declared). Note that this is similar to OpenMath binding and attribution objects, but different from OpenMath application objects (for which OpenMath does not prescribe but strongly suggests function application as the intended semantics).

Complex Mmt objects subsume most OpenMath application and binding and all error objects as follows. If

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

then

$$\begin{aligned}\mathcal{A}(c, O_1, \ldots, O_n) &\simeq c(\cdot;\cdot; E_1, \ldots, E_n)\\ \mathcal{B}(c; V_1, \ldots, V_m; O_1) &\simeq c(\cdot; X_1, \ldots, X_n; E_1)\\ \mathcal{E}(c; O_1, \ldots, O_n) &\simeq c(\cdot;\cdot; E_1, \ldots, E_n)\end{aligned}$$

Here we write $\cdot$ for the empty substitution, context, or argument list.

5.1 Limitations

A major limitation of MMT objects is that the first child of application and binding objects is assumed to be a symbol.

Application Objects This limitation is quite severe for application objects, where it excludes objects $\mathcal{A}(O, O_1, \ldots, O_n)$ where O is, e.g., a variable or a λ-abstraction.

In the setting of MMT, this is acceptable because MMT assumes that such application objects are anyway only meaningful in the presence of a formal language that defines the meaning of functions, e.g., a λ-calculus.

Representations of λ-calculi in MMT are most natural if they provide three symbols `arrow`, `lambda`, and `apply`. In that case, application objects with complex functions can be written in MMT as $\texttt{apply}(\cdot;\cdot; E, E_1, \ldots, E_m)$.

However, for OPENMATH in general, it is much less clear whether such a limitation would be reasonable.

Binding Objects The limitation is less severe for objects $\mathcal{B}(O; V_1, \ldots, V_m; O_1)$ because O is usually a symbol anyway.

Occasionally, it is useful to allow binders where $O = \mathcal{A}(c, o_1, \ldots, o_l)$, e.g., for a definite integral $\int_a^b$ with $O = \mathcal{A}(\int, a, b)$. In MMT, these arguments can be represented by using the substitution γ as in

$$\mathcal{B}(\mathcal{A}(\textstyle\int, a, b); x; f(x)) \simeq \textstyle\int(\texttt{from} = a, \texttt{to} = b; x; f(x))$$

Arguably, the MMT representation is even more natural than the OPENMATH representation in this example because it makes all arguments available at the toplevel of the object.

Error Objects The first child of an OPENMATH error object is a symbol anyway so that the representation in MMT does not limit expressivity.

However, it has the effect that applications and errors are represented in the same way. This may be seen as a limitation because the syntactical distinction between error and application objects is lost. For example, MMT makes it harder to spot errors and to separate errors from the mathematically meaningful objects.

On the other hand, the syntactic distinction employed by OPENMATH can also cause additional work. For example, in many programming languages exceptions are normal values that can, e.g., be passed as arguments to functions.

In the setting of MMT, more emphasis is placed on the head symbol of a complex object anyway. Thus, it remains reasonably easy to distinguish errors from non-error objects by inspecting the type or role of the head symbol.

5.2 Generalizations

Complex MMT objects are more general than OPENMATH objects in 2 ways: by allowing 0 or more scopes in a binder and by adding the substitution γ.

Scopes The generalization to 0 or more scopes in binding objects has already been proposed for future versions of OPENMATH in [DK09, Hel13].

Substitution There are various situations where the substitution γ is useful. For example, it can be used to represent reified substitutions (e.g., in an explicit substitution calculus), record values (which OPENMATH does not support naturally), or named arguments (instead or in addition to unnamed positional arguments as used in OPENMATH application objects).

Examples The following gives example uses for the various combinations of substitution, context, and argument list in an complex MMT object:

None: Objects of the form $c(\cdot;\cdot;\cdot)$ can represent the application of a function to 0 arguments. For example, $+(\cdot;\cdot;\cdot)$ is the empty sum (which simplifies to 0). This case is the reason why MMT does not identify c and $c(\cdot;\cdot;\cdot)$ (even though that would save one more production in the grammar).

