JACOBS UNIVERSITY

Knowledge Management across Formal Libraries

by

Dennis Müller

PhD Research Proposal

Prof. Dr. Michael Kohlhase

Jacobs University, Germany

Prof. Dr. Herbert Jäger

Jacobs University, Germany

Prof. Dr. Claudio Sacerdoti Coen

University of Bologna, Italy

Dr. Florian Rabe

Jacobs University, Germany

Dr. Natarajan Shankar

SRI International, USA

Date of submission: November 18, 2015

School of Engineering and Science

Abstract

In my PhD, I want to contribute to developing an open archive of formalizations, a common and open infrastructure for managing and sharing formalized mathematical knowledge represented as theories, definitions, and proofs, based on a uniform foundation-independent representation format for libraries, integrate existing formal libraries based on arbitrary logical systems into this archive and develop methods to efficiently transfer and share information between different libraries and logical foundations.

Contents

1 Introduction and Motivation

In the last decades, the formalization of mathematical knowledge (and the verification and automation of formal proofs) has become of ever increasing interest. Formal methods nowadays are not just used by computer scientists to verify software and hardware as well as in program synthesis, but – due to problems such as Kepler's conjecture [1], the classification theorem for finite simple groups [2], etc. – are also becoming increasingly popular among mathematicians in general.

By now, there is a vast plurality of formal systems and corresponding libraries to choose from. However, almost all of these are non-interoperable, because they are based on differing, mutually incompatible logical foundations (e.g. set theories, higher-order logic, variants of type theory etc.), library formats, library structures, and much work is spent developing the respective basic libraries in each system.

Moreover, since a library in one such system is not reusable in another system, developers are forced to spend an enormous amount of time and energy developing basic library organization features such as distribution, browsing/search or change management for each library format; all of which binds resources that could be used to improve core functionality and library contents instead.

One reason for the incompatibility is the widespread usage of the *homogeneous method*[1], which fixes some logical foundation with all primitive notions (e.g. types, axioms, inference rules) and uses conservative extensions of this foundation to allow for modeling some specific domain knowledge. While homogeneous reasoning is conveniently implementable and verifiable, it implies that a lot of work is needed just to model the basic domains of discourse necessary for mathematics, such as real numbers, algebraic theories, etc. Moreover, the resulting formalizations are actually less valuable to mathematicians, since they are intimitely dependent on (and framed in terms of) the underlying logic, making it virtually impossible to move the results between different foundations or to abstract from them.

In contrast, mathematical practice favors the *heterogeneous method*, as exemplified by the works of Bourbaki [3]. In heterogeneous reasoning, theories are used to introduce new primitive notions and the truth of some proposition is relative to a theory (as opposed to absolute with respect to some foundation). This method is closely related to the "little theories" paradigm [4] which prefers to state each theorem in the weakest possible theory, thus optimizing reusability of mathematical results. Even though in theory, all of mathematics can be reduced to first principles – most prominently first-order logic with (some extension of) ZFC – it is usually carried out in a highly abstract setting that hides the foundation, often rendering it irrelevant and thus allowing it to be ignored.

Correspondingly, there is an inconvenient discrepancy between the currently existing formal libraries and the way (informal) mathematics is usually done in practice. Furthermore, the available formal knowledge is distributed among dozens of different mutually incompatible formal library systems with considerable overlap between them.

What we envision instead is a *universal archiving solution for all formal knowledge*, that

1. does not depend on a specific logical foundation,
2. can import libraries from different formal systems, allowing one to choose which system to use for generating new content,
3. allows for abstracting from and transferring results between differing foundations and library structures and
4. is capable of identifying equivalent/identical results in different libraries.

In my PhD, I want to contribute to realizing such a universal library system.

2 State of the Art

The attempt to systematically formalize both mathematical knowledge and its semantics goes back at least to the seminal work by Bertrand Russell and Albert North Whitehead [5]. In the 1950s and

[1] The terminology *homogeneous/heterogeneous* is due to Florian Rabe, unpublished.

1960s, computer systems were added to the tool chest for this endeavour, shifting the focus to designing foundations that combine both machine-friendliness and human readability. This enabled *automated theorem proving*, thanks to ideas going back to Allen Newell, Herbert A. Simon, and Martin Davis. This has been most succesful for first-order logic and related systems. For more expressive languages, the *verification* of human-written formal proofs has been the more successful approach, going back to John McCarthy, Nicolaas Govert de Bruijn, Robin Milner, and Per Martin-Löf. Modern proof assistants usually combine both approaches, generally verifying user input interactively and employing automation for routine proof steps whenever possible.

