MMT:
A Foundation-Independent Approach to Formal Knowledge

Florian Rabe

July 15, 2016*

Abstract

MMT is a framework for representing declarative languages such as logics, type theories, set theories, etc.. It achieves a high level of generality by systematically avoiding a commitment to a particular syntax or semantics. Instead, individual language features (e.g., λ-abstraction, conjunction, etc.) and syntax features (keywords, notations, etc.) are defined as separate, reusable modules, from which individual languages are assembled. These modules can be declarative by specifying features as MMT theories or programmatic by providing individual rules as plugins.

Despite this high degree of abstraction, it is possible to implement advanced algorithms generically at the MMT level. These include knowledge management algorithms (e.g, IDE, search, change management) as well as logical algorithms (e.g., parsing, type reconstruction, module system). Thus, we can use MMT to obtain strong implementations of declarative languages at extremely low cost.

Moreover, the focus on modularity and language-independence enables system integration, where MMT can mediate the exchange of knowledge across different foundational systems and concrete syntaxes.

1 Motivation

Methods based on formal knowledge pay off almost exclusively at large scales due to the high level of theoretical understanding and practical investment that they require from both developers and users. But for the same reason, it is difficult for individual approaches to reach those large scales. Today's successful projects such as the Mizar Mathematical Library [TB85] or the formal proof of the Kepler conjecture [HAB+14] are built on double-digit person years of investment.

Therefore, the last few decades have seen increasing specialization into **isolated, mutually incompatible systems** and **incompatible overlapping libraries of formal knowledge**. Moreover, during that time the advances in computer and internet technology have dramatically changed our expectations regarding scalability. Many requirements have become critical that are not anticipated in the designs of formal systems such as collaboration, system interoperability, and modularity. **Three central design choices have proved problematic**:

Fixed Foundations Virtually all current systems are based on a fixed foundation, i.e., a fixed logic in which all formalizations in that system are stated. While the vast majority of classical

*The latest version of this document is available at http://uniformal.github.io/doc/philosophy/articles/mmt.pdf.

mathematics has been formulated in axiomatic set theory, alternative foundations such as higher-order logic or constructive type theory have become important in computer-supported approaches. Today almost all systems use foundations that are different from each other and different from classical set theory, and all attempts to find the "mother of all logical systems" (and convince others to use it) have failed, e.g., the qed project [Ano94]. This is all the more frustrating because these incompatibilities are often irrelevant for high-level formalization goals.

Homogeneity Classical mathematics uses the heterogeneous method, going back to the works by Bourbaki [Bou64], which focuses on defining *theories* and stating every result in the smallest possible theory. This allows using *theory morphisms* to move results between theories in a truth-preserving way [FGT92]. Consequently, mathematics is usually carried out in highly abstracted settings where the foundational details are hidden. Yet, virtually all formalizations in current systems are based on the homogeneous method, which uses only conservative extensions (e.g., definitions, theorems) of the fixed foundation to model mathematical knowledge. Therefore, formalizations inherently depend on the fixed foundation.

Local Scale Mathematical research and applications are distributed globally, and mathematical knowledge is highly interlinked by explicit and implicit references. Therefore, a computer-supported management system should support global interlinking, and management algorithms have to scale up to large (global) data sets. Yet, virtually all current systems operate under the implicit assumption that all knowledge is locally available and loaded into main memory. In fact, because these systems have initially focused on soundness and efficiency at small scales, large scale knowledge management has proved very difficult to add as an afterthought, often prohibitively so.

However, **developers' resources are stretched thin already** by developing (and maintaining) their system at all. Therefore, the gradual migration towards new designs that overcome these problems is extremely difficult.

The MMT language and system have been designed from scratch to provide a uniform solution to the above three problems: MMT **is a globally scalable Module system for Mathematical Theories that abstracts from and mediates between different foundations and maximizes the reuse of concepts, tools, and formalizations**. Thus, it provides a theoretical and practical foundation for formal knowledge in the large.

2 The MMT Language

MMT is systematically **foundation-independent**. It separates large scale concerns, which are addressed generically by MMT, and small scale concerns, which are addressed by individual foundational languages. Thus, foundation-specific development can focus on the logical core of the foundation instead of spending resources on ad hoc large scale support. Dually, large scale support can be developed generically (and often more easily) at the MMT level.

