How is MMT different from System X, Y, Z, . . . ?

Florian Rabe

November 24, 2014*

Abstract

This is an overview of systems and frameworks for mathematics and logic along with high-level descriptions of how they relate to/differ from MMT. [1]

See https://svn.kwarc.info/repos/MMT/doc/introduction/mmt.pdf for an overview of and references to MMT.

The present version of this document is incomplete and will be extended from time to time.

1 Interchange Formats

These are not tools but languages intended as representation formats for other tools, in particular for the interchange of data between systems. They usually come with a suite of tools or libraries for parsing and serializing data. With the exception of MMT, the languages do not define mathematical well-formedness and the parsers cannot verify it.

1.1 General Purpose

XML XML [W3C98] is a format that allows representing any kind of structured data. It is agnostic about the kind of data being represented, and users must define and document an XML language (= a set of XML tags, often called a schema) for each kind of data. When applied to logic tools, it is usually used to represent syntax trees (using one XML tag for each non-terminal symbol) with respect to the context-free grammar underlying the tool.

Most programming languages provide libraries for parsing and serializing XML. Some logic tools offer exports in custom XML languages for exporting their data, e.g., Mizar.

JSON JSON [JSO97] is similar to XML in motivation and applications. Its syntax is less verbose, and its handling of data types is more refined. It is widely used in web applications, but not commonly used in logic tools.

MMT MMT is designed specifically for representing mathematical and logical data, in particular formal theories. It offers an XML language as one representation format, but adds modularity, context-sensitivity, and the representation of semantics.

*The latest version of this document is available at https://svn.kwarc.info/repos/MMT/doc/introduction/mmt-vs-X.pdf.

[1]This document is based on a note by Lambert Meertens.

1.2 Logic-Independent

OpenMath OpenMath [BCC+04] consists of two parts. Firstly, it is an XML language for representing mathematical formulas, called OpenMath objects. These objects are similar to S-expressions except that they add binding, key-value attributions, and a fixed set of literals. The leafs of the S-expressions may be references to symbols, which are introduced in OpenMath content dictionaries.

Secondly, it is a collection of content dictionaries aiming primarily at the fragment of mathematics taught at the high-school level. These content dictionaries declare symbols and describe their meaning and usage. Content dictionaries remain oriented on single mathematical formulas in a fragment of traditional mathematics, and is not usable for representing more abstract mathematics or mathematical theories.

MathML MathML [ABC+03] consists of two parts. Firstly, content MathML is an XML language that is essentially isomorphic to OpenMath (different XML tags but same meaning).

Secondly, presentation MathML is an XML language for the shape of mathematical formulas. For example, where content MathML uses a primitive for "exponentiation applied to x and n", presentation MathML uses a primitive for "x with a superscript n".

Given notations for all symbols, it is possible to translate from content to presentation MathML (although MMT appears to be only tool that implements the general case well); the reverse transformation is AI-complete.

MathML is an official part of HMTL, and many browsers (most notably Firefox) are able to render presentation MathML formulas at LaTeX-level quality.

OMDoc OMDoc [Koh06] is an XML language aiming at representing all of mathematics. It subsumes OpenMath and content MathML as alternatives for representing formulas and presentation MathML for representing semi-formal formulas (i.e., formulas whose content structure is not or only partially known).

It adds primitives for formal theories (theories and morphisms, type declarations, axioms/theorems). It also adds LaTeX-style narrative features such as text, sectioning, lists, and citations.

MMT MMT is both a restriction and an improvement of OMDoc. It is restricted to the formal aspects and excludes the semi-formal and narrative aspects (although this is ongoing work). The XML language that MMT uses is essentially a fragment of OMDoc.

Contrary to OpenMath, MathML, and OMDoc, MMT allows representing the semantics of mathematical objects and theories. For example, MMT defines and can check whether a used symbol is declared and imported into the current scope. If the respective type systems and logics are represented as MMT theories themselves, MMT defines and can check whether objects are well-formed or true.

OpenMath, MathML, and OMDoc focus on the creation of a language standard and relegate the development of tools to users of the standard. MMT additionally provides a reference implementation with a suite of services. For example, MMT theories can provide notations for symbols, and MMT can use these notations to parse text representations and to render content as HTML (including presentation MathML for formulas).

1.3 Logic-Specific

TPTP TPTP [SS98] consists of three parts.