Since developing sophisticated proof systems requires both a lot of practical work and a high level of theoretical knowledge, most popular proof assistants have invested a large amount of time and energy into building their systems. Consequently, formalization within one such system pays off mostly at large scales, calling for a community effort to extend the corresponding formal libraries. In practice, however, the availability of many different and mutually incompatible systems – each with their own advantages and disadvantages – has instead driven towards ever increasing specialization within the formal mathematics community, resulting in lots of different libraries with considerable (and from a heterogeneous point of view unnecessary) overlap of actual content.

A more extensive summary of the state of the art can be found in [6] by the supervisors of this proposal.

2.1 Foundations

Overview The notion of a *foundation* goes back to the *foundational crisis of mathematics* at the beginning of the 20th century. Resulting from a general confusion regarding the ontological status of informally defined objects like infinitesimals in real analysis, non-euclidean geometries and sets as objects of study in their own right, as well as debates over the validity of certain non-constructive proof techniques of ever growing abstraction and intricacy, the idea developed to find a formal, logical basis, that fixes both an ontology as well as an unambiguous notion of what constitutes a valid proof.

By now, it is generally accepted within the mathematical community, that the answer to this problem is the combination of first-order logic and (some system of) set theory. However, already going back to *Principia Mathematica* by Russell and Whitehead (which can be seen as an early variant of type theory), alternative foundations have been around.

There's an inherent trade-off in every such foundation with respect to their complexity and expressiveness. Theoretically, a foundation should be simple with very few primitive notions, to make reasoning *about* the foundation more convenient and establish trust in its consistency. On the other hand, since a foundation should ultimately establish a framework for all of mathematics, it should be as expressive as possible to allow mathematicians to talk about all of the desired objects and realms in terms of the foundation. This trade-off naturally lead to a large diversity of different foundations, which nowadays are used in different proof assistants.

All of these systems fix one specific foundation as a basis for their specification language, usually variants of either constructive type theory, higher-order logic or (as implicitely used in mathematics) first-order set theory. The constructive type theories are mostly based on Martin-Löf type theory [7] or the calculus of constructions [8] and make use of the Curry-Howard correspondence [9, 10] to treat propositions as types (and proofs as λ-terms). Systems include Nuprl [11], Agda [12], Coq [13], and Matita [14]. The second group of systems go back to Church's higher-order logic [15] and include HOL4 [16], ProofPower [17], Isabelle/HOL [18], and HOL Light [19].

Since type theories and higher-order logics are more conveniently machine implementable, systems using set theories are noticably rarer. These are e.g. Mizar [20], Isabelle/ZF [21], and Metamath [22].

The foundation of the PVS system (see Section 4.4.1) includes a variant of higher-order logics, but has been specifically designed to have a set theoretic semantics. The IMPS system [23] is based on a variant of higher-order logic with partial functions. The foundation of ACL2 [24] is an untyped language based on Lisp.

Heterogeneous Reasoning As mentioned in the introduction, even though all of mathematics is assumed to be reducible to some foundation, the way mathematics is usually practiced is according to

the **heterogeneous method**, in which all foundational aspects are "hidden" and left implicit, unless necessary in the respective context. One major advantage of this method, in which theories are used to introduce new primitive notions, is that (in connection with the *little theories* approach) it allows for reusing mathematical results and moving them along *theory morphisms* (i.e. truth-preserving maps) between theories. 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 [25] and ASL [26].

To accomodate for this more convenient style of reasoning, most formal systems have specific features that introduce some form of heterogeneity either *explicitely* or *implicitely*. Explicit features include e.g. "locales" in Isabelle, "parametric theories" in PVS, "modules" in Coq, or "structures" in Mizar. The IMPS system [23] is somewhat unique in so far, as it was designed specifically with heterogeneous reasoning in mind.

To use the heterogeneous method implicitely, the user introduces a new formal construct (such as real numbers) by defining them in terms of existing ones (e.g. the usual construction via equivalence classes on Cauchy sequences) and proving the actually relevant (i.e. construction independent) theorems about them (e.g. field axioms, topological completeness). Finally, the user can rely on the proven properties alone, without ever needing to expand the concrete definition originally implemented, which in the process is rendered irrelevant for all practical purposes.