MMT **integrates successful representational paradigms**

- the logics-as-theories representation from proof theoretical frameworks like LF [HHP93],
- categories of theories from model theoretical frameworks like institutions [GB92],
- the structured theories from algebraic specification languages like [SW83],
- reuse along theory morphisms from the "little theories" approach [FGT92],
- the Curry-Howard correspondence from type/proof theory [CF58, How80],
- URIs as logical namespace identifiers from OPENMATH [BCC+04],
- standardized XML-based interchange syntax from markup languages like OMDOC [Koh06]

and makes them available in a single, coherent representational system for the first time. The combination of these features is **based on a small set of carefully chosen, orthogonal primitives** in order to obtain a simple and extensible language design.

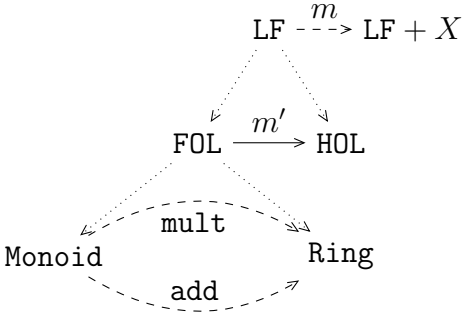
The central concept of MMT is that of a **theory**, which is a named list of declarations. Most importantly, the declaration of a **constant** introduces a new name possibly with additional attributes such as type, definiens, or notation. Relative to a theory, **objects** are formed as syntax trees with binding.

These three concepts are sufficient to naturally represent virtually all formal languages.

- All languages are represented uniformly as MMT theories. This includes foundations, logical frameworks, logics and type theories, signatures and theories, set theories.
- All operators and symbols of a language are represented as MMT constants. This includes the sort, constant, function, and predicate symbols of logics, the base type, type operators, and term constructors of type theories, the concepts, relations, and individuals of ontologies, as well as – via the Curry-Howard correspondence – the judgments, inference rules, axioms, and theorems of calculi.
- All composed expressions are represented as objects. This includes formulas, derivations and proofs, terms, types, kinds, and universes, etc.

All MMT theories are related via **theory morphisms**, which are used to uniformly describe representation theorems between theories. These include translations, functorial representations, implementations, and models. They are also used as the semantics of **import** declarations within theories, which uniformly represent all aspects of building theories modularly. These include inheritance and instantiation.

A key innovation of MMT is the **meta-theory** relation between theories. Let us write M/T to express that we work in the object language T using the meta-language M. For example, most of mathematics is carried out in `FOL/ZFC`, i.e., first-order logic is the meta-language, in which set theory is defined. `FOL` itself might be defined in a logical framework such as LF [HHP93], and within ZFC, we can define the language of natural numbers, which yields `LF/FOL/ZFC/Nat`.



In MMT, all of these languages are represented as theories, each of which may be reused as the meta-theory of another one. Crucially, the meta-theory indicates both to humans and to machines how a theory is to be understood. For example, interpretations of `ZFC` must understand `FOL`, and the typing relation of `FOL` is inherited from `LF`. The diagram of MMT theories on the right gives an example. Here dotted arrows denote the meta-theory relations, solid arrows are language translations, and dashed arrows are imports. $\texttt{LF} + X$ represents any extension of LF with additional features.

We can see MMT as the **last step in a progression towards more abstract formalisms** as indicated below. In conventional mathematics, domain knowledge is expressed directly in ad hoc notation. Logic provided a formal syntax and semantics for this notation. Logical frameworks provided a formal way to define this syntax and semantics. Now MMT adds a meta-level, at which we can design logical frameworks. That makes MMT very robust against future language developments: We can develop $\texttt{LF} + X$ without any change to the MMT infrastructure and can easily migrate all results obtained within LF.

Mathematics	Logic	Logical Frameworks	Foundation-Independence
			MMT
		logical framework	logical framework
	logic	logic	logic
domain knowledge	domain knowledge	domain knowledge	domain knowledge

[RK13] provides a comprehensive (albeit by now slightly outdated) introduction to the MMT language. A more recent treatment that focuses on the representation of logics in MMT is given in [Rab14].

