QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge

MICHAEL KOHLHASE and FLORIAN RABE
Jacobs University Bremen
http://kwarc.info

Proposed in 1994, the "QED project" was one of the seminally influential initiatives in automated reasoning: It envisioned the formalization of "all of mathematics" and the assembly of these formalizations in a single coherent database. Even though it never led to the concrete system, communal resource, or even joint research envisioned in the QED manifesto, the idea lives on and shapes the research agendas of a significant part of the community

This paper surveys a decade of work on representation languages and knowledge management tools for mathematical knowledge conducted in the KWARC research group at Jacobs University Bremen. It assembles the various research strands into a coherent agenda for realizing the QED dream with modern insights and technologies.

1. INTRODUCTION

Even though short-lived and ultimately unsuccessful, the QED project and manifesto [Ano94] of 1994 were enormously influential on automated reasoning. The QED manifesto urged the automated reasoning community to work towards a universal, computer-based database of all mathematical knowledge, strictly formalized in logic and supported by proofs that can be checked mechanically. The QED database was intended as a communal resource that would guide research and allow the evaluation of automated reasoning tools and systems. This database was never realized – not even started as a concrete collection or system – but the discussion about it influenced a whole generation of researchers and practitioners in the automated reasoning and formal methods communities. Indeed the QED idea – if not the system and project – is very much alive in the community today. On the twentieth anniversary of the QED manifesto we survey the state of formalization and proving and the chances of realizing the twenty-year-old dream after all.

The work presented here has partially been funded by the DFG under grants KO 2428/13-1, KO 2428/9-1, and RA-1872/3-1. The intuitions and results presented here summarize the efforts of the KWARC group at Jacobs University Bremen and a number of external collaborators as listed in the various cited publications. In particular, we benefited from discussions with Jacques Carette, Mihai Codescu, William Farmer, Till Mossakowski, and Claudio Sacerdoti Coen. The MathHub.info system was built in collaboration with Mihnea Iancu and Constantin Jucovschi. Two anonymous referees wrote very insightful, detailed, and constructive reviews, which helped improve the article significantly.

We find that many of the problems that prevented the QED project from succeeding were already part of the initial discussion on the QED mailing list. The pertinent topics were *i*) the choice of the *root logic*, its calculi and proof formats, in particular regarding types and partial functions, *ii*) the question whether mathematics truly needed formalization and consequently the acquisition of funding. While the latter point was probably the main practical obstacle, the former was and still is arguably the biggest theoretical stumbling block for a *universal* library of formalized mathematics. There was a consensus that a root logic was needed, i.e., a logic in which all of mathematics would be formalized. But most groups only advocated the logic of their own system as the most feasible candidate. The discussion ended in a proposal by Bob Boyer – the driving force behind QED – to use Primitive Recursive Arithmetic [Sko67] (PRA), a quantifier-free formalization of the natural numbers. But while PRA is a canonical choice of root logic due to its extreme simplicity, it was generally regarded as insufficiently expressive for *practical* formalization. For example, in the recollection of the first author, Peter Andrews expressed this sentiment with the sarcastic comment that "if PRA, then we should gödelize it first!".

Actually, Andrews' position in the discussion was much more constructive than this quote suggests and has motivated much of the authors' research. In one post [And94] to the QED mailing list, he advocated to "accept diversity": a library organization that accepts theorems and proofs in *any* logic after that logic has itself been submitted to the database. The reasoning behind this is that the differing between root logics (then and now) have evolved for good reasons: Every such system caters to a different aspect of formalization or area of application. However, at the time of QED the development of logical frameworks [Pfe01], which can give a theoretical basis for this vision, had just begun.

Since then, the plurality of logic designs and of the corresponding theorem proving technologies has become even more formidable. Today there are about half-a-dozen libraries with $\sim 10^5$ formalized mathematical theorems, such as the ones of Mizar [TB85], Coq [Tea], and the HOL systems [Har96]. These include deep theorems such as the Odd-Order Theorem [GA+13] or the Kepler Conjecture [HA+15]. But each library is based on a separate root logic, all of which are mutually incompatible, which leads to duplication of work and missed opportunities for knowledge transfer.

In his post [And94], Andrews argues for another so far-unrealized goal: "I envision the QED library as a vast database and *a large number of associated utilities* for accessing, displaying, translating, manipulating, and using the items in the database" (emphasis ours). This places the utilities at the same level as the formalizations and not at the level of the individual systems contributing to the QED database. This makes sense if we consider the need for additional tools that integrate formalization systems with symbolic computation and large scale knowledge management systems – a very real need as evidenced by the formation of the CALCULEMUS and MKM (Mathematical Knowledge Management) communities, respectively, in the last two decades.

In 2007, Freek Wiedijk identified two further reasons for the failure of QED [Wie07]. Firstly, even though expressivity was an important factor in choosing the root logics, they still lack *practical* expressivity in comparison with conventional mathematics. He gives four deep but conventionally non-problematic mathematical theorems from calculus, abstract algebra, category theory, and set theory and concludes that no system allows proving all of them based on natural definitions and notations. This is compatible with the observation that each root logic has particular strengths and no single root logic can be satisfactory for all applications.

Secondly, the large libraries that have been developed lack library organization strategies to keep them coherent in the presence of many contributors. Wiedijk observes that today's large libraries are either well-curated and integrated or have a large number of contributors. Naturally, while it is possible to develop library management support such as refactoring and dependency tracking, it would significantly ease the implementation burden on each system if these features were provided generically by the QED database.

Contribution and Overview. We begin with a thorough review of the state of the art in Section 2. We describe the existing tools, their root logics, and their libraries, focusing on the integration problems caused by the plurality of root logics.

Our contribution is a novel design that we propose for a **universal community-driven QED database**. Contrary to all existing large QED-style databases, ours is systematically **pluralistic**: We build the diversity of root logics into the representation language from the beginning. This is a key step towards solving the problem of incompatible root logics.

A pluralistic QED database must include the definitions of the various root logics (and their semantics) and understand formalizations in any of them. Therefore, we use logical frameworks in which the root logics are to be formalized. Moreover, we complement logical frameworks with a concrete meta-language that allows representing both these logic definitions and the libraries. We use our OMDoc/Mmt language as this meta-language, which we present in Section 3.1. Mmt also acts as a pluralistic deduction system, which can be used to implement different logical frameworks, check the definitions of the root logics, and to support the QED database with knowledge management services. We present this in Section 3.2. We do not, however, anticipate that such a single pluralistic system will replace existing ones — instead, we see our database as a uniform archiving and integration platform for formalizations obtained in a variety of deductions systems, each optimized for a different root logic.

On top of Mmt, we have built the MathHub.info system as a prototypical QED database infrastructure, which we describe in Section 4. It provides user management, repository manager, a web interface for trans-library navigation, and a mathematical search engine. Going beyond the goals of QED, it also includes support for informal mathematical documents.

We have already started building a nucleus of a pluralistic QED database on top of MathHub.info, and we report on our current state in Section 5. Presently, it includes several major mathematical libraries including those of two QED-style proof assistants. Section 5.1 and 5.2 describe how we can represent the root logics and their respective libraries uniformly. Section 5.3 discusses the most promising methods for using our database to integrate these libraries with each other.

Throughout the article, we point out open problems that still remain despite these promising prototypes, and we conclude in Section 6.

2. STATE OF THE ART IN FORMALIZATION AND DEDUCTION

The systematic formalization of mathematical knowledge and its semantics go back at least to the seminal work by Russell and Whitehead [WR13]. The use of computer systems started in the 1950s and 1960s focusing on designing foundations that combine human and machine-friendliness. *Automated theorem proving*, going back to ideas by Newell, Simon, and Davis, has been most successful for first-order predicate logic and related languages. For more expressive languages that more adequately model mathematical knowledge, best results were reached in the automated *interactive verification of human-written proofs*, going back to ideas McCarthy, de Bruijn, Milner, and Martin-Löf. Modern QED-style deduction systems usually follow the interactive approach and employ automation support where possible.

Formalization pays off most at large scales due to the high level of theoretical understanding and practical investment that it requires from both developers and users. Therefore, today's flagship projects are built on double-digit person years of investment. (However, small formalizations are increasingly used to support individual papers at, e.g., the POPL or RDP conferences.) Even though this would naturally call for a community effort, 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 deduction systems such as system interoperability and massively-multi-user collaboration.

2.1 Deduction Systems

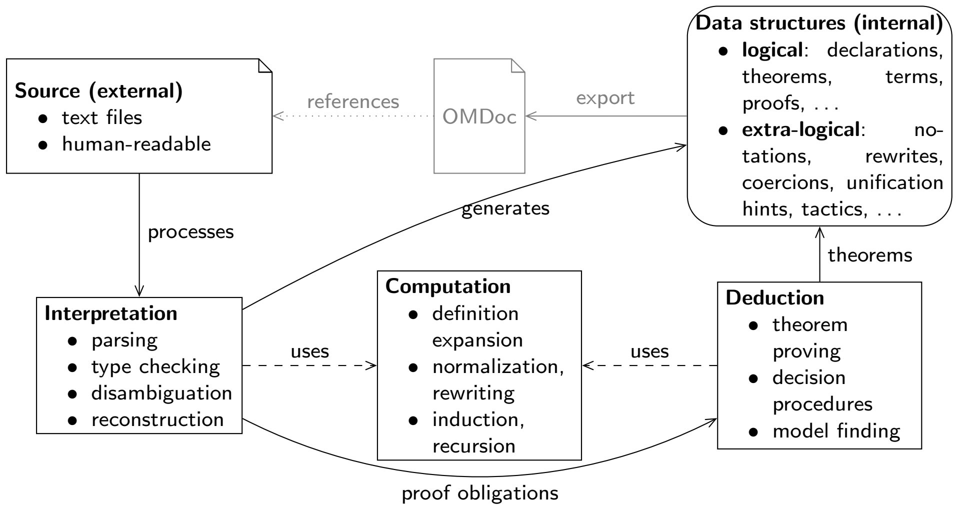


Fig. 1. Typical Architecture of a Deduction System

Even though there are different possible architectures for QED-style deduction systems, some recurring features can be identified. We survey a typical architecture (see Figure 1) in order to introduce names for the important concepts.

Formalizations are written in text files using concrete **source** syntax optimized for fast and convenient writing by human users. These differ from the abstract internal **data structures**, which model the logical foundation of the system. The most important work flow is the transformation of the source into logically verified data structures. Typically, three components are involved.