Firstly, it is a text-based Prolog-parsable interchange syntax for a family of related logics. The TPTP syntax is context-free and expressive enough to represent formal theories of a large variety of logics, but only for some logics the exact syntax has been officially specified. Originally this was only untyped first-order logic; later it was extended to typed first-order logic, higher-order logic, and their variants with arithmetic and shallow polymorphism.

Secondly, it is a library of test problems (i.e., theories in which some statements are marked as axioms and others as conjectures) for automated theorem provers for one of the TPTP logics (mostly first-order logic).

Thirdly, it is a suite of tools for syntax-checking, presenting, and transforming TPTP problems into other languages (mostly the input syntaxes of various theorem provers).

OpenTheory OpenTheory [Hur09]

MMT While TPTP fixes a small set of logics, whose semantics is assumed to be given externally, MMT is logic-independent and allows (in fact: requires) representing the syntax and semantics of the logic itself. All TPTP logics have been defined inside MMT/LF and an exporter from TPTP to MMT/LF is available. In fact, this constitutes the official (and executable) definition of semantically valid TPTP theories.

MMT does not focus on theorem proving but can be used in the same way as TPTP.

2 Logical Frameworks

2.1 Declarative and Proof-Theoretical

LF LF [HHP93] is a logical framework based on a version of typed lambda calculus that extends simple type theory with dependent function types. Twelf is a well-known concrete implementation of LF. LF is a logical framework that can be used to define concrete logics, based on the judgments-as-types methodology. As such it can serve as the meta-theory of many methematical frameworks ? which, however, will tend to be forms of constructive mathematics, thus excluding many traditionally accepted proofs and proof techniques. MMT can be used to represent LF and use it as the meta-theory for imported LF-based logics. MMT has a concrete-syntax plugin for Twelf,

Dedukti Dedukti [BCH12]

Isabelle Isabelle [Pau94] is a logical framework as well as an LCF-style interactive theorem prover. Most of what has been said about other logical frameworks applies here as well. MMT has no emphasis on theorem proving, but can be used to represent Isabelle as a logical framework and logics defined using Isabelle, such as that of Isabelle/HOL.

2.2 Abstract and Model-Theoretical

Institutions Institutions [GB92] is another logical framework, based on categorical model theory. It is difficult to compare LF and institutions because the approaches are almost orthogonal, but like LF, the framework of institutions can be used to define concrete logical systems. Much of the specifics of concrete logics are in fact instantiations of generic, categorically definable concepts in the institutional framework. LFI is a logical framework that can be viewed as a synthesis of the LF approach and the institutions approach, combining the advantages of each in one unified framework. It allows the specifier to specify the foundations needed to define the semantics ? proof theory and model theory ? of concrete mathematical formalisms in a fully explicit and formal way, while many specific aspects are as it were inherited from the higher-level theory. As before, MMT is so the speak one level higher up. The observation from LFI that many concepts can be defined at a higher level and then be inherited was extended one step further in MMT, thereby also offering a major simplification in the sense of representational uniformity. Theories at all levels ? logical frameworks, logical theories, abstract mathematical theories and concrete mathematical theories are all represented uniformly in MMT, using the same representational formalism.

Hets Hets [MML07]

3 Selected Individual Proof Assistants

While interchange formats and logical frameworks do not focus on a particular logic, the deepest tool support (in particular for type-checking and automated and interactive theorem proving) has been built for specific logics. These usually come with an idiosyncratic text input language, a tactic language, a module system, a dedicated (usually monolithic) tool for parsing, type-checking, and proving, and a library of formalized theories.

Interoperability between these tools is usually non-existent or highly brittle. Usually, each pair of tools differs in non-trivial ways in all of the following respects: the syntax (i.e., well-formedness) and semantics (i.e., provability) of the logic underlying the tool, the concrete input syntax used for it, the tactic language, the module system, and the definitions used in the library.

The following list is not complete and tries to focus on languages that provide the expressivity and proof support to be practical for the verification of mathematical theories or software.

Each of these languages L can be represented in Mmt, either directly, i.e., without a meta-theory, or with a logical framework as its meta-theory. In either case, the respective library can be represented as an Mmt library with meta-theory L.