3 The MMT System

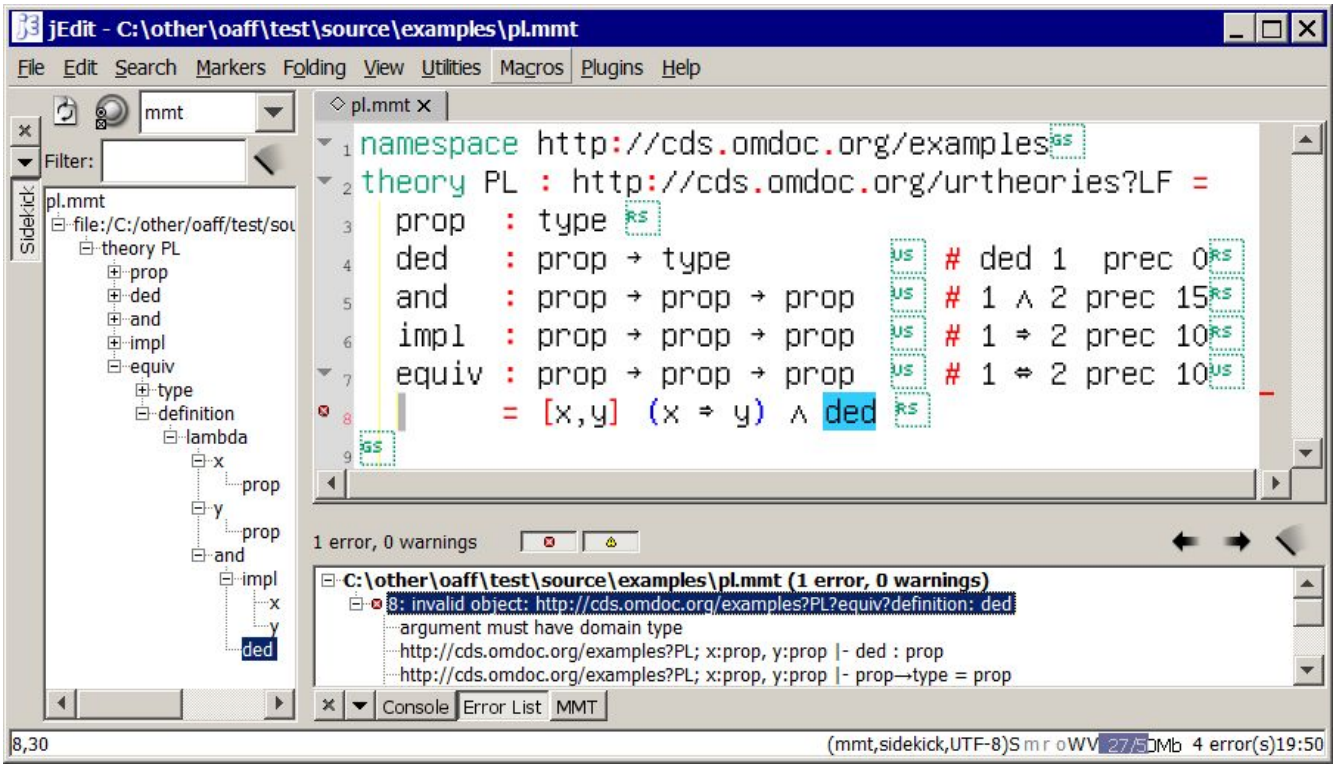


Figure 1: MMT IDE based on jEdit

Exploiting the small number of primitives in the MMT language, the MMT API provides a simple scalable implementation of MMT and MMT-based functionality (written in the functional and object-oriented language Scala [OSV07]).

Due to the generality of MMT, one might think that it is not possible to implement deep, meaningful functionality at the MMT level. Indeed, implementing any result requires first generalizing it to the MMT level. However, **practical experience has shown that most results can be generalized to the MMT level**. For each result, this may require a substantial research effort, which samples existing results for specific foundations and recovers them as special cases of a general principle. But it is doubly rewarding: Besides yielding a general result, the abstract level of MMT provides a more focused view on a concept and often yields clearer intuitions.