Firstly, a lot of the intelligence lies in the **interpretation** component, which bridges the representational gap between source and data structures. It is often split into parsing and type checking, but these may also be intertwined. Interpretation must mimic mathematical practices such as abuse of notation, context-sensitive disambiguation, omission of inferable parts, sloppy handling of undefinedness, or silent use of coercions. Typically this makes substantial use of extra-logical data structures that maintain notations, overloading, implicit argument declarations, type coercions, etc. A promising recent trend, e.g., heavily used in [GA+13], is the use of extra-logical data structures that trigger certain computations and controlled search procedures so that users can flexibly guide the interpretation process.

Secondly, the **deduction** component is traditionally the central part of the system. It employs complex decision and search procedures to discharge the proof obligations generate during interpretation and records the proved theorems in the data structures.

Thirdly, the **computation** component implements decidable normalization algorithms, user-configurable rewrite systems, or Turing-complete recursive functions. A major challenge is to understand when and which computations should be triggered during interpretation and deduction.

We also already indicate in gray the export of the data structures into our OMDoc interchange language. Its main purpose is to permanently store the otherwise-transient internal data structures and trace their origin from the sources. We will get back to that in Sect. 2.3.

2.2 Foundations

Overview. The notion of a **foundation** comes from the debate about the foundations of mathematics at the beginning of the 20th century. This tried to find a logical basis for mathematics and came up with first-order logic + set theory as one answer. *Philosophically*, a foundation should be very *simple* because we cannot determine the consistency of the foundation itself (except by reasoning in some other foundation). *Practically*, on the other hand, a foundation should be very *expressive* in order to allow structurally adequate and convenient formalizations. Traditionally, the implementation of a deduction system started with and centered around the data structures representing the foundation. The diverse group of foundations in today's QED-motivated systems are the result of trying to optimize the above simple-expressive trade-off.

It is crucial to observe that all of these systems are based on a **fixed foundation**, i.e., a fixed root logic in which all formalizations in that system are stated. Most of these derive from constructive type theories, higher-order logic, or first-order set theory. The constructive type theories are mostly based on Martin-Löf type theory [ML74] or the calculus of constructions [CH88] and make use of the Curry-Howard correspondence [CF58; How80] to treat propositions as types (and proofs as λ-terms). Systems include Nuprl [CA+86], Agda [Nor05], Coq [Tea], and Matita [ACTZ06]. The second group of systems go back to Church's higher-order logic [Chu40]. Systems include HOL4 [HOL4], ProofPower [Art], Isabelle/HOL [NPW02], and HOL Light [Har96]. Notably, only Mizar [TB85], Isabelle/ZF [PC93], and Metamath [MeMa] are based on variants of axiomatic set theory and thus most similar to the foundation of conventional mathematics. The foundation of the PVS system [ORS92] includes a variant of higher-order logics but with a significantly extended type system inspired by set theory. The IMPS system [FGT93] is based on a variant of higher-order logic with partial functions. The foundation of ACL2 [KMM00] is an untyped language based on Lisp.

Heterogeneous Reasoning. In principle – and according to mathematical folklore – most of mathematics can be formalized **homogeneously** in a giant collection of first-order formulas based on a small collection of set-theoretic axioms. Here all mathematical knowledge is formalized using only conservative extensions (e.g., definitions, theorems) of the fixed foundation.

A major drawback of this approach is that a lot of expensive formalization work is needed just to build the setting of interest (e.g., the real numbers) as a conservative extension of the fixed foundation. Moreover, this monolithic formalizations lack the structuring devices needed for coping with the size and complexity of the collection, with development workflows, and with the constantly changing presentation of mathematical knowledge. Therefore, conventional mathematics uses what we call the **heterogeneous method**. Going back to the works by Bourbaki [Bou64], this method focuses on defining axiomatic *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, while all mathematics can be reduced to first principles, it is usually carried out in highly abstracted settings that hide the foundation. This approach has been applied successfully in software engineering and algebraic specification, where formal module systems are used to build large theories out of little ones, e.g., in SML [MTHM97] and ASL [SW83].

QED systems support heterogeneity in various ways.

Several systems allow heterogeneity explicitly by introducing abstract theories, e.g., locales in Isabelle, parametric theories in PVS, modules in Coq, or structures in Mizar. IMPS [FGT93] even uses the heterogeneous method systematically as a central design feature.

Moreover, in all systems, one can apply the heterogeneous style *implicitly*. For example, one first introduces a new construct (e.g., the real numbers) by defining it in terms of existing ones (e.g., Cauchy sequences). Then one expands the definition while proving the characteristic theorems about the new construct. Finally, afterwards only those theorems are used, and the definitions are never expanded again. In particular, it becomes irrelevant, which out of several possible definitions (e.g., Dedekind cuts) was used. We call this *implicit* heterogeneity because this irrelevance is not made explicit (e.g., by introducing the second-order theory of the real numbers) in a way that would allow easy reuse across deduction systems.

Many systems provide special support for implicit heterogeneity by using definition principles. Here the interpreter elaborates high-level axiomatic declarations into conservative extensions. Example extension principles are type definitions in the HOL systems, provably terminating functions in Coq or Isabelle/HOL, or provably well-defined indirect definitions in Mizar. Often elaboration-based approaches have the effect that heterogeneity becomes quite explicit in the source syntax while remaining implicit in the data structures.

Finally, several data types provide a systematic way of using heterogeneity implicitly. (Co)inductive types are theory-like abstraction mechanisms because they are also described by a set of declarations – the constructors and selectors, respectively – and axioms for (co)induction. They have the advantage that they additionally allow performing computations in the underlying foundation. Similarly, record types can be used as an alternative to theories, e.g., as done by Mizar structures and with Coq records in the Mathematical Components project [MC]. This has the major advantage that no meta-formalism is needed for heterogeneous reasoning: Theory morphisms become first-class functions between record types.

The Incompatibility Problem. The combination of fixed foundations and implicit heterogeneity has become the dominant approach in the QED area. This has several reasons. Firstly, it allows integrating computation into deduction by evaluating terms in the foundation. This makes proofs much simpler because it eliminates the tedious step-wise application of axioms to reason about equality. Secondly, the homogeneous method avoids adding axioms, which would end up growing the untrusted code base when using code extraction. Thirdly, in some domains like code certification, it is hard to be sure that all the invariants are respected unless every single detail is formalized homogeneously.

But the resulting formalizations are not easily reusable across systems. Firstly, implicit heterogeneity makes it difficult to systematically identify the heterogeneous structure of a library, which would be necessary to guide reuse. Secondly, the foundation-specific features used to obtain heterogeneity in one system may not be present in another system. Therefore, it is today virtually impossible to exploit heterogeneity when moving theorems across systems or even across different libraries of the same system. Therefore, almost all current systems and libraries are mutually incompatible, with only a few ad hoc translations between them (e.g., [KW10; KS10]).

In principle, highly reusable formalizations can be developed best with the heterogeneous method. It maximizes the level of abstraction for each result, and theories can be understood uniformly in all foundations. But it remains an open question how to combine the best of both worlds. One option may be to write heterogeneous theories in richer logics that induce computations, e.g., by using rewrite rules.

In any case, while a fixed foundation may be reasonable for an individual deduction system, it is counter-productive for a universal library of mathematics because different domains may require different foundations. Similarly, different communities involved in the formalization effort may favor different foundations (maybe only because of the formalization styles or workflows they support). Thus the library level of QED would profit from **foundational pluralism**, i.e., the ability to support multiple foundations in a single universal library.

2.3 Formal Libraries

Overview. All major deduction systems provide support for managing formalizations in "libraries" (usually sets of source files) and collect some kind of library of formalizations. Often a certain basic library is loaded upon startup, and the user can load additional libraries on demand. The library mechanism can be decentralized with users developing and/or hosting individual libraries or centralized with a committee collecting and possibly curating the library.

A very sensitive issue here is backwards compatibility, i.e., the question whether a library is still readable after upgrading the main system. Only for centralized libraries, this can be guaranteed by the system developers. For example, in the L4 verification project [KA+10] (7 years, 390000 lines of Isabelle/HOL), Isabelle updates and change management turned out to be major problems [BDKK12].

The Isabelle and the Mizar groups maintain one centralized library each – the "Archive of Formal Proof" [AFP] and the "Mizar Mathematical Library" [MizLib], respectively. The Coq group maintains a similar set of contributions. These libraries contain individual formalizations with relatively few interdependencies.

Highly-integrated libraries are usually found as part of a single formalization project whose size required the development of a separate library. Even though these libraries started as auxiliary devices, they are valuable results in their own right – maybe even more valuable than the primary formalization. Examples are Tom Hales's formalizations in HOL Light for the Kepler conjecture [HA+15] and Georges Gonthier's work in Coq for the recently proved Feit-Thompson theorem [GA+13]. John Harrison's formalizations in his HOL Light system [Har96] and the NASA PVS library [PVS] have a similar flavor although they were not motivated by a single theorem but by a specific application domain. The latter is one of the biggest decentralized libraries, whose maintenance is disconnected from that of the system. All of the above, use variants of implicit heterogeneity.

Heterogeneous libraries are the IMPS library [FGT], the LATIN logic library [CH+11] by the authors, and the TPTP library [Sut09] of challenge problems for automated theorem provers. However, none of these enjoys the level of interpretation, deduction, and computation support developed for individual fixed foundations.

The OpenTheory format [Hur09] offers some support for heterogeneity in order to allow moving theorems between systems for higher-order logic (specifically HOL Light, HOL4, and ProofPower). It provides a generic representation format for proofs within higher-order logic that makes the dependency relation (i.e., the operators and theorems used by a theorem) explicit. The OpenTheory library comprises several theories that have been obtained by manually refactoring exports from HOL systems.

Library Integration. There are several facets of library integration. Firstly, one can *refactor a single library* to increase reuse through modularity, sharing, and inheritance. Typically, this amounts to using the heterogeneous method. Secondly, one can *align two (or more) libraries.* Here alignments are special relations that pair each concept of one library with the corresponding concept in the other one. Alignment relations may range from "subsumes the intent of" to "are equivalent and proofs can be shared". Thirdly, one can *merge two libraries.* Usually, this requires translating the libraries into a common language and then identifying and eliminating overlap between them.

No strong tool support is available for any of these facets. The state-of-the-art for refactoring a single library is manual ad hoc work by experts, maybe supported by simple search tools (often text-based). Merging libraries can hardly be attempted because the state-of-the-art is still short of satisfactory translations into common languages.

