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

Florian Rabe

November 26, 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. Other than 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-specific tools offer exports in custom XML languages for exporting their data, e.g., Mizar.

JSON JSON [Bra14] 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 objects may be references to globally visible symbols, which are declared 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 are not usable for representing the semantics of mathematical theories or type systems.

MathML MathML [ABC+03] consists of two XML languages. Firstly, *content* MathML is essentially isomorphic to OpenMath (different XML tags but same meaning). Secondly, *presentation* MathML marks up the shape (the presentation) 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/content MathML for representing formulas and presentation MathML for representing formulas whose content structure is not or only partially known. (OMDoc calls the latter semi-formal objects.)

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. More difficultly, 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 expected users of the standard. MMT additionally provides a reference implementation with a well-integrated 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/presentation MathML.

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 the fragment of the TPTP syntax that corresponds to the well-formed formulas has been officially specified only for some individual logics. 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. As such, it has become a standardization layer that makes logic tools interoperable. However, this interoperability is mostly restricted to first-order logic tools; for more complex type systems and logics, the ad hoc specification of the well-formed fragments has proved very challenging.

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

OpenTheory OpenTheory [Hur09] was specifically developed for the HOL family of systems. It aims at exporting libraries in order to reimport them in other provers. Consequently, it provides support for abstracting from definitions (which may be incompatible between libraries) and preserving high-level proof steps (which are more robust to import).

MMT While TPTP and OpenTheory fix 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 well-formed TPTP theories. All variants of HOL have been or can easily be represented in Mmt/LF.

Mmt does not focus on theorem proving and exchange between specific theorem provers. Therefore, TPTP and OpenTheory provide better tool support for their specific application and their specific family of logics.

2 Logical Frameworks

Logical frameworks are formalisms in which the syntax and semantics of logics (and similar languages) can be defined. [Rab14] gives an overview. Two groups of frameworks can be distinguished.

A *declarative* framework F is themselves a logic and allows representing other logics L as theories of F. Results about F such as normalization or type-checking induce according results for each L. Typically but not necessarily, declarative frameworks focus on proof theoretical and constructive aspects, thus excluding many traditionally accepted proofs and proof techniques. [Pfe01] gives an overview.

An *abstract* logical framework defines logics using abstract sets of sentences and models or (less often) proofs. Abstract frameworks are often formulated in terms of category theory. They do not focus on concrete syntax of logics and are not limited to finite or computable logic definitions.

2.1 Declarative and Proof-Theoretical

LF LF [HHP93] is a logical framework based on the dependently-typed λ-calculus. It uses the judgments-as-types methodology. In particular, logic definitions usually use a declaration $\texttt{proof} : \texttt{form} \to \texttt{type}$ such that $\texttt{proof}\, F$ is the type of proofs of F.

Twelf [PS99] is the most mature concrete implementation of LF. It includes a theorem prover for meta-theorems, i.e., theorems *about* the defined logics.

A wide variety of logics have been defined in Twelf, e.g., in the LATIN library [CHK+11]. It includes an exporter to Mmt's XML language.

Dedukti Dedukti [BCH12] implements LF modulo rewriting. By supplying rewrite rules (whose confluence Dedukti assumes) in addition to an LF theory, users can give more elegant logic encodings.

A number of logic libraries have been exported to Dedukti, which is envisioned as a universal proof checker. An export from Dedukti to Mmt is planned.

Isabelle Isabelle [Pau94] is a logical framework based on intuitionistic higher-order logic. Seen as a pure type system [ABI+96] with propositions-as-types, its underlying Pure logic is very similar to LF. The key declaration in a logic definition is usually of the form $\texttt{true} : \texttt{form} \to \texttt{prop}$.

Isabelle includes an LCF-style interactive theorem prover and a tactic language for proving object theorems, i.e., theorems *within* the defined logic. Despite being logic-independent, most of the proof support in Isabelle is optimized for individual logics defined in Isabelle, most importantly Isabelle/HOL, with which Isabelle is often mistakenly identified. Other logic definitions with sizable libraries are Isabelle/ZF and Isabelle/FOL.

MMT Mmt abstracts from and subsumes declarative logical frameworks. Thus, the logical framework can be chosen and evolved flexibly. Moreover, almost all of the tool support in Mmt is defined independent of the logical framework. Mmt currently lacks sophisticated theorem provers as in Twelf or Isabelle.

A declarative framework F is represented in Mmt as a theory F. Then logics L defined in F are represented as theories L with meta-theory F, and L-theories T are represented as theories T with meta-theory L.

The semantics of an Mmt theory is defined by its meta-theory. Thus, if the semantics of F is defined, it induces the semantics of L and T. For theories without meta-theory such as F, the semantics can be defined by a set of inference rules implemented in a plugin.