3.1 Higher-Order Logic

HOL HOL [Gor88]

HOL Light HOL Light [Har96]

Isabelle/HOL Isabelle/HOL [NPW02]

3.2 Set Theory

Mizar Mizar [TB85] provides, next to a system, also a rich language for representing mathematical theorems and proofs. While there is an emphasis on proof checking, so that only verified theorems are included in the extensive Mizar library, the language can also be used for just representing mathematical theories. Mizar is based on specific logical and set-theoretic foundations, rather similar to ZFC, while MMT has no predefined foundations. The specific foundations of Mizar can be represented as an MMT theory, though, and be used as the meta-theory for imported Mizar theories. MMT has a concrete-syntax plugin for Mizar that can be used to parse and import the Mizar library.

Isabelle/ZF Isabelle/ZF [PC93]

3.3 Dependent Type Theory

Agda Agda [Nor05]

Coq Coq [Coq14]

Matita Matita [ACTZ06]

Nuprl Nuprl [CAB+86]

3.4 Other Foundations

ACL2 ACL2 [KMM00]

PVS PVS [ORS92]

Specware Specware [**?**] provides, next to a system, a language for representing mathematical theories, called ?specs? in Specware parlance. Although there is an emphasis on defining computable functions for which executable code can be generated, the language also allows one to specify functions and other mathematical objects by their properties, independent of notions of executability. Specware is also a module system in which one can specify morphisms between specs, both simple import morphisms and more complicated morphisms involving translations and interpretations. There is no notion of one spec being the meta-theory of another spec: all live at the same level, and the foundations, although not formally specified, are fixed. MMT is like Specware in that it is also a module system allowing one to define morphisms between theories. Unlike Specware, it accommodates several levels.

References

[ABC+03] R. Ausbrooks, S. Buswell, D. Carlisle, S. Dalmas, S. Devitt, A. Diaz, M. Froumentin, R. Hunter, P. Ion, M. Kohlhase, R. Miner, N. Poppelier, B. Smith, N. Soiffer, R. Sutor, and S. Watt. Mathematical Markup Language (MathML) Version 2.0 (second edition), 2003. See http://www.w3.org/TR/MathML2.

[ACTZ06] A. Asperti, C. Sacerdoti Coen, E. Tassi, and S. Zacchiroli. Crafting a Proof Assistant. In T. Altenkirch and C. McBride, editors, *TYPES*, pages 18–32. Springer, 2006.

[BCC$^+$04] S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaetano, and M. Kohlhase. The Open Math Standard, Version 2.0. Technical report, The Open Math Society, 2004. See `http://www.openmath.org/standard/om20`.

[BCH12] M. Boespflug, Q. Carbonneaux, and O. Hermant. The $\lambda\Pi$-calculus modulo as a universal proof language. In D. Pichardie and T. Weber, editors, *Proceedings of PxTP2012: Proof Exchange for Theorem Proving*, pages 28–43, 2012.

[CAB$^+$86] R. Constable, S. Allen, H. Bromley, W. Cleaveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. Sasaki, and S. Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, 1986.

[Coq14] Coq Development Team. The Coq Proof Assistant: Reference Manual. Technical report, INRIA, 2014.

[GB92] J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[Gor88] M. Gordon. HOL: A Proof Generating System for Higher-Order Logic. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer-Academic Publishers, 1988.

[Har96] J. Harrison. HOL Light: A Tutorial Introduction. In *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 265–269. Springer, 1996.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[Hur09] J. Hurd. OpenTheory: Package Management for Higher Order Logic Theories. In G. Dos Reis and L. Théry, editors, *Programming Languages for Mechanized Mathematics Systems*, pages 31–37. ACM, 2009.

[JSO97] JavaScript Object Notation, 1997. See `http://www.standardml.org/Basis/`.

[KMM00] M. Kaufmann, P. Manolios, and J Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.

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

[MML07] T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *Tools and Algorithms for the Construction and Analysis of Systems 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522, 2007.

[Nor05] U. Norell. The Agda WiKi, 2005. `http://wiki.portal.chalmers.se/agda`.

[NPW02] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[ORS92] S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Springer, 1992.

[Pau94] L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[PC93] L. Paulson and M. Coen. Zermelo-Fraenkel Set Theory, 1993. Isabelle distribution, ZF/ZF.thy.

[SS98] G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[TB85] A. Trybulec and H. Blair. Computer Assisted Reasoning with MIZAR. In A. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 26–28. Morgan Kaufmann, 1985.

[W3C98] W3C. Extensible Markup Language (XML), 1998. `http://www.w3.org/XML`.