This is despite the large need for more integrated and easily reusable large libraries. For example, in Tom Hales's Flyspeck project [HA+15], his proof of the Kepler conjecture is formalized in HOL Light. But it relies on results achieved using Isabelle's reflection mechanism, which cannot be easily recreated in HOL Light. And this is an integration problem between two tools based on the same root logic!

Library Exports. In most cases, integration attempts falter already when trying to access the library in the first place. Here, we have two options: We can work with the sources or the internal data structures.

The usefulness of the sources is limited because interpretation, computation, and deduction are highly non-trivial algorithms. This has the effect that each source syntax can usually only be understood by a single system: the respective deduction system.

The data structures on the other hand are usually difficult to access from the outside. Deduction systems are typically realized as read-evaluate-print interfaces to the data structures, optimized for batch-processing source files, and appear to the outside as monolithic black boxes. In particular, the individual components are tightly integrated and practically inseparable from the overall system.

Therefore, an export of the internal data structures is the only way to obtain machine-level access to the libraries. This is indicated in Figure 1, where we use our OMDoc format as an example for a standardized interchange format that holds the exported data structures.

However, many deduction systems do not provide any export, let alone an export to OMDoc. For example, in the case of Mizar, it proved notoriously difficult [DW97; BK07; Urb06] until the authors obtained an export [IKRU13] that they could actually make use of in their applications.

And even where exports exist – for example, Coq, HOL Light, Mizar, and PVS provide idiosyncratic, system-near exports using XML syntax or binary files – they have several problems.

Firstly, the use of elaboration means that often many parts of the data structures are the result of non-trivial transformations, and their structure can differ substantially from that of the source syntax. In particular, it may be difficult to infer the implicit heterogeneous structure (even it is was explicit in the source) that would be valuable for reuse.

Secondly, the exports quickly become out-of-date as new features are added to the main system. The only exception are exports that are actively maintained by the main developers, but this is rarely the case. Even the Mizar XML export has gotten somewhat out-of-sync recently although the XML export was tightly integrated with (and thus essential for) the main system.

Thirdly, there is usually little support for connecting the human-optimized source syntax with the machine-optimized export syntax. To let human users interoperate with libraries of other systems, however, this is exactly what is needed. One compromise here is to export *source references*. If every fragment of the data structures remembers the exact range in the source file that was interpreted to that fragment, then such source references can be added to the export. This allows at least localizing change or conflict notifications in the source – and thus to integrate them into the library management facilities of the system.

Library Translations. For importing fragments of a library L into a library L', we need an export facility for L, an import facility for L', and additionally one of two things: a translation between the underlying logics or a logical framework that can handle libraries in multiple logics. Both are rare. Therefore, there are only a few examples of bridging libraries between two large deduction systems.

A small number of library bridges have been realized using ad-hoc logic translations, typically in special situations. [KW10] translates from HOL Light [Har96] to Coq [Tea] and [OS06] to Isabelle/HOL. Both translations benefit from the well-developed HOL Light export and the simplicity of the HOL Light foundation. [KS10] translates from Isabelle/HOL [NPW02] to Isabelle/ZF [PC93]. Here import and export are aided by the use of a logical framework to represent the logics. The Coq library has been imported into Matita once, aided by the fact that both use very similar foundations. The OpenTheory format [Hur09] facilitates sharing between HOL-based systems but has not been used extensively.

The second approach requires a logical framework in which the logics can be represented. Then it is straightforward to map L to the library mechanisms of the framework. Then the framework can serve as a uniform intermediate data structure, via which other systems import L. The authors used this approach in [IKRU13] for Mizar and in [KR14] for HOL Light. These used the logical framework LF [HHP93] and made the libraries available to knowledge management services. Another example is the Dedukti system [BCH12], which imports, e.g., Coq and HOL Light into a similar logical framework, namely LF extended with rewriting.

The Library Integration Problem. Even when an export-import pair is available, it is usually still very difficult to integrate libraries due to what we dub the *library integration problem*.

The prevalent use of implicit heterogeneity means that results of L that are interesting to reuse in L' depend on a chain of conservative extensions all the way down to the foundation underlying L. This is disastrous for integration in the typical case where different libraries use different definitions (e.g., Dedekind cuts vs. Cauchy sequences). Thus, a logic translation will usually not map the real numbers of one system to the real numbers of another system. In explicitly heterogeneous libraries, on the other hand, each result depends only on some axiomatic theories, which can be mapped more easily to corresponding theories in L'.

Very little work exists to address this problem. In [OS06], some support for library integration was present: Defined identifiers could be mapped to arbitrary identifiers ignoring their definition. No semantic analysis was needed because the translated proofs were rechecked by the importing system anyway. This approach was revisited and improved in [KK13], which systematically aligned the concepts of the basic HOL Light library with their Isabelle/HOL counterparts and proved the equivalence in Isabelle/HOL. The approach was further improved in [GK14] by using machine learning to identify large sets of further alignments.

The OpenTheory format [Hur09] provides representational primitives that, while not explicitly using theories, effectively permit heterogeneous developments in HOL. The bottleneck here is manually refactoring the existing homogeneous libraries to make use of heterogeneity.

We sketched a partial solution aimed at overcoming the integration problem in [RKS11].

2.4 Logical Frameworks

In response to the plurality of logical systems, logical frameworks have been introduced. These are formalisms in which the logical parts of the data structures (but usually not the concrete source syntax and not all extra-logical parts) of logics can be defined. [Pfe01] gives an overview.

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 [CH+11].

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. Moreover, rewriting can be used to integrate computation into the logical framework. A number of logic libraries have been exported to Dedukti, which is envisioned as a universal proof checker.

Isabelle [Pau94] implements intuitionistic higher-order logic, which (if seen as a pure type system with propositions-as-types) is rather similar to LF. 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 and Isabelle/ZF.

All logical frameworks allow representing logics as theories. This allows using the heterogeneous method to build logics from small components and to express logic translations as theory morphisms in the logical framework. The authors' LATIN library [CH+11] systematically employs this approach to formalize a large collection of logics.

However, state-of-the-art logical frameworks are not expressive enough yet to allow natural formalizations of the complex foundations underlying state-of-the-art deduction systems. For example, the LATIN project could not obtain full formalizations of the major foundations in LF.

Moreover, logical frameworks cannot replace foundation-specific systems because the increase in generality always unavoidably leads to additional overhead and a decrease in efficiency. In particular, logical frameworks lack strong support for computation and thus cannot represent proofs efficiently that rely on computation. The use of rewriting in Dedukti is motivated by this shortcoming. Therefore, several systems that are expressive enough to act as logical frameworks such as Coq or Nuprl have not pursued these avenues.

2.5 Knowledge Management (KM)

KM Languages. Mathematical knowledge, research, and applications are distributed globally, and mathematical knowledge is highly interlinked by explicit and implicit references and citations. Therefore, a computer-supported management system should support global interlinking and scale to very large libraries. Yet, virtually all current deduction systems operate under the implicit assumption that all relevant knowledge is locally available and can be loaded into main memory – i.e., at local scale.

On the other hand, there are representation languages developed for global scale mathematics. OpenMath [BC+04] and MathML [AB+10] provide general definitions of the concrete and abstract syntax of mathematical objects. OMDoc [Koh06] extends these with abstract and concrete syntax for statements, heterogeneous theory development, informal knowledge, and mathematical documents. These languages can be customized by content dictionaries, which introduce additional primitives and describe their semantics. While there are only very few and weak content dictionaries that were natively written in these languages, content dictionaries can be generated automatically from heterogeneous formalizations as we do in [HR15].

These languages have been used as system-level interchange formats for mathematical/symbolic systems (e.g., the use of OpenMath in the SCIEnce project [HR09]), as a basis for integrating mathematics with the semantic web (e.g., in the MONET FP6 project and the HELM/MoWGLI FP6 project), or as markup languages for web browsers (e.g., by the integration of MathML into HTML5). They are the basis of generic assistant systems such as MathWebSearch [KŞ06] for search or ActiveMath [MB+03] for user-adaptive learning. Many mathematical assistant systems from symbolic computation or (to a much smaller extent) formal deduction can use them for the export or import of their knowledge.

KM for Deduction Systems. As a rough general rule, deduction systems fare badly on KM challenges like large scale collaborative projects, change and distribution management, and integrated development environments. Retro-fitting them with KM support has proved very expensive and unsatisfactory. In fact, because these systems have initially focused on soundness and efficiency at local scales, large scale KM has proved very difficult to add as an afterthought, often prohibitively so. Moreover, 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. It is no coincidence that Matita [ACTZ06] — one of the few major new systems of the last decade — is the most KM-friendly among them.

More concretely, most proof assistants come with some support for searching statements (usually a plain text search of the source files or a search for identifiers with certain properties after loading the library) and ad hoc change management (usually based on file-modified timestamps). Most systems are able to export their library as browsing-oriented HTML, using styles, cross-references, and visibility management.

These services tend to be low-level services based on data structures that are close to the plain text source (e.g., text-based search) or high-level services based on the volatile in-memory data structures (e.g., searching for identifiers with certain types). The systems often lack persistent high-level representations of the library (e.g., in OMDoc) that could serve as the basis for generic large scale KM services that can be developed and run independent of the deduction system. If such representations exist, such as Coq's binary or Mizar's XML files, their KM potential is not fully exploited, often due to a lack of resources among the experts and a lack of documentation for outsiders.

One of the most advanced system-specific solutions is the automated reasoning service for HOL Light [KU13], which uses machine learning to select from a large knowledge base of theorems those that can help to prove an open proof obligation automatically. One of the most advanced system-independent KM projects in this area is [ABMU11], which develops a general Wiki infrastructure that is applied to Mizar and Coq.

For Isabelle, a partial reimplementation in a different programming language (Scala instead of ML) permitted a tighter coupling between deduction kernel and KM applications (in this case authoring support) [Wen12]. A related line of work built asynchronous theorem proving [Wen10], which allows delaying all checks that are only relevant to the correctness of the theory but do not have immediate bearing on the user interface. This allows a systematic change management where small changes to large formalizations can be made more efficiently.

3. THE OMDOC/MMT LANGUAGE AND SYSTEM

3.1 A Universal Language for Pluralistic Mathematical Libraries

We have developed the OMDoc XML format [Koh06], which can serve as a standardized representation language of a QED-style database. In particular, it can represent the exports of deduction system libraries and their documentation. In the last five years we re-developed the fragment of OMDoc pertaining to formal knowledge resulting in the Mmt language [RK13; HKR12; Rab14b]. Mmt greatly extends the expressivity, clarifies the representational primitives, and formally defines the semantics of this OMDoc fragment.