To enable implicit heterogeneity, the system has to provide corresponding definition principles. Examples include type definitions in the HOL systems, provably terminating functions in Coq or Isabelle/HOL, or provably well-defined indirect definitions in Mizar. However, it should be noted, that such implicit heterogeneous definitions are usually internally expanded into conservative extensions of the foundation. Alternatively, several data types provide additional way to use heterogeneity implicitely. These include (Co)inductive types and record types e.g, as done by Mizar structures and with Coq records in the Mathematical Components project [27]. This has the additional advantage, that it allows for computation within the foundation.

The Incompatibility Problem The homogeneous method (in combination with implicit heterogeneity) has an obvious advantage: Since the foundation is fixed and can not be extended by new axioms, the trusted code base remains fixed as well. Furthermore, is allows integrating computational methods e.g. to reason about equality, without having to tediously instantiate and apply the corresponding axioms. However, it has the major disadvantage of making reuse of formalized knowledge difficult, if not impossible. Since the techniques for implicit heterogeneity are elaborated on the basis of the foundation, the actual heterogeneous structure of some fragment of implemented knowledge is difficult to identify. Furthermore, since different systems offer different techniques to introduce implicit heterogeneity, the construction method used in one system can often not be easily translated into another system. Their elaborated implementations themselves are again framed in terms of the foundation (or an intrinsic part thereof), and are consequently useless for a system based on a differing foundation.

Even though a fixed foundation is therefore reasonable for an individual formal system, if we envision a *universal library of mathematics* – a major goal of formal mathematics going back at least to the QED project and manifesto [28] of 1994 – this is the wrong approach. Furthermore, different areas of mathematics favor different foundations (such as category theory, specifically in algebra), as do different communities dealing with formalizing mathematics, whether due to technical reasons or simply familiarity. Thus, formal libraries would profit from **foundational pluralism**, i.e., the ability to support multiple foundations in a single universal library.

2.2 Formal Libraries

Overview Usually, formal systems have some notion of a *library*, a collection of formalizations (usually just the corresponding source files) that can be handled in specific ways – imported and used when creating new content, collectively exported, etc. Most systems are distributed with some base library providing the most useful and ubiquitous settings of interest, such as Booleans or number spaces, and often maintain additional, community-created libraries with more advanced contents.

The Isabelle and the Mizar groups maintain one centralized library each – the "Archive of Formal Proof" [29] and the "Mizar Mathematical Library" [30], respectively. The Coq group maintains a similar set of contributions. These libraries contain individual formalizations with relatively few interdependencies. Other libraries are generated and maintained by communities apart from the developers of the system, however, these are still often valuable in their own right, such as Tom Hales's formalizations in HOL Light for the Kepler conjecture [1] and Georges Gonthier's work in Coq for the recently proved Feit-Thompson theorem [31]. John Harrison's formalizations in his HOL Light system [19] and the NASA PVS library [32] 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.

As mentioned, most of these systems are based on the homogeneous method. However, there are some libraries that are intrinsically heterogeneous, such as the IMPS library [33], the LATIN logic library [34] developed at KWARC (see Section 2.4.2) and the TPTP library [35] of challenge problems for automated theorem provers. Unfortunately, none of these enjoys the level of interpretation, deduction, and computation support developed for individual fixed foundations.

The OpenTheory format [36] 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 two problems concerning library integration. The first is, given a single library, **refactoring** its contents to increase modularity. This results in a "more heterogeneous" set of theories, thus making it easier to reuse and blowing up the corresponding theory graph to make shared or inherited theories and results more visible and explicit. An attempt at giving a formal calculus for theory refactoring has recently been published [37].

The second problem is, given two or more libraries, to **integrate** them *with each other*, so that knowledge formalized in one of them can be reused in, translated to or identified with contents of the others. In the best case, two libraries might even be **merged** into a single library with ideally no redundant content.

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). Also, the widespread use of the homogeneous method makes integrating and merging libraries in different systems extremely difficult, since usually basic concepts in one foundation cannot be directly translated to corresponding concepts in the other [38].