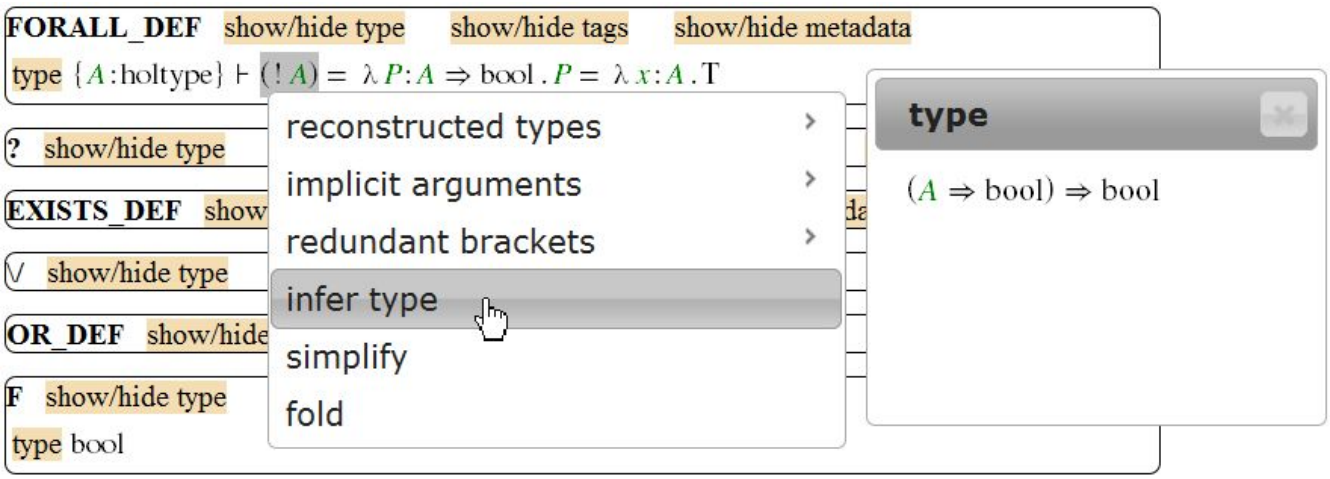


Figure 2: Type Inference in the Mmt Web Server

All Mmt-level functionality is implemented foundation-independently, and foundation-specific aspects (if any) are left abstract and supplied by plugins. Crucially, experience shows that the vast **majority of the implementation is foundation-independent**. For example, the Mmt API comprises $> 30,000$ lines of code, and the LF plugin, which provides in particular typing and proof rules for LF, comprises < 2000 lines.

Logical results implemented at the Mmt level include notations and parsing, module system and theory transformations, type reconstruction and simplification, as well as a (so far very basic) theorem prover. For example, the foundation-independent aspects of type reconstruction include lookup of identifiers, implicit arguments, solving for meta-variables, constraint delay, and error reporting. The LF plugin only supplies ~ 10 rules of a few lines each, which correspond directly to the statement of the rules on paper – advanced aspects such as modularity and unsolved meta-variables remain transparent to the plugin developer.

Knowledge management results implemented at the Mmt level include project management, build system, IDE (build by integrating Mmt with the jEdit text editor), interactive browsing using HTML+presentation MathML [ABC+03] and JavaScript, change management, as well as indexing, querying, and search (the latter builds on top of MathWebSearch [KŞ06]). Plugin interfaces allow the convenient import/export of content in other formats.

Mmt URIs serve as identifiers throughout the implementation and abstract from physical storage units such as file systems or versioned repositories. Mmt maintains a catalog that maps URIs to physical locations. Mmt content is loaded into memory only when needed, and the distribution of content over physical storage and networks remains transparent to Mmt-based services.

For example, Fig. 1 shows screenshot of a definition of propositional logic with meta-theory LF in the Mmt IDE. An intentionally introduced error was detected by type reconstruction and highlighted. Note how the sidebar shows the abstract syntax tree of the theory: The types of the variables x and y were inferred and are displayed in the syntax tree even though type reconstruction for the whole object failed. Other features include hyperlinks, context-sensitive auto-completion, or interactively solving for subexpressions based on the expected type.