Mmt introduces a rigorous, **foundation-independent** conceptualization of the data structures of deduction systems. It systematically avoids any commitment to a particular foundation, abstracts from and mediates between different foundations, and thus maximizes the reuse of concepts, tools, and formalizations. As such, Mmt provides a uniform solution to the root logic controversy of QED.

More concretely, Mmt *integrates successful representational paradigms*

- the logics-as-theories representation from logical frameworks,
- theories and the reuse along theory morphisms from the heterogeneous method,
- the Curry-Howard correspondence from type theoretical foundations,
- URIs as globally unique logical identifiers from OpenMath,
- the standardized XML-based interchange syntax of OMDoc,

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

$$\begin{tikzcd}
{} \arrow[r, "\text{translation}"] & {} \\
{} \arrow[r, dashed, "\text{import}"] & {} \\
{} \arrow[r, dotted, "\text{meta-theory}"] & {}
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & \mathtt{LF} \arrow[r, dashed, "m"] \arrow[dl, dotted] \arrow[dr, dotted] & \mathtt{LF}+X \\
 & \mathtt{FOL} \arrow[rr, "m'"] \arrow[dl, dotted] \arrow[d, dotted] \arrow[drr, dotted] & & \mathtt{HOL} \\
\mathtt{Monoid} \arrow[r] \arrow[rrr, dashed, bend left, "\mathtt{mult}"] & \mathtt{cgroup} \arrow[rr, dashed, bend right, "\mathtt{add}"'] & & \mathtt{Ring}
\end{tikzcd}$$

Fig. 2. Meta-Theories and Theory Morphisms in MMT

possibly with additional attributes such as type, definiens, or notation. Relative to a theory, **objects** are formed as syntax trees with binding.

Although further work remains regarding certain features (e.g., computation), these three concepts already allow natural representations of most 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.

MMT theories can be related to each other via **theory morphisms**, which are used to uniformly describe the **modular structure** of and **representation theorems** between theories. Regarding modular structure, theory morphisms occur as import declarations within theories, which uniformly represent, e.g., inheritance and instantiation. Regarding representation theorems, theory morphisms occur as translations, functorial representations, implementations, and models.

MMT maintains a formal **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 used as the meta-theory of another one. 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 in Fig. 2 gives an example, where $\mathtt{LF}+X$ represents some extension of LF with additional features.

The combination of foundation-independence and theory morphisms makes MMT very suitable for building a universal community-driven QED database. Practically, this makes the foundation (root logic) a parameter of the formalization, which enables an inclusive "bring-your-own-foundation" approach to community building.

Moreover, it becomes easier to design new root logics because the Mmt module system can be reused. Theoretically, theory morphisms allow relating theories across foundations, a crucial prerequisite for the systematic integration of systems and libraries.

We can see Mmt as the next step in a **progression towards more abstract formalisms** as indicated in the table below. In conventional mathematics (first column), domain knowledge is expressed directly in ad hoc notation. Logic (second column) provided a formal syntax and semantics for this notation. Logical frameworks (third column) provided a formal meta-logic in which to define this syntax and semantics. Now Mmt (fourth column) adds a meta-meta-level, at which we can design even the logical frameworks flexibly. (This meta-meta-level gives rise to the name Mmt with the last letter representing both the underlying theory and the practical tool.) That makes Mmt very robust against future language developments: We can, e.g., develop `LF` $+X$ without any change to the Mmt infrastructure and can easily migrate all results obtained within `LF`.

Mathematics	Logic	Meta-Logic	Foundation-Independence
domain knowledge	logic domain knowledge	logical framework logic domain knowledge	Mmt logical framework logic domain knowledge

The increased level of generality in Mmt makes it harder to obtain Mmt-level results because any result has to be generalized to apply to every foundation. Indeed, one might think that it is not possible to obtain deep, meaningful results at all. However, as exemplified below, practical experience has shown that *many results can be generalized* to the Mmt level. This expands on existing experiences with logical frameworks – for example, Isabelle contains ~ 40 ML files with logic-independent functionality such as for induction or code extraction.

For each result, this may require a substantial research effort, which samples existing results for specific foundations and integrates them into a general result. 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.

3.2 Foundation-Independent Implementation

Exploiting the small number of primitives in the Mmt language, the Mmt API [Rab13b] provides a simple scalable implementation of Mmt (written in the functional and object-oriented language Scala [OSV07]), which serves as the kernel of a tool suite for the creation and life-cycle management of OMDoc/Mmt-encoded libraries of mathematical knowledge.

It implements data structures, algorithms for interpretation, computation, and (very experimentally) deduction, as well as knowledge management support foundation-independently. For each algorithm, foundation-specific aspects (if any) are supplied by plugins via various plugin interfaces, which often take the form of sets of rules. Our 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 $< 2,000$ lines. Thus much functionality can be made available to individual foundations at small cost.

Logical results obtained generically at the MMT level include the fundamental concepts of logic [Rab14b], module system [RK13] and theory transformations [Hor14], literals [Rab15a], notation-based parsing, type reconstruction and simplification, and a (so far) very basic theorem prover. (The latter results have not been published independently but are part of the MMT system.) [Rab15b] generalizes the method of logical relations to the "almost MMT" level, i.e., it makes some mild assumptions about the foundations.

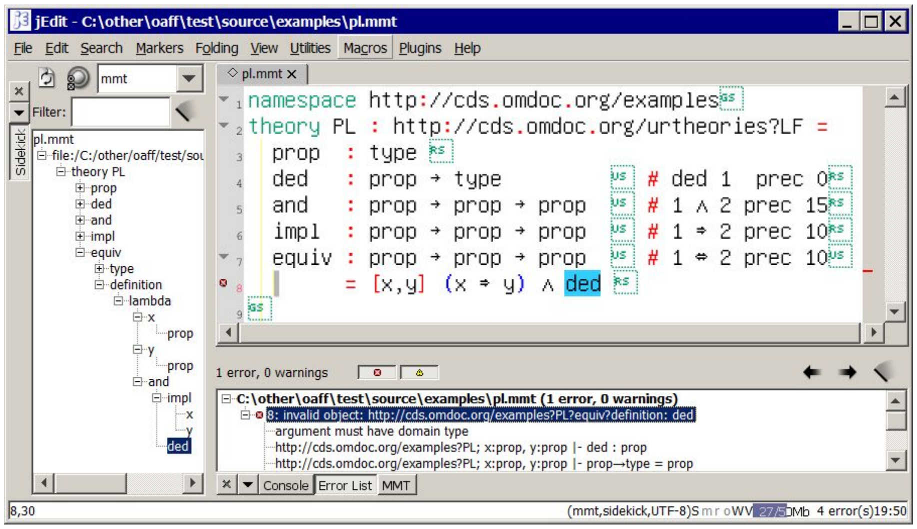
These results can be used for rapidly prototyping new foundations. This can help build a variety of specialized deduction systems that make small contributions to a large QED database. But no foundation-independent implementation can compete with foundation-specific ones for tasks such as proof development and proof checking, which are efficiency-critical and prone to combinatorial explosions. Therefore, MMT is most useful in those cases where foundation-specific solutions do not exist in the first place, e.g., for new or experimental foundations.

In any case, the most important application of MMT as a foundation-independent deduction system is to *implement logical frameworks* and use those to formalize the foundations of existing deduction systems.

For example, MMT's type reconstruction algorithm handles implicit arguments and unsolved meta-variables, constraint delay, and error reporting foundation-independently. The plugin that implements the logical framework LF supplies only ~ 10 LF-specific rules of a few lines each, which correspond directly to the usual typing and equality rules for Π, λ, and application. To instantiate the algorithm with other logical frameworks, we only have to change the set of rules. For example, any other pure type system can be implemented accordingly by changing the type inference rule for Π, and shallow (rank-1) polymorphism is obtained by adding a single rule. MMT's computation algorithm is also rule-based and integrated with type reconstruction; therefore, Dedukti-style type theories modulo are obtained simply by adding rewrite rules. We believe this approach will extend to more complex foundations (including, e.g., universe hierarchies, induction, or subtyping), but we have not investigated that yet.

Contrary to logical results, **knowledge management** results typically require no foundation-specific customization and thus offer higher pay-offs when applied to existing foundation-specific libraries. Results implemented generically at the MMT level include project management and build system [HI+11], IDE (build by integrating MMT with the jEdit text editor) [Rab14a], interactive browsing using HTML+presentation MathML and JavaScript [GLR09], change management [IR12], as well as indexing, querying, and search [Rab12; KMP12; KI12]. 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 and unloaded from memory dynamically and transparently, and the distribution of content over physical storage and networks remains transparent to MMT-based services.


jEdit - C:\other\oaff\test\source\examples\pl.mmt
File Edit Search Markers Folding View Utilities Macros Plugins Help
mmt
Filter:
Sidekick
pl.mmt
Console Error List MMT

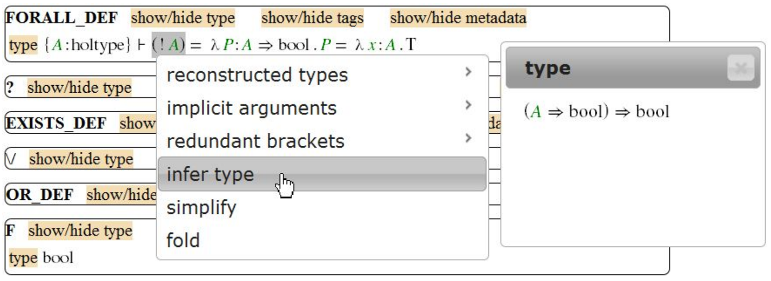
```
namespace http://cds.omdoc.org/examples
theory PL : http://cds.omdoc.org/urtheories?LF =
  prop  : type
  ded   : prop → type                 # ded 1  prec 0
  and   : prop → prop → prop          # 1 ∧ 2 prec 15
  impl  : prop → prop → prop          # 1 ⇒ 2 prec 10
  equiv : prop → prop → prop          # 1 ⇔ 2 prec 10
        = [x,y] (x ⇒ y) ∧ ded
```

```
pl.mmt
  file:/C:/other/oaff/test/sou
    theory PL
      prop
      ded
      and
      impl
      equiv
        type
        definition
          lambda
            x
              prop
            y
              prop
            and
              impl
                x
                y
              ded
```

```
1 error, 0 warnings
C:\other\oaff\test\source\examples\pl.mmt (1 error, 0 warnings)
  8: invalid object: http://cds.omdoc.org/examples?PL?equiv?definition: ded
    argument must have domain type
    http://cds.omdoc.org/examples?PL; x:prop, y:prop |- ded : prop
    http://cds.omdoc.org/examples?PL; x:prop, y:prop |- prop→type = prop
```

```
8,30                                   (mmt,sidekick,UTF-8)Smr oWV 27/50Mb 4 error(s)19:50
```

Fig. 3. Mmt IDE based on jEdit



type $\{A : \text{holtype}\} \vdash (!\,A) = \lambda\, P : A \Rightarrow \text{bool}\,.\, P = \lambda\, x : A\,.\,\text{T}$

$(A \Rightarrow \text{bool}) \Rightarrow \text{bool}$

type bool

Fig. 4. Type Inference in the Mmt Web Server

For example, the Mmt-based foundation-independent IDE is realized as a plugin to the jEdit text editor. It allows defining logics in any logical framework that is implemented within Mmt. Figure 3 shows a definition of propositional logic with meta-theory LF. An – intentionally introduced – error was detected by type reconstruction and highlighted in blue. 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 expression failed. Other features include hyperlinks, type inference tooltips for selected subexpressions, context-sensitive auto-completion, or interactively solving for missing subexpressions based on the expected type. Mmt's design makes it possible to realize this advanced functionality using only $< 1,000$ lines of glue code between jEdit and Mmt.