This is despite the large need for more integrated and easily reusable large libraries. For example, in Tom Hales's Flyspeck project [1], 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 Translations There are two ways to take on library integration. Firstly, one can try to translate the contents of one formal system directly into another system; I will call his a *library bridge*. This requires intimate knowledge of both systems and their respective foundations used, and is necessarily somewhat ad-hoc. A small number of library bridges have been realized, typically in special situations. [39] translates from HOL Light [19] to Coq [13] and [40] to Isabelle/HOL. Both translations benefit from the well-developed HOL Light export and the simplicity of the HOL Light foundation. [41] translates from Isabelle/HOL [18] to Isabelle/ZF [21]. 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 [36] facilitates sharing between HOL-based systems but has not been used extensively.

The second way is to use a more general *logical framework* (see Section 2.3) which provides some way to specify the respective foundations, and integrate the libraries under consideration directly into that framework. Then the framework can serve as a uniform intermediate data structure, via which other

systems can import the integrated libraries. This approach has been used in [42] for Mizar and in [43] for HOL Light, using the logical framework LF [44] and making the libraries available to knowledge management services. Another example is the Dedukti system [45], which imports, e.g., Coq and HOL Light into a similar logical framework, namely LF extended with rewriting.

Again, the prevalence of the homogeneous method constitutes a major problem here. Even with implicit heterogeneity, the fact that theories such as the real numbers are still modeled as conservative extensions of a fixed foundation using some intricate construction principle (cauchy sequences, Dedekind cuts) means, that using different definitions can make it impossible to align a theory in one library to the corresponding theory in a different library, even though from a mathematical point of view they are "the same" (i.e. isomorphic). And even if the same abstract construction principle is used in two libraries, their implementations in terms of the underlying foundation can be different enough to make it difficult to identify the two resulting theories.

Very little work exists to address this problem. In [40], 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 [46], 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 [47] by using machine learning to identify large sets of further alignments.

The OpenTheory format [36] 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.

A partial solution aimed at overcoming the integration problem was sketched in [48].

2.3 Logical Frameworks

Recently, formal systems have added an additional metal level through the introduction of logical frameworks. These provide tools to specify logical systems themselves in a formal way. An overview of the current state of art is given by [49].

An example for one such framework is LF [44], which is based on the dependently-typed λ-calculus and uses the judgments-as-types paradigm. This means, that, when specifying some logic, one usually introduces a new type constructor $\vdash$, which maps a proposition φ of that logic to the corresponding *type of proofs of* φ. An element of type $\vdash \varphi$ is therefore a proof of φ and declaring such an element corresponds to the judgement "φ holds".

Logical frameworks introduce the possibility to additionally reason *about* logics, as e.g. in Twelf [50], which is the currently most mature implementation of LF. Twelf has been used as a basis for the LATIN library (see Section 2.4.2). Also, since in logical frameworks logics are themselves represented as theories, they allow for defining logical systems heterogeneously, by building them up in a modular way.

Dedukti [45] 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 [51] implements intuitionistic higher-order logic, which (if seen as a pure type system with propositions-as-types) is rather similar to LF. 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.

Unfortunately, logical frameworks are not an efficient alternative to the prevailing homogeneous formal systems. State of the art proof assistents rely heavily on the specific peculiarities of their underlying foundation to provide efficient proof search techniques, which would be impossible to implement at the high level of generality that logical frameworks provide, at least without introducing cosiderable overhead and thus reducing efficiency.

Another problem is that specific concepts used by some logic (noticably record types, subtyping principles, inductive definitions, etc.) may be difficult to realize in a logical framework in such a way,

that type checking can be done effectively, since the necessary information for doing so can not always easily be lifted to the rather general level on which the type checker operates (see Section 4.2).

2.4 Projects at the KWARC research group

There has been a series of research projects at the KWARC research group that clearly "converges" towards solving some of the problems described above, starting with MMT/OMDOC and continuing via LATIN to the OAF project. My research will be part of (and is funded by a DFG grant for) the latter.

2.4.1 MMT/OMDoc

OMDOC [52] is a representation language developed by Michael Kohlhase, that extends OpenMath and MathML and allows for providing general definitions of the syntax of both mathematical objects as well as mathematical documents. They make use of *content dictionaries*, which introduce primitive notions and their semantics. In particular, it can represent the exports of deduction system libraries and their documentation.

In the last five years, (chiefly) Florian Rabe re-developed the fragment of OMDOC pertaining to formal knowledge resulting in the MMT language [53, 54, 55]. MMT greatly extends the expressivity, clarifies the representational primitives, and formally defines the semantics of this OMDOC fragment. It is designed to be foundation-independent and introduces several concepts to maximize modularity and to abstract from and mediate between different foundations, to reuse concepts, tools, and formalizations.