Fig. 2 shows a screenshot of a part of the Mmt web server. It shows a fragment of the HOL Light library as imported into Mmt in [KR14a]. It shows how the web browser infers and displays

the type of the selected subexpression (in the definition of the universal quantifier, which is written ! in HOL Light). Other interactive features include folding subexpressions, hiding/showing inferred types, implicit arguments, and redundant brackets, or retrieving the definition of a symbol. It is also

All this **functionality can be made available to individual formal systems at small cost** – all foundation-specific code for the above two examples is contained in the LF plugin.

Many features of the Mmt system are described in individual publications. An overview is given in [Rab13b]. Source code, binaries, and documentation of the Mmt system are available at http://uniformal.github.io.

4 Integrating Languages: The LATIN Atlas

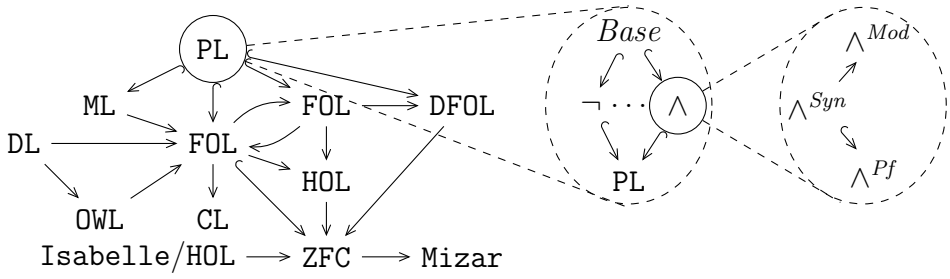


Figure 3: A Fragment of the LATIN Atlas

The LATIN project [CHK+11, KMR09] (2009-2012) built a **library of formalizations of logics and related languages** as well as translations between them. It used LF as the meta-theory for all languages.

Fig. 3 shows a high-level view of a fragment of the resulting diagram of Mmt theories. The left side shows some of the logics, the middle zooms into the modular definition of propositional logic, and the right side zooms in further to show the definition of conjunction as a triple of syntax, proof theory, and model theory.

The formalized **logics** include propositional, first-order, sorted first-order, common, higher-order, modal, description, and linear logics. Type theoretical features, which can be freely combined with logical features, include the λ-cube, product and union types, as well as base types like booleans or natural number. In many cases alternative formalizations are given (and related to each other), e.g., Curry- and Church-style typing, or Andrews and Prawitz-style higher-order logic. The logic **morphisms** include the relativization translations from modal, description, and sorted first-order logic to unsorted first-order logic, the negative translation from classical to intuitionistic logic, and the translation from first to sorted first- and higher-order logic. The model theoretical semantics is represented as theory morphisms from a logic into a theory representing a foundation. The **foundations** include Zermelo-Fraenkel set theory, Church's higher-order logic, and Mizar's formalized set theory [TB85].

All representations systematically exploit modularity and form a single **highly interconnected diagram of Mmt theories**. Every logical principle, e.g., as conjunction, the universal quantifier of first-order logic, or the extensionality principle of higher-order logic, is formalized in a separate module. Thus, **logics can be composed modularly** from the individual features using

the Mmt module system. For example, logic of Isabelle [Pau94] can be obtained by combining the modules for Church-style typing, simple function types, a boolean type, implication, typed universal quantification, and typed equality, as well as corresponding theories for the proof theory and corresponding theory morphisms for the model theory.

The full LATIN graph comprises several hundred modules and is available at `img/latin-graph.html`[1] and spreads about 200 files. Particularly interesting representations are described in detail in [IR11, HR11, BRS08]. The theoretical background of representing the proof and model theory of logics in Mmt is discussed in [Rab13a, Rab14].