Another example is the Mmt web server. Fig. 4 shows a fragment of the HOL Light library as imported into Mmt in [KR14]. It shows how the context menu is used to interactively call type inference on 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 or of an included theory.

The web server can be run as a dedicated server or locally. For example, it can be run from within the jEdit IDE, in which case browser and editor become connected, e.g., for synchronous navigation.

4. THE MATHHUB.INFO ARCHIVE AND PORTAL

The MMT system described in Section 3 can be used to give individual users access to a mathematical library and supports their knowledge management workflows. But a full-scale, global QED database requires user/rights management, distributed revision control, and Web 3.0 features (e.g., discussions and user-generated annotations). For that purpose, we introduce the MathHub.info system. It is realized as an instance of the Planetary system [Koh12], which we have substantially extended in the course of the work reported here.

4.1 Architecture

MathHub.info has four main components (see Figure 5):

i) a versioned *backend* holds the libraries,

ii) MMT as the kernel tool understands the libraries provides semantic services for them,

iii) a web-based *frontend* makes the libraries and services available to users,

iv) a Javascript *plugin architecture* enriches document presentations with localized semantic services.

We use best-of-breed open source systems for the components going beyond MMT. In the backend, we use GIT for versioning, distribution, and user/rights management adapting the GitLab repository manager [GL], an open-source alternative to GitHub. For the frontend, we use the Drupal container management system.[1] For the Javascript library we use our JOBAD framework [GLR09; Koh12], which embeds semantic services into HTML documents and thus makes them interactive and user-adaptive. Even though JOBAD is just a relatively thin layer of glue code that picks up on semantic annotations in the generated HTML5, its effect for the users is rather profound: It gives them access to added-value services "at the point of pain/interest", i.e., in the user interface. Figure 4 shows JOBAD in action: it links fragments of the formula presentations with computations in the MMT system and makes both available to the user embedded in the document.

Figure 5 shows the detailed architecture. Here GitLab provides distributed versioned storage of the libraries and organizes them into repositories owned by users and groups. And Drupal supplies uniform theming, discussion forums, and a plugin infrastructure for adding interface functionality. Both systems provide user management, but we automatically synchronize the users and permissions between them.

[1]Drupal and similar systems self-describe as "content management systems", but they actually only manage the documents and their metadata – essentially document containers – without changing their internal structure.

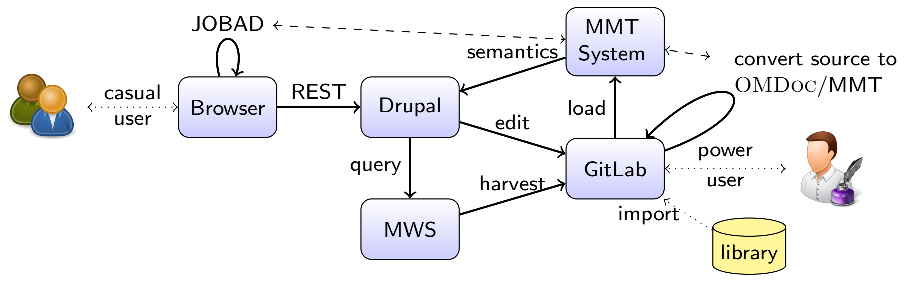


Fig. 5. The modular MathHub.info Architecture

This componentized architecture has the advantage that we can combine two methods for accessing the contents of MathHub.info: *i*) an online, web-based workflow for the casual user, and *ii*) an offline authoring workflow based on git working copies for power users and bulk edits. Users can fork or pull the relevant repositories from GitLab, edit them, and submit them back to MathHub.info either via a pull request to the repository masters or a direct commit/push. As the content is often highly interlinked and distributed across multiple interdependent repositories, we have developed tool support for managing multiple working copies across repository borders.

The interactive functionalities in MathHub.info are based on the OMDoc/Mmt representation of the libraries, but authors and users have to interact with them in the respective source language of the library. Both the source and OMDoc/Mmt representations are versioned in GitLab and the respective source representations must be converted into OMDoc/Mmt by language-specific custom exporters. Correspondingly library import is managed by MathHub.info at the level of the GitLab repository.

Then we can dedicate a specific git working copy together with an Mmt instance to a user or a group that shares permissions. Thus, the Mmt instance sees (and takes into account for its services) only the documents accessible to the group. If an authenticated user edits MathHub.info content, the changes are committed under his name into the specific working copy. This makes it easy to cope with multiple synchronous users, for which MathHub.info uses separate working git clones and Mmt instances.

4.2 Content

MathHub.info is intended as a portal and archive for both QED-style formalized mathematics and *flexiformal* mathematics, i.e., informal or partially formalized documents. To deal with flexiformal mathematical content, we have also extended the Mmt API to allow informal fragments and to degrade gracefully in their presence. For example, Mmt's type reconstructions works on those subexpressions for which type information is available, and Mmt's change management for those documents for which dependency information is available.

We are currently hosting a nucleus of formal and flexiformal libraries and their OMDoc/Mmt representations to develop and evaluate the functionality. Concretely, these are

i) SMGloM [Koh14], a flexiformal glossary containing about 1500 definitions and notation written in sTeX [Koh08] (semantically annotated LaTeX).
ii) about 6500 flexiformal sTeX files containing teaching materials (slides, course notes, problems, and solutions) in computer science,
iii) the LATIN logic library [CH+11] with about 1000 modules,
iv) the TPTP library [Sut09] of over $20,000$ theorem proving challenge problems,
v) the Mizar Mathematical Library [TB85] of about 1,000 articles and 50,000 statements, and
vi) the HOL Light Library of about 200 files and over 15,000 statements,
vii) very recent and still partial versions of the Open Encyclopedia of Integer Sequences (OEIS) [Slo03] and the libraries of PVS [ORS92], Specware [SJ95], and Theorema [Win14].

In the future we want to open the portal up for user-supplied content, eliciting documents from mathematics and nearby disciplines. We anticipate that MathHub.info may be attractive to authors because it *i)* offers free private repositories, *ii)* allows transforming mathematical papers into hosted, searchable HTML5 documents, *iii)* allows adding interactivity by semantic annotations in a stepwise fashion.

But ultimately, we are interested in a communal resource, in which the document sources are available for inspection, re-use and semantic analysis. Therefore, the free private repositories are in what we call *public escrow*: They are private as long as the user actively requests them to be. Otherwise, they are published under a copyleft license of the user's choice.

Even though the MathHub.info system is more general than necessary for a QED-style database of formal mathematics, it is well-suited for it. In fact, we expect that the inclusion of informal documents will support stepwise formalization workflows and allow interlinking conventional mathematical texts with their formalizations.

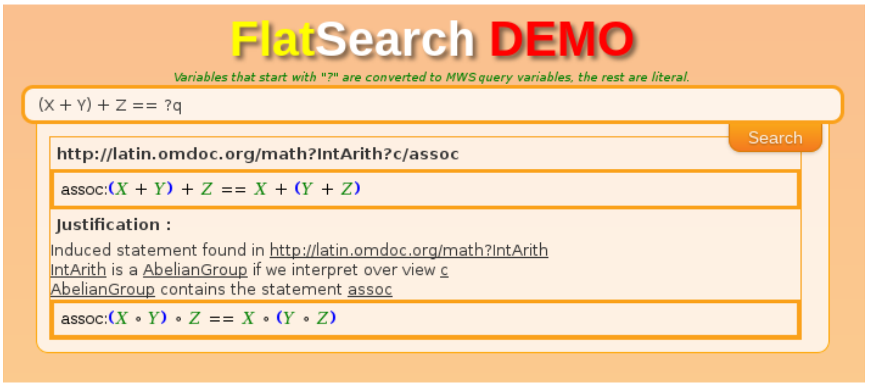
4.3 Global Services in MathHub.info

Like the deduction systems surveyed above, the MMT system provides its services based on data structures in main memory – essentially a must for reasoning and knowledge management services – even though the OMDoc/MMT language supports global scope and linking via its XML-based syntax and MMT URIs. Essentially, the memory consumption of the MMT system adapts dynamically to the scope of the respective user's field of attention.

The MathHub.info system can also integrate services that have a global view, i.e., services that range over all the libraries in MathHub.info. Such services usually operate on a special, reduced representation of content – e.g., the dependency graph. This representation is often produced by mapping a translation service provided by the MMT system over the library; this is very efficient because many of the MMT-based algorithms are systematically incremental, i.e., they can translate a single file or theory without loading its dependencies.

A good example of a global-scope service is the MathWebSearch service (MWS; [KŞ06]), a formula search engine. It consists of a web service that harvests formulae from all MathHub.info content and submits them to a specialized substitution tree index, which can be efficiently queried by the user (cf. Figure 5). In contrast to local services, which load content into memory on demand, MWS must keep the whole index in memory [KMP12].

In MathHub.info, we are using an extension (♭search [KI12; IK15]) of MWS, which makes use of the modular structure in OMDoc/Mmt encoded libraries: ♭search uses the Mmt system to flatten all theories (essentially copying over all inherited statements) creating a monolithic knowledge space, which can then be indexed in the regular MWS engine. In this way, we can search the exponentially larger knowledge space induced by the modular theories hosted in MathHub.info.



Variables that start with "?" are converted to MWS query variables, the rest are literal.