More concretely, the MMT language *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.

OMDOC/MMT offers vey few primitives, which have turned out to be sufficient for most practical settings. These are

1. *constants* with optional types and definitions,
2. types and definitions of constants are *objects*, which are syntax trees with binding, using previously defined constants as leaves,
3. *theories*, which are lists of constant declarations and
4. *theory morphisms*, that map declarations in a domain theory to expressions built up from declarations in a target theory.

Using these primitives, logical frameworks, logics and theories *within* some logic are all uniformly represented as MMT theories, rendering all of those equally accessible, reusable and extendable. Constants, functions, symbols, theorems, axioms, proof rules etc. are all represented as constant declarations, and all terms which are built up from those are represented as objects.

Theory morphisms represent truth-preserving maps between theories. Examples include theory inclusions, translations/isomorphisms between (sub)theories and models/instantiations (by mapping axioms to theorems that hold within a model), as well as a particular theory inclusion called *meta-theory*, that relates a theory on some meta level to a theory on a higher level on which it depends. This includes the relation between some low level theory (such as the theory of groups) to its underlying foundation (such as first-order logic), and the latter's relation to the logical framework used to define it (e.g. LF).

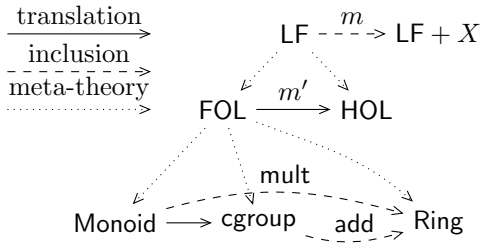

Figure 1: A Theory Graph with Meta-Theories and Theory Morphisms in MMT (from [6])

All of this naturally gives us the notion of a *theory graph*, which relates theories (represented as nodes) via vertices representing theory morphisms (as in Figure 1), being right at the design core of the MMT language.

Given this modular approach of OMDOC/MMT, heterogeneity is made explicit in the sense, that even though foundations are present via meta-theories, only those aspects of the foundation that are used in the definition of a theory T are present in the theory itself, making it easy to abstract from the foundation and reuse the theory.

The OMDOC/MMT language is used by the MMT **system** [53], which provides a powerful API to work with documents and libraries in the MMT language, including a terminal to execute MMT specific commands, a web server to display information about MMT libraries (such as their theory graphs) and a plugin for the text editor jEdit, that can be used to create, type check and compile documents in the MMT language. The API is heavily customizable via plugins to e.g. add foundation specific type checking rules and import and translate documents from different formal systems.

All of this puts MMT on a new meta level, which can be seen 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 $\mathsf{LF} + X$ without any change to the MMT infrastructure and can easily migrate all results obtained within LF.

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

2.4.2 LATIN

The LATIN project [34] was a DFG funded project running from 2009 to 2012 under the principal investigators Michael Kohlhase and Till Mossakowski. Its aim has been to build a heterogeneous, highly integrated library of formalizations of logics and related languages as well as translations between them. It uses MMT as a framework, with LF as a meta-theory for the individual logics.

True to the general MMT philosophy, all the integrated theories are built up in a modular way and 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 numbers. 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.

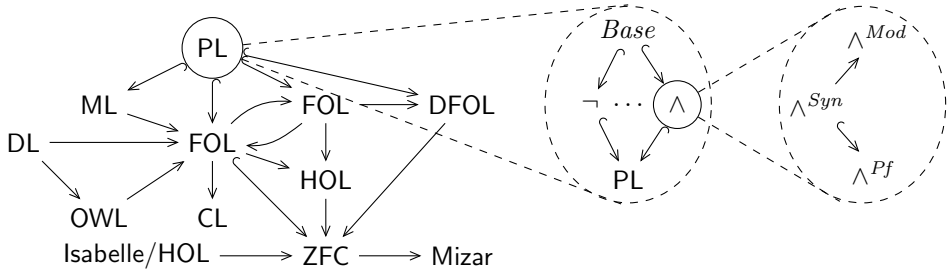

Figure 2: A Fragment of the LATIN Atlas (from [6])

The left side of Figure 2 shows a fragment of the LATIN atlas, focusing on first-order logic (FOL) being built on top of propositional logic (PL), its translation to HOL and ultimately resulting in the foundations of Mizar, Isabelle/HOL and ZFC, as well as translations between them. The formalization of propositional logic includes its syntax as well as its proof and model theory, as shown on the right of Figure 2.