Substitution: Objects of the form $c(\gamma;\cdot;\cdot)$ can represent record values.

Context: Objects of the form $c(\cdot;\Gamma;\cdot)$ can represent record types, classes, or dependent products.

Argument list: Objects of the form $c(\cdot;\cdot;\vec{E})$ can represent applications in the usual way.

Substitution+Context: Objects of the form $c(\gamma;\Gamma;\cdot)$ can represent matching pairs of a context and a substitution out of it.

Substitution+Argument list: Objects of the form $c(\gamma;\cdot;\vec{E})$ can represent applications with named arguments γ and unnamed positional arguments $\vec{E}$.

Context+Argument list: Objects of the form $c(\cdot;\Gamma;\vec{E})$ can represent bindings (possibly with multiple scopes) in the usual way.

Substitution+Context+Argument list: Objects of the form $c(\gamma;\Gamma;\vec{E})$ can represent bindings where the binder takes additional parameters, e.g., as in the integral example above.

Further Generalizations Some of the above examples would in fact be even more natural in the absence of α-conversion. It may be interesting to investigate whether α-conversion could be made optional in languages like OpenMath or Mmt.

Another possible generalization is to allow the cases of the substitution γ, the declarations in Γ, and the arguments in $\vec{E}$ to occur mixed. The scoping rule would be that every variable is bound in anything to the right of it.

6 Literals

Literals present a challenge to knowledge representation languages like OpenMath because there is no good canonical choice which fixed set of literals to build into the language. OpenMath somewhat arbitrarily fixes 4 types of literals. Thus, it lacks, e.g., bounded or arbitrary-base integer literals or character literals (not to mention less mathematical types like URIs, dates, or colors).

Mmt merges all literals into a single production and keeps the set of literals extensible by content dictionaries. This simplifies the language tremendously while making it more general.

Every literal $\mathcal{L}^c(l)$ value occurs as a string l in the concrete syntax. The symbol c has two functions. Firstly, it defines the mappings between the string l and the actual values. This information should be specified in the content dictionary that defines c. Secondly, it is the type of $\mathcal{L}^c(l)$ in the presence of a type system.

For example, by using four fixed symbols `OMI`, `OMF`, `OMSTRING`, and `OMB`, we can recover OpenMath literals as

$$\begin{aligned} \mathcal{I}(i) &\simeq \mathcal{L}^{\texttt{OMI}}(i') \\ \mathcal{F}(f) &\simeq \mathcal{L}^{\texttt{OMF}}(f') \\ \mathcal{BA}(b) &\simeq \mathcal{L}^{\texttt{OMB}}(b') \\ \mathcal{S}(s) &\simeq \mathcal{L}^{\texttt{OMSTRING}}(s') \end{aligned}$$

where l' denotes an appropriate string-encoding of the literal value l.

Note that we could already use $\mathbb{Z}$ instead of `OMI`, but there seem to be no corresponding symbols for the other 3 literal types in existing content dictionaries.

References

[DK09] J. Davenport and M. Kohlhase. Unifying Math Ontologies: A Tale of Two Standards. In J. Carette, L. Dixon, C. Sacerdoti Coen, and S. Watt, editors, *Intelligent Computer Mathematics*, pages 263–278. Springer, 2009.

[Hel13] L. Hellström. Quantifiers and n-ary binders: an OpenMath standard enhancement proposal. In C. Lange, D. Aspinall, J. Carette, J. Davenport, A. Kohlhase, M. Kohlhase, P. Libbrecht, P. Quaresma, F. Rabe, P. Sojka, I. Whiteside, and W. Windsteiger, editors, *Joint Proceedings of the MathUI, OpenMath, PLMMS, and ThEdu Workshops and Work in Progress at CICM*. CEUR-WS.org, 2013.

[Koh06] M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer, 2006.

[Rab13] F. Rabe. The MMT API: A Generic MKM System. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Intelligent Computer Mathematics*, pages 339–343. Springer, 2013.

[RK13] F. Rabe and M. Kohlhase. A Scalable Module System. *Information and Computation*, 230(1):1–54, 2013.