(X + Y) + Z == ?q

http://latin.omdoc.org/math?IntArith?c/assoc

assoc:$(X + Y) + Z == X + (Y + Z)$

Justification :

Induced statement found in http://latin.omdoc.org/math?IntArith
IntArith is a AbelianGroup if we interpret over view c
AbelianGroup contains the statement assoc

assoc:$(X \circ Y) \circ Z == X \circ (Y \circ Z)$

Fig. 6. ♭search on a Theory Graph similar to the one in Figure 2

Figure 6 shows a query for the associativity of integer addition, and shows a hit in the theory IntArith, which inherits the associativity axiom from the theory AbelianGroup. This alleviates a common concern with the heterogeneous method: that the space-efficient representation given by reuse of theories in modular structures hinders accessibility of contents. By using Mmt and ♭search, the heterogeneous structure of formalizations can remain transparent to casual users.

5. QED RELOADED

With the Mmt representation format and system and the MathHub.info infrastructure, we have first versions of all the critical pieces for a modern incarnation of the QED database. In this section, we show how they can work together to form a pluralistic formal library system of mathematical knowledge. We also discuss current short-comings, in particular regarding the representation of complex foundations and library integration. In any case, actually filling this infrastructure with the QED content remains a substantial community effort.

5.1 Representing Foundations

The LATIN project [CH+11] built a heterogeneous, highly integrated library of formalizations of logics and related languages as well as translations between them. It represented logics as Mmt theories with the logical framework LF as the meta-theory. Fig. 7 shows a high-level view of a fragment of the resulting diagram of Mmt theories, where $\hookrightarrow$ represents inclusions and $\rightarrow$ other theory morphisms, and the meta-theory `LF` is omitted. The left side shows some of the logics. For example, `FOL` is included into `ZFC` set theory because it is its meta-theory; and the model theoretical semantics of `FOL` is given by a second morphism into `ZFC`, which factors through HOL. All logics are defined modularly, and the middle zooms into the modular definition of propositional logic, which arises by importing each connective. The right side zooms in further to show the definition of conjunction as a triple of syntax (the binary connective and its notation), proof theory (the introduction and elimination rules), and model theory (the cases of the interpretation function).

The **logics** formalized in LATIN 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.

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 LATIN library also allows representing model theoretical semantics as theory morphisms from a logic into a theory representing a foundation [Rab13a]. A paradigmatic example for first-order logic and its proof- and set-theoretical semantics was published as [HR11]. The **foundations** in LATIN include Zermelo-Fraenkel set theory, Church's higher-order logic, and Mizar's formalized set theory [IR11]. These representations can also double as a documentation layer for the foundations.

Representing a foundation in a logical framework can be very difficult. Often it requires a deeper understanding of the logic and its implementation than published in the literature. Moreover, state-of-the-art logical frameworks such as LF or Isabelle are not strong enough to represent all features of the typical foundations of

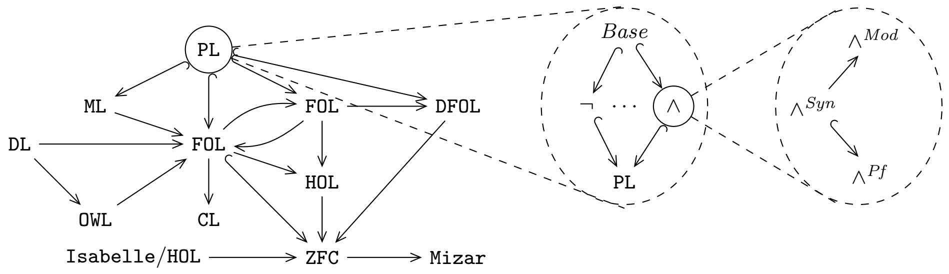


Fig. 7. A Fragment of the LATIN Atlas

QED systems. Already Mizar's soft type system cannot be represented elegantly. Similarly, inductive and record types, subtyping and implicit coercions, and computation and reflection are notoriously difficult to represent in logical frameworks. Moreover, sometimes unexpected features prove important: We designed a version of LF with sequences [HRK14] in response to practical needs.

This is the main motivation for MMT to allow plurality even at the logical frameworks level: It lets us gradually design the logical frameworks necessary to represent the various foundations while already building the overall QED database. Our practical experience has confirmed the feasibility of this approach: Originally, MMT only supported LF, and additional features such as pure type systems, shallow polymorphism, and rewriting could be added very easily.

5.2 Representing Libraries

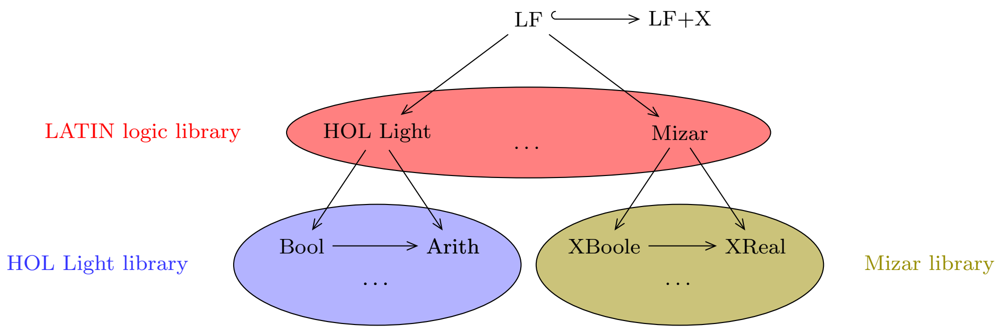


Fig. 8. Representing Libraries in MMT

We anticipate that most contributions to the reloaded QED database will be produced by foundation-optimized systems even though our proposed framework is foundation-independent. Therefore, the development of QED libraries should continue in foundation-specific systems, and we propose extending these systems with export work flows that publish their libraries to our central QED database.

Concretely, after representing the foundation F of a deduction system in an appropriate logical framework, we can add an export function to the system that outputs its library as a set of MMT-theories with meta-theory F. In our experience, the specification of these exports is relatively simple except for handling a few idiosyncratic features that each foundation tends to have. However, their implementation and maintenance is very expensive and can even be impossible without refactoring the system. Therefore, substantial collaboration from within the respective developer community is indispensable.

The most advanced library exports at this point are those for Mizar [IKRU13] and HOL Light [KR14], resulting in the MMT diagram of Fig. 8. If our proposal for a pluralistic QED library gains traction, it will become a maturity signal for a deduction system to support such an export work flow.

We do not expect a "back-translation" from OMDOC to the source syntax in the near future – indeed this is usually very difficult in our experience. Moreover, the curation and extension of the various libraries will continue to operate on the respective source files (rather than the exported OMDoc files) using the existing foundation-specific work flows. Therefore, the QED database should store both the source and the OMDoc files, and the latter should include fine-granular source references. Then generic library management functions can operate on the OMDoc representation and use the source references to annotate source fragments, e.g., with refactoring suggestions or user comments.

5.3 Integrating QED Libraries across Foundations

In principle, deduction systems can use the QED database to import results from libraries with compatible foundations or adapt proofs and proof scripts by analogy. However, this is still hampered by the library integration problem. We believe our pluralistic database is necessary for systematically attacking this problem. In fact, we might even say that we need it just to develop the tools for surveying the extent of the problem. We propose two methods for working towards a solution.

Alignments. We subscribe to the analogy that considers conventional mathematics as akin to (informal) software specification and formalized mathematics as akin to implementations. Consequently, the different foundations correspond to the different programming languages used to implement a specification.

Now the flexiformal MathHub.info system that we suggest for QED already permits representing both the formalizations and the conventional mathematics. In fact, we regard the flexiformal glossary SMGloM mentioned in Section 4.2 as the kernel of a collection of informal reference specifications: SMGloM is a theory-graph structured set of theories, which contain flexiformal definitions for standard mathematical concepts and objects [Koh14]. As such, they abstract away from many of the foundational choices we have to make in formalization. Thus, we can link each concept in a formalization to the glossary concept that it “implements”.

Then we can define two symbols in two libraries to be **aligned** if they are linked to the same concept in the glossary. Alignment will have to be a very complex relation. For example, HOL Light has a type of booleans, which is aligned with two concepts in Mizar: the type first-order propositions and the 2-element set of booleans. Conversely, Mizar’s number 0 is aligned with multiple numbers in HOL Light including the natural and the real number 0.

Therefore, alignments will not induce formal translations between libraries. But they will still allow a variety of important services, such as:

- Mathematicians can use common mathematical syntax to search all QED libraries up to alignment.
- Formalizers can look for useful theorems formalized in other systems. This may even allow to automatically transport proof ideas, e.g., by using a heuristic that uses alignments to translate essential lemmas.
- If the types of aligned constants are aligned as well, libraries of one system can be presented in the notations of another.

Interface Theories. The method of alignments can be deepened to what we call **interface logics and interface theories**. Following the above analogy, these correspond to the use of formal specifications in software engineering.

Interface logics will be the minimal logics needed to state a problem and thus will be much less expressive than common foundations. This is no coincidence: Whereas foundations are designed to be simple and expressive at the same time (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. For example, axiomatic set theory, higher-order logic, and constructive type theory are common foundations. But we can specify the real numbers using a much less expressive interface logic, such as monadic second-order logic, whose definition may be more involved. While giving a logic translation between any two typical foundations can be prohibitively difficult, a weak interface logic can be easily translated into many foundations. The LATIN logic library already constitutes a large heterogeneous library of interface logics.

Similarly, interface theories will use interface logics as their meta-theories and differ from QED-style formalizations by excluding any definitions and proofs. For example, the interface theory for the real numbers will declare derived operations such as exponentiation and their properties. But it will not commit to a concrete definition and will not prove the properties.

We propose building a community-curated library of interface theories for the typical domains of computer science and mathematics. We project that mathematicians will be eager to join this effort. Indeed, the creation of interface theories is an essential part of mathematics, and we can already see this effect in our experiences with the glossary.

Moreover, in many cases, interface theories can be extracted automatically from existing formalizations. This extraction will, however, be difficult for libraries that use complex interpretation algorithms to handle heterogeneity implicitly. It remains open how to trade-off between reusability and expressivity in these cases.

In any case, it is crucial that the interfaces are built heterogeneously to ensure that each commitment that substantially limits the possible realizations is made in a separate theory. For example, division must first be specified in an interface that can still be realized in intuitionistic foundations before refining it to a binary function in a way that tacitly assumes classical logic.

We can leverage interface logics and theories for the integration of QED in two ways. Firstly, deduction and symbolic computation procedures can refer to them to declare their scope. Accordingly, proof assistants can use them to describe the scope of open proof obligations. It is then straightforward to build mediator systems that match up problem producers with problem solvers.