5 Integrating Libraries: The Open Archive of Formalizations

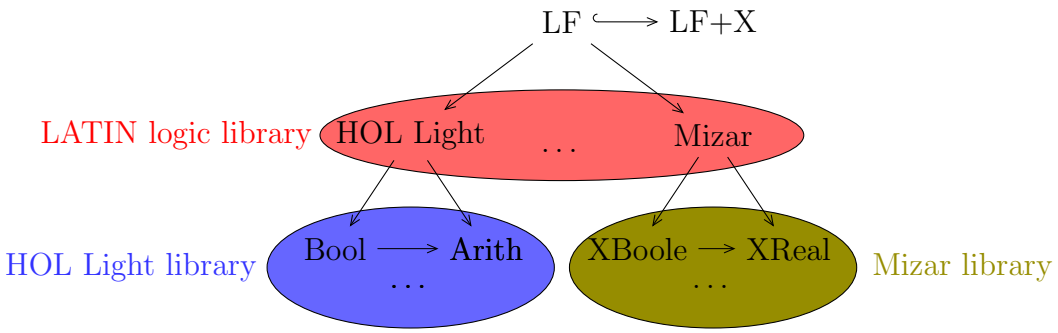


Figure 4: Representing Libraries in Mmt

The OAF project [KR14b] (2014-2017) aims at **overcoming the isolation and mutual incompatibility between the various large libraries** of formal knowledge obtained by applying the homogeneous method to various fixed foundations such as Mizar [TB85], the HOL systems [Gor88, NPW02, Har96], PVS [ORS92], Coq and Matita [Coq15, ACTZ06], as well as the heterogeneous libraries of TPTP [Sut09] and IMPS [FGT93]. The project proceeds in 2 phases, both of tremendous difficulty.

In the first phase, we **make the existing libraries available in a standardized format** by importing them into Mmt. This requires two steps for each library:

1. We represent the logic of the used system in Mmt relative to a logical framework. This representation doubles as a documentation layer for the logic and often requires a deeper understanding of the logic and its implementation than published in the literature. In some cases LF will be sufficient as a logical framework, but in other cases stronger logical frameworks may have to by designed within Mmt.
2. Much more difficultly, we implement exports from the used system into Mmt. The specification of the export is relatively simple except when languages use very unusual idiosyncrasies. But the difficulty of the implementation is huge and can even be impossible without refactoring the system. Therefore, substantial collaboration from within the respective developer community is indispensable.

[1] absolute path: `http://uniformal.github.io/doc/philosophy/articles/img/latin-graph.html`

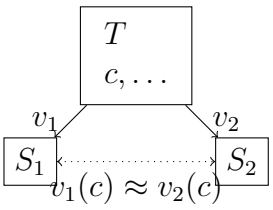
At this point, these steps have been mostly completed for Mizar and HOL Light, resulting in the MMT diagram of Fig. 4.

In the second phase, we will be able to attack the **library integration problem**. This problem cripples state-of-the-art integration attempts between homogeneous systems: Theorems about, for example, the real numbers defined in one foundation cannot be used to reason about the real numbers as defined in another foundation. The heterogeneous method could avoid this problem because all formalizations could be moved easily along morphisms. Therefore, an important goal is to refactor the existing libraries to be more heterogeneous and to investigate partial or approximate solutions.

A central method will be to use **interface logics and interface theories** akin to the use of interfaces in software engineering. Every problem should be stated in the weakest possible interface logic and theory, and then individual foundations that realize the interface can be used to obtain the proof. This is crucial because foundations are usually substantially more expressive than the interface logic needed to state a problem. For example, axiomatic set theory, higher-order logic, and constructive type theory are common foundations, but we can give an interface theory for the real numbers in a much weaker logic, e.g., a fragment of second-order logic. While giving a logic translation between any two typical foundations can be prohibitively difficult, a weak interface logic can be easily imported into all of them. This is no coincidence: Whereas foundations are designed to be simple and logically very expressive (because they should be fixed and implemented once and for all), interface logics should be as inexpressive as possible even if that makes them more complex.

The LATIN logic library already developed most of the needed interface logics. This will have to complemented with a library of interface theories for the typical domains of computer science and mathematics.

Interfaces also provide a way to integrate libraries by using **alignments** [RKS11]. In the diagram on the right, an interface theory T is realized by two systems S_1 and S_2, which is witnessed by two theory morphisms $v_i : T \to S_i$. For a T-symbol c, we say that $v_1(c)$ and $v_2(c)$ are *aligned* via (c, v_1, v_2). Alignments provide a semantically backed concept for cross-references between libraries and provide a starting point for library translations that overcome the library integration problem.