Mmt includes a set of plugins that define the inference rules for several frameworks related to LF, including in particular LF itself, LF with rewriting, and LF with shallow polymorphism. The semantics of other frameworks can be defined accordingly.

2.2 Abstract and Model-Theoretical

Institutions Institutions [GB92] is the most developed abstract logical framework. Logics are defined as a tuple of a category of theories, two functors mapping theories to sentences and model classes respectively, and a satisfaction relation between sentences and models. Most logics can be formulated as institutions, and many meta-theorems can be generalized to the institution-independent level [Dia08].

Institutions live inside category theory and provide no tool support in themselves. From a practical perspective, their primary strength is that they introduce intuitions and methodologies that, if followed, yield elegant implementations.

Hets Hets [MML07] is an implementation of institutions. It defines the data type of institutions as a class in a programming language. On top of that, it adds support for parsers and checkers, logic translations (institution comorphisms), modular specifications, and the shipping of proof obligations to external provers.

It comes with a number of built-in individual institutions and translations, centered around the CASL family of logics [CoF04]. Contrary to declarative frameworks, adding a new institution requires modifying the Hets source code.

MMT [Rab14, Rab13] can be viewed as a synthesis of the declarative and the institutions approach, combining the advantages of each in one unified framework. The basic idea is to use declarative logical frameworks as the environment in which logics, theories, formulas, proofs, and models are defined concretely (and for which tool support is available) and institutions as the platonic background in which these objects exist (and in which meta-logical results are established).

In particular, [Rab13] shows how logic definitions in LF induce institutions. This was used in [CHK+12] to write logic definitions in Twelf, export them to Mmt, and then import the logic definitions into Hets.

Mmt does not provide any explicit support for institutions, but the available primitives are designed to represent those institutions induced by declarative frameworks. This yields a major simplification in the sense of representational uniformity: Theories at all levels – logical frameworks, logics, and abstract and concrete mathematical theories, and mathematical foundations – are represented uniformly as Mmt theories.

Consider a theory T of a logic L defined in a declarative framework F. Then formulas, proofs, and other expressions over T are represented as Mmt objects over T. Consider, moreover, a foundation of mathematics (e.g., any of the languages from Sect. 3) defined as an Mmt theory: this could be for example a theory ZFC with meta-theory LF as in [IR10]. Then models of T are represented as Mmt theory morphisms $T \to ZFC$.

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 mathematics 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.

There is relatively little survey literature that compares these systems across the board. [Wie06] and the related work in [RK13] survey individual aspects.

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]

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 [SJ95]

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.

[ABI+96] P. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A Theorem-Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.

[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 λΠ-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.

[Bra14] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, Internet Engineering Task Force, 2014.

[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.

[CHK+11] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, and F. Rabe. Project Abstract: Logic Atlas and Integrator (LATIN). In J. Davenport, W. Farmer, F. Rabe, and J. Urban, editors, *Intelligent Computer Mathematics*, pages 289–291. Springer, 2011.

[CHK+12] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, F. Rabe, and K. Sojakova. Towards Logical Frameworks in the Heterogeneous Tool Set Hets. In T. Mossakowski and H. Kreowski, editors, *Recent Trends in Algebraic Development Techniques 2010*, pages 139–159. Springer, 2012.

[CoF04] CoFI (The Common Framework Initiative). *CASL Reference Manual*, volume 2960 of *LNCS*. Springer, 2004.

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

[Dia08] R. Diaconescu. *Institution-independent Model Theory*. Birkhäuser, 2008.

[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.

[IR10] M. Iancu and F. Rabe. Formalizing Foundations of Mathematics, LF Encodings, 2010. see https://latin.omdoc.org/wiki/FormalizingFoundations.

[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.

[Pfe01] F. Pfenning. Logical frameworks. In J. Robinson and A. Voronkov, editors, *Handbook of automated reasoning*, pages 1063–1147. Elsevier, 2001.

[PS99] F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. *Lecture Notes in Computer Science*, 1632:202–206, 1999.

[Rab13] F. Rabe. A Logical Framework Combining Model and Proof Theory. *Mathematical Structures in Computer Science*, 23(5):945–1001, 2013.

[Rab14] F. Rabe. How to Identify, Translate, and Combine Logics? *Journal of Logic and Computation*, 2014. to appear.

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

[SJ95] Y. Srinivas and R. Jüllig. Specware: Formal Support for Composing Software. In B. Möller, editor, *Mathematics of Program Construction*. Springer, 1995.

[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.

[Wie06] F. Wiedijk. *The Seventeen Provers of the World*. Springer, 2006.