Secondly, interfaces can provide a way to translate between libraries. Consider the diagram on the right where an interface theory T is implemented by two foundations F_1 and F_2. The correctness of these implementations is witnessed by two theory morphisms $r_1 : T \to F_i$.

$$\begin{tikzcd}
 & \boxed{\begin{array}{l} T \\ c, \dots \end{array}} \arrow[dl, "r_1"'] \arrow[dr, "r_2"] & \\
\boxed{F_1} \arrow[rr, dotted, leftrightarrow, "r_1(c) \approx r_2(c)"'] & & \boxed{F_2}
\end{tikzcd}$$

In particular, for a T-symbol c, we say that $r_1(c)$ and $r_2(c)$ are aligned via (c, r_1, r_2). Typically, r_i maps symbols to symbols so that it induces a partial inverse r_i^{-1}. As we already suggested in [RKS11], we can then compose r_1^{-1} with r_2 to translate partially from F_1 to F_2. This partial translation will only be defined for concepts that are expressible in the interface theory and thus in particular exclude proofs.

A major difference in the analogy between software specification and formalization is that the former only has to show that an implementation is *correct*. In the latter, we also have to ask whether it is *conservative*. We define r_i to be **conservative** if for any T-formula A, the language F_i proves $r_1(A)$ only if T proves A. Trusted library translations can now be obtained as follows: If r_1 is conservative, then every F_1 proof of $r_1(A)$ guarantees the truth of $r_2(A)$ in F_2.

However, even if we assume the consistency of F_1, it is still a major theoretical challenge to prove the conservativity of r_1. This challenge has received surprisingly little attention in the QED community so far because each individual QED system with foundation F implicitly assumes that the reference definition for the set of T-provable formulas is provided by F itself (e.g., the calculus of inductive constructions) or an informal foundation into which F can be embedded (e.g., set theory if F is higher-order logic).

6. CONCLUSION & OUTLOOK

To assess the feasibility of actually starting work on the QED project, 20 years after its announcement, we have surveyed the advances in formalized libraries. We find that while there have been massive improvements in individual deduction systems and their respective libraries, they remain insular and non-interoperable, leading to duplication of work, sub-standard re-formalizations, and missed opportunities for sharing. Moreover, the current crop of deduction systems have not been designed for interoperability or global-scale libraries.

We contend that in the current situation, where we see at least half-a-dozen major deduction systems and libraries[2], we will only reach critical mass and make progress towards an QED-inspired library if we take Peter Andrews' suggestion to "accept diversity" seriously and design a powerful, joint, pluralistic library system that federates the various libraries, provides mathematicians with a unified view on the library, and provides comprehensive library management functions based on that view.

We have presented a body of work conducted with the aim of starting such a pluralistic, global-scale library for mathematical knowledge. This work has been conducted independently from the more established automated deduction and formal methods community under the headings of "logical frameworks" (for logical plurality) and "mathematical knowledge management" (for global scale). We view this work as missing parts needed for establishing a QED library that lives up to the community's ambitions.

It will still require a large community effort to integrate the existing and future libraries into our proposed infrastructure. In several areas, especially logical frameworks and library integration, further research is necessary to improve on the existing methods and to further automate the work flows.

We contend that 20 years after the QED manifesto the time is ripe for putting together the advances in deduction systems, logical frameworks, and knowledge management to create a QED library system and organize a community effort to

[2]In [Wie07] Wiedijk compares surface and system languages of deduction system and finds that there is no inherent "winner", and we do not expect this to change in the near future.

work towards realizing this vision. Now, as then, the greater automated reasoning community has much to gain from such an effort. Especially, when some of its spin-offs are becoming standard tools in industry.

As a final word of caution relativizing what was said here, we point out that the QED effort will hardly be able to keep up with the $\sim 1.2 \cdot 10^5$ articles published annually in research mathematics alone. Even if we assume that most of these articles are not important enough to be formalized, we foresee that we will have to generalize the methods presented in this paper to include partial formalizations. Fortunately, our research shows that (contrary to complete mechanical proof verification) many aspects of the infrastructure and knowledge management can be generalized well to the partially formal setting. We conjecture that this will allow small-step formalizations that will be more efficient and flexible than the current big-step formalization process.

References

[AB+10] Ron Ausbrooks, Stephen Buswell, et al. *Mathematical Markup Language (MathML) Version 3.0.* W3C Recommendation. World Wide Web Consortium (W3C), 2010. URL: `http://www.w3.org/TR/MathML3`.

[ABMU11] J. Alama, K. Brink, L. Mamane, and J. Urban. "Large Formal Wikis: Issues and Solutions". In: *Intelligent Computer Mathematics.* Ed. by J. Davenport, W. Farmer, J. Urban, and F. Rabe. Springer, 2011, pp. 133–148.

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

[AFP] AFP. *Archive of Formal Proofs.* URL: `http://afp.sf.net` (visited on 12/20/2011).

[And94] Peter B. Andrews. *Accept Diversity.* 1994. URL: `http://mizar.org/qed/mail-archive/volume-2/0199.html`.

[Ano94] Anonymous. "The QED Manifesto". In: *Automated Deduction.* Ed. by A. Bundy. Springer, 1994, pp. 238–251.

[Art] R. Arthan. *ProofPower.* `http://www.lemma-one.com/ProofPower/`.

[BC+04] Stephen Buswell, Olga Caprotti, et al. *The Open Math Standard, Version 2.0.* Tech. rep. The OpenMath Society, 2004. URL: `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: *Proceedings of PxTP2012: Proof Exchange for Theorem Proving.* Ed. by D. Pichardie and T. Weber. 2012, pp. 28–43.

[BDKK12] T. Bourke, M. Daum, G. Klein, and R. Kolanski. "Challenges and Experiences in Managing Large-Scale Proofs". In: *Intelligent Computer Mathematics.* Ed. by J. Jeuring, J. Campbell, et al. Springer, 2012, pp. 32–48.

[BK07] Grzegorz Bancerek and Michael Kohlhase. "Towards a Mizar Mathematical Library in OMDoc Format". In: *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*. Ed. by R. Matuszewski and A. Zalewska. Vol. 10:23. Studies in Logic, Grammar and Rhetoric. University of Białystok, 2007, pp. 265–275. URL: `http://kwarc.info/kohlhase/papers/trybook.pdf`.

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

[CA+86] R. Constable, S. Allen, et al. *Implementing Mathematics with the Nuprl Development System.* Prentice-Hall, 1986.

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

[CH+11] M. Codescu, F. Horozal, et al. "Project Abstract: Logic Atlas and Integrator (LATIN)". In: *Intelligent Computer Mathematics.* Ed. by J. Davenport, W. Farmer, F. Rabe, and J. Urban. Springer, 2011, pp. 289–291.

[CH88] T. Coquand and G. Huet. "The Calculus of Constructions". In: *Information and Computation* 76.2/3 (1988), pp. 95–120.

[Chu40] A. Church. "A Formulation of the Simple Theory of Types". In: *Journal of Symbolic Logic* 5.1 (1940), pp. 56–68.

[DW97] Ingo Dahn and Christoph Wernhard. "First Order Proof Problems Extracted from an Article in the Mizar Mathematical Library". In: *Proceedings of the International Workshop on First order Theorem Proving.* Ed. by Ulrich Furbach and Maria Paola Bonacina. RISC-Linz Report Series 97-50. Johannes Kepler Universität Linz, 1997, pp. 58–62.

[FGT] William M. Farmer, Joshua Guttman, and Javier Thayer. *The IMPS online theory library.* URL: `http://imps.mcmaster.ca/theories/theory-library.html` (visited on 12/17/2014).

[FGT92] W. Farmer, J. Guttman, and F. Thayer. "Little Theories". In: *Conference on Automated Deduction.* Ed. by D. Kapur. 1992, pp. 467–581.

[FGT93] W. Farmer, J. Guttman, and F. Thayer. "IMPS: An Interactive Mathematical Proof System". In: *Journal of Automated Reasoning* 11.2 (1993), pp. 213–248.

[GA+13] G. Gonthier, A. Asperti, et al. "A Machine-Checked Proof of the Odd Order Theorem". In: *Interactive Theorem Proving.* Ed. by S. Blazy, C. Paulin-Mohring, and D. Pichardie. 2013, pp. 163–179.

[GK14] T. Gauthier and C. Kaliszyk. "Matching concepts across HOL libraries". In: *Intelligent Computer Mathematics.* Ed. by S. Watt, J. Davenport, et al. Springer, 2014, pp. 267–281.

[GL] *GitLab.* URL: `http://gitlab.org` (visited on 02/24/2014).

[GLR09] J. Gičeva, C. Lange, and F. Rabe. “Integrating Web Services into Active Mathematical Documents”. In: *Intelligent Computer Mathematics*. Ed. by J. Carette, L. Dixon, C. Sacerdoti Coen, and S. Watt. Springer, 2009, pp. 279–293.

[HA+15] Thomas Hales, Mark Adams, et al. *A formal proof of the Kepler conjecture*. 2015. URL: 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*. Springer, 1996, pp. 265–269.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. “A framework for defining logics”. In: *Journal of the Association for Computing Machinery* 40.1 (1993), pp. 143–184.

[HI+11] F. Horozal, A. Iacob, et al. “Combining Source, Content, Presentation, Narration, and Relational Representation”. In: *Intelligent Computer Mathematics*. Ed. by J. Davenport, W. Farmer, F. Rabe, and J. Urban. Springer, 2011, pp. 212–227.

[HKR12] F. Horozal, M. Kohlhase, and F. Rabe. “Extending MKM Formats at the Statement Level”. In: *Intelligent Computer Mathematics*. Ed. by J. Campbell, J. Carette, et al. Springer, 2012, pp. 64–79.

[HOL4] The HOL4 development team. *HOL4*. URL: http://hol.sourceforge.net/ (visited on 12/17/2014).

[Hor14] F. Horozal. “A Framework for Defining Declarative Languages”. PhD thesis. Jacobs University Bremen, 2014.

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

[HR09] Peter Horn and Dan Roozemond. “OpenMath in SCIEnce: SCSCP and POPCORN”. In: *MKM/Calculemus Proceedings*. Ed. by Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt. LNAI 5625. Springer Verlag, July 2009, pp. 474–479. ISBN: 978-3-642-02613-3.

[HR11] F. Horozal and F. Rabe. “Representing Model Theory in a Type-Theoretical Logical Framework”. In: *Theoretical Computer Science* 412.37 (2011), pp. 4919–4945.