Acknowledgments

The MMT language was developed in collaboration with Michael Kohlhase. The MMT system includes contributions from various people at Jacobs University, most importantly Mihnea Iancu. The results of the LATIN project were obtained in collaboration with Michael Kohlhase, Till Mossakowski (who were principal investigators), Fulya Horozal, and Mihai Codescu; the project was supported by DFG grants KO 2428/9-1 and MO 971/2-1. The OAF project is conducted in collaboration with Michael Kohlhase; it is supported by DFG grants KO 2428/13-1 and RA-1872/3-1; the mentioned library imports were implemented in collaboration with Josef Urban and Mihnea Iancu for Mizar and with Cezary Kaliszyk for HOL Light.

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.

[Ano94] Anonymous. The QED Manifesto. In A. Bundy, editor, *Automated Deduction*, pages 238–251. Springer, 1994.

[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.

[Bou64] N. Bourbaki. Univers. In *Séminaire de Géométrie Algébrique du Bois Marie - Théorie des topos et cohomologie étale des schémas*, pages 185–217. Springer, 1964.

[BRS08] C. Benzmüller, F. Rabe, and G. Sutcliffe. THF0 – The core of the TPTP Language for Higher-Order Logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *4th International Joint Conference on Automated Reasoning*, pages 491–506. Springer, 2008.

[CF58] H. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.

[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.

[Coq15] Coq Development Team. The Coq Proof Assistant: Reference Manual. Technical report, INRIA, 2015.

[FGT92] W. Farmer, J. Guttman, and F. Thayer. Little Theories. In D. Kapur, editor, *Conference on Automated Deduction*, pages 467–581, 1992.

[FGT93] W. Farmer, J. Guttman, and F. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11(2):213–248, 1993.

[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.

[HAB+14] T. Hales, M. Adams, G. Bauer, D. Tat Dang, J. Harrison, T. Le Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. Tat Nguyen, T. Quang Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. Thi Ta, T. Nam Tran, D. Thi Trieu, J. Urban,

K. Khac Vu, and R. Zumkeller. A formal proof of the Kepler conjecture, 2014. http://arxiv.org/abs/1501.02155.

[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.

[How80] W. Howard. The formulas-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[HR11] F. Horozal and F. Rabe. Representing Model Theory in a Type-Theoretical Logical Framework. *Theoretical Computer Science*, 412(37):4919–4945, 2011.

[IR11] M. Iancu and F. Rabe. Formalizing Foundations of Mathematics. *Mathematical Structures in Computer Science*, 21(4):883–911, 2011.

[KMR09] M. Kohlhase, T. Mossakowski, and F. Rabe. The LATIN Project, 2009. see https://trac.omdoc.org/LATIN/.

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

[KR14a] C. Kaliszyk and F. Rabe. Towards Knowledge Management for HOL Light. In S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics*, pages 357–372. Springer, 2014.

[KR14b] M. Kohlhase and F. Rabe. The OAF Project, 2014. see https://svn.kwarc.info/repos/frabe/www/OAF/index.html.

[KŞ06] M. Kohlhase and I. Şucan. A Search Engine for Mathematical Formulae. In T. Ida, J. Calmet, and D. Wang, editors, *Artificial Intelligence and Symbolic Computation*, pages 241–253. Springer, 2006.

[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.

[OSV07] M. Odersky, L. Spoon, and B. Venners. *Programming in Scala*. artima, 2007.

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

[Rab13a] F. Rabe. A Logical Framework Combining Model and Proof Theory. *Mathematical Structures in Computer Science*, 23(5):945–1001, 2013.

[Rab13b] 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.

[Rab14] F. Rabe. How to Identify, Translate, and Combine Logics? *Journal of Logic and Computation*, 2014. doi:10.1093/logcom/exu079.

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

[RKS11] F. Rabe, M. Kohlhase, and C. Sacerdoti Coen. A Foundational View on Integration Problems. In J. Davenport, W. Farmer, F. Rabe, and J. Urban, editors, *Intelligent Computer Mathematics*, pages 107–122. Springer, 2011.

[Sut09] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[SW83] D. Sannella and M. Wirsing. A Kernel Language for Algebraic Specification and Implementation. In M. Karpinski, editor, *Fundamentals of Computation Theory*, pages 413–427. Springer, 1983.

[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.