[HR15] F. Horozal and F. Rabe. “Formal Logic Definitions for Interchange Languages”. In: *Intelligent Computer Mathematics*. Ed. by M. Kerber, J. Carette, et al. Springer, 2015, pp. 171–186.

[HRK14] F. Horozal, F. Rabe, and M. Kohlhase. “Flexary Operators for Formalized Mathematics”. In: *Intelligent Computer Mathematics*. Ed. by S. Watt, J. Davenport, et al. Springer, 2014, pp. 312–327.

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

[IK15] Mihnea Iancu and Michael Kohlhase. "Math Literate Knowledge Management via Induced Material". In: *Intelligent Computer Mathematics 2015*. LNCS. submitted. Springer, 2015, pp. 187–202. URL: http://kwarc.info/kohlhase/papers/cicm15-induced.pdf.

[IKRU13] M. Iancu, M. Kohlhase, F. Rabe, and J. Urban. "The Mizar Mathematical Library in OMDoc: Translation and Applications". In: *Journal of Automated Reasoning* 50.2 (2013), pp. 191–202.

[IR11] M. Iancu and F. Rabe. "Formalizing Foundations of Mathematics". In: *Mathematical Structures in Computer Science* 21.4 (2011), pp. 883–911.

[IR12] M. Iancu and F. Rabe. "Management of Change in Declarative Languages". In: *Intelligent Computer Mathematics*. Ed. by J. Campbell, J. Carette, et al. Springer, 2012, pp. 325–340.

[JC+12] Johan Jeuring, John A. Campbell, et al., eds. *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics (CICM) (Bremen, Germany, July 9–14, 2012). LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012. ISBN: 978-3-642-31373-8.

[KA+10] G. Klein, J. Andronick, et al. "seL4: formal verification of an operating-system kernel". In: *Communications of the ACM* 53.6 (2010), pp. 107–115.

[KI12] Michael Kohlhase and Mihnea Iancu. "Searching the Space of Mathematical Knowledge". In: *DML and MIR 2012*. Ed. by Petr Sojka and Michael Kohlhase. in press. Masaryk University, Brno, 2012. ISBN: 978-80-210-5542-1. URL: http://kwarc.info/kohlhase/papers/mir12.pdf.

[KK13] C. Kaliszyk and A. Krauss. "Scalable LCF-style proof translation". In: *Interactive Theorem Proving*. Ed. by S. Blazy, C. Paulin-Mohring, and D. Pichardie. Springer, 2013, pp. 51–66.

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

[KMP12] Michael Kohlhase, Bogdan A. Matican, and Corneliu C. Prodescu. "MathWebSearch 0.5 – Scaling an Open Formula Search Engine". In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics (CICM) (Bremen, Germany, July 9–14, 2012). Ed. by Johan Jeuring, John A. Campbell, et al. LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012, pp. 342–357. ISBN: 978-3-642-31373-8. URL: http://kwarc.info/kohlhase/papers/aisc12-mws.pdf.

[Koh06] Michael Kohlhase. OMDOC – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[Koh08] M. Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304.

[Koh12] Michael Kohlhase. “The Planetary Project: Towards eMath3.0”. In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics (CICM) (Bremen, Germany, July 9–14, 2012). Ed. by Johan Jeuring, John A. Campbell, et al. LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012, pp. 448–452. ISBN: 978-3-642-31373-8. arXiv: `1206.5048 [cs.DL]`.

[Koh14] Michael Kohlhase. “A Data Model and Encoding for a Semantic, Multilingual Terminology of Mathematics”. In: *Intelligent Computer Mathematics 2014*. Conferences on Intelligent Computer Mathematics (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt, James Davenport, et al. LNCS 8543. Springer, 2014, pp. 169–183. ISBN: 978-3-319-08433-6. URL: `http://kwarc.info/kohlhase/papers/cicm14-smglom.pdf`.

[KR14] C. Kaliszyk and F. Rabe. “Towards Knowledge Management for HOL Light”. In: *Intelligent Computer Mathematics*. Ed. by S. Watt, J. Davenport, et al. Springer, 2014, pp. 357–372.

[KS10] A. Krauss and A. Schropp. “A Mechanized Translation from Higher-Order Logic to Set Theory”. In: *Interactive Theorem Proving*. Ed. by M. Kaufmann and L. Paulson. Springer, 2010, pp. 323–338.

[KU13] C. Kaliszyk and J. Urban. “Automated Reasoning Service for HOL Light”. In: *Intelligent Computer Mathematics*. to appear. Springer, 2013.

[KW10] C. Keller and B. Werner. “Importing HOL Light into Coq”. In: *Interactive Theorem Proving*. Ed. by M. Kaufmann and L. Paulson. Springer, 2010, pp. 307–322.

[KŞ06] Michael Kohlhase and Ioan Şucan. “A Search Engine for Mathematical Formulae”. In: *Proceedings of Artificial Intelligence and Symbolic Computation, AISC’2006*. Ed. by Tetsuo Ida, Jacques Calmet, and Dongming Wang. LNAI 4120. Springer Verlag, 2006, pp. 241–253. URL: `http://kwarc.info/kohlhase/papers/aisc06.pdf`.

[MB+03] Erica Melis, Jochen Büdenbender, et al. “Knowledge Representation and Management in ActiveMath”. In: *Annals of Mathematics and Artificial Intelligence* 38.1–3 (2003), pp. 47–64.

[MC] *Mathematical Components*. URL: `http://www.msr-inria.fr/projects/mathematical-components-2/` (visited on 12/17/2014).

[MeMa] *Metamath Home page*. URL: `http://us.metamath.org`.

[MizLib] *Mizar Mathematical Library*. URL: `http://www.mizar.org/library` (visited on 09/27/2012).

[ML74] P. Martin-Löf. “An Intuitionistic Theory of Types: Predicative Part”. In: *Proceedings of the ’73 Logic Colloquium*. North-Holland, 1974, pp. 73–118.

[MTHM97] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML,* Revised edition. MIT Press, 1997.

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

[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: *11th International Conference on Automated Deduction (CADE)*. Ed. by D. Kapur. Springer, 1992, pp. 748–752.

[OS06] S. Obua and S. Skalberg. "Importing HOL into Isabelle/HOL". In: *Automated Reasoning*. Ed. by N. Shankar and U. Furbach. Vol. 4130. Springer, 2006.

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

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

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

[Pfe01] F. Pfenning. "Logical frameworks". In: *Handbook of automated reasoning*. Ed. by J. Robinson and A. Voronkov. Elsevier, 2001, pp. 1063–1147.

[PS99] F. Pfenning and C. Schürmann. "System Description: Twelf - A Meta-Logical Framework for Deductive Systems". In: *Automated Deduction*. Ed. by H. Ganzinger. 1999, pp. 202–206.

[PVS] *NASA PVS Library*. URL: `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/` (visited on 12/17/2014).

[Rab12] F. Rabe. "A Query Language for Formal Mathematical Libraries". In: *Intelligent Computer Mathematics*. Ed. by J. Campbell, J. Carette, et al. Springer, 2012, pp. 142–157.

[Rab13a] F. Rabe. "A Logical Framework Combining Model and Proof Theory". In: *Mathematical Structures in Computer Science* 23.5 (2013), pp. 945–1001.

[Rab13b] F. Rabe. "The MMT API: A Generic MKM System". In: *Intelligent Computer Mathematics*. Ed. by J. Carette, D. Aspinall, et al. Springer, 2013, pp. 339–343.

[Rab14a] F. Rabe. "A Logic-Independent IDE". In: *Workshop on User Interfaces for Theorem Provers*. Ed. by C. Benzmller and B. Woltzenlogel Paleo. Elsevier, 2014, pp. 48–60.

[Rab14b] F. Rabe. "How to Identify, Translate, and Combine Logics?" In: *Journal of Logic and Computation* (2014). doi:10.1093/logcom/exu079.

[Rab15a] F. Rabe. "Generic Literals". In: *Intelligent Computer Mathematics*. Ed. by M. Kerber, J. Carette, et al. Springer, 2015, pp. 102–117.

[Rab15b] F. Rabe. "Lax Theory Morphisms". In: *ACM Transactions on Computational Logic* (2015). accepted pending minor revisions; see `http://kwarc.info/frabe/Research/rabe_lax_14.pdf`.

[RK13] F. Rabe and M. Kohlhase. "A Scalable Module System". In: *Information and Computation* 230.1 (2013), pp. 1–54.

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

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

[Sko67] Thoralf Skolem. "The foundations of elementary arithmetic established by means of the recursive mode of thought, without the use of apparent vari- ables ranging over infinite domains". In: 3rd printing, 1997. Source books in the history of the sciences series. Cambridge, MA: Harvard Univ. Press, 1967, pp. 302–333. ISBN: 0-674-32450-1.

[Slo03] Neil J. A. Sloane. "The On-Line Encyclopedia of Integer Sequences". In: *Notices of the AMS* 50.8 (2003), p. 912.

[Sut09] G. Sutcliffe. "The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0". In: *Journal of Automated Reasoning* 43.4 (2009), pp. 337–362.

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

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

[Tea] Coq Development Team. *The Coq Proof Assistant: Reference Manual*. INRIA. URL: https://coq.inria.fr/refman/.

[Urb06] Josef Urban. "XML-izing Mizar: making semantic processing and presentation of MML easy". In: *Mathematical Knowledge Management, MKM'05*. Ed. by Michael Kohlhase. LNAI 3863. Springer Verlag, 2006, pp. 346 –360.

[Wen10] M. Wenzel. "Asynchronous Proof Processing with Isabelle/Scala and Isabelle/jEdit". In: *User Interfaces for Theorem Provers*. Ed. by D. Aspinall and C. Sacerdoti Coen. ENTCS, 2010, pp. 101–114.

[Wen12] M. Wenzel. "Isabelle/jEdit - A Prover IDE within the PIDE Framework". In: *Intelligent Computer Mathematics*. Ed. by Johan Jeuring, John A. Campbell, et al. Springer, 2012, pp. 468–471.

[Wie07] F. Wiedijk. "The QED Manifesto Revisited". In: *From Insight to Proof, Festschrift in Honour of Andrzej Trybulec*. 2007, pp. 121–133.

[Win14] W. Windsteiger. "Theorema 2.0: A System for Mathematical Theory Exploration". In: *International Congress on Mathematical Software*. Ed. by H. Hong and C. Yap. Springer, 2014, pp. 49–52.

[WR13] A. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 1913.