3 Research Objectives

My Ph.D. will be part of the **OAF** project [56], a DFG funded project running from 2015 to 2017 under the principal investigators Michael Kohlhase and Florian Rabe. The goal is to provide an **O**pen **A**rchive of **F**ormalizations: a universal archiving solution for formal libraries, corresponding library management services (such as distributing, browsing and searching library contents) and methods for integrating libraries in different formalisms in a unifying framework – namely OMDoc/Mmt, as in Figure 3 – to allow for sharing and translating content across them. We further want it to be scalable with respect to both the size of the knowledge base and the diversity of logical foundations.

Theoretically, the main prerequisite has been established in the LATIN project. However, whereas LATIN provides a logical framework, there still remains the problem of integrating the existing formal libraries. Consequently, there are two major objectives I want to participate in:

O1: Making existing libraries accessible to a unifying framework We want to be able to make available theorem prover libraries accessible to the Mmt system. To do so, we will have to provide

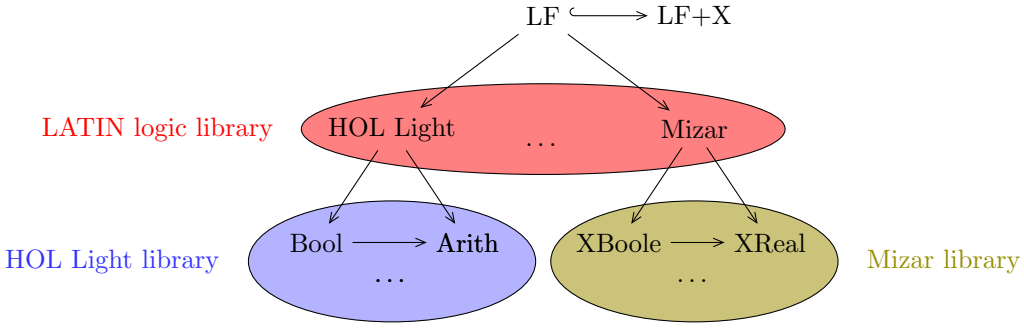

Figure 3: Representing Libraries in Mmt (from [6])

an MMT meta theory for the foundational logic underlying the formal system under consideration and write a plugin for the MMT system, which handles the importing of the library and translating its content into the OMDoc/MMT format. As mentioned in Section 2.3, to handle peculiar concepts provided by the theorem prover and allow for type checking those, this will entail extending LF, the OMDoc/MMT language and the MMT system accordingly. As a result, implementing the necessary plugins and theories for new formal systems should be as convenient as possible, to quickly build up the set of libraries available in MMT.

The HOLLight, Mizar and TPTP libraries have already been integrated this way; I will attempt to do the same with PVS (see Section 4.4.1), which is more challenging, since its foundation contains primitives which are not currently conveniently specifiable in the OMDoc/MMT language.

O2: Refactoring and integrating libraries Once we have several libraries integrated into MMT, I want to look into methods to

1. **refactor** the available libraries in such a way, that the heterogeneous part of some theory can be recognized as such and seperated from the foundational aspects of its library, and
2. **align/integrate** the libraries with each other, such that equivalent theories can be identified and contents can be reused and transferred between libraries.

Naturally, both aspects are heavily interrelated, since integrating libraries is easier after some suitable refactoring, and already aligned libraries can potentially be analyzed more easily and further refactored.

Even though there are several concepts that might be helpful in this objective, their effectiveness is currently very much speculative and the results are difficult to foresee.

Given the very general and foundation-independent nature of the MMT system, I think it is perfectly suitable to serve as a framework for both objectives; and fulfilling both could ideally lead to OMDoc/MMT becoming a functional basis for a general, modular and interlinked library of all formal mathematics, unconstrained by foundational aspects or a specific theorem prover system.

4 Methodology

4.1 Preliminary Work Plan

My work will be roughly divided into the following three work areas; the first two being theoretical research, the last one being case studies to evaluate the former two:

<table>
<tr><td>WA1</td><td>Extending LF (and MMT) with<table><tr><td>WP1.1</td><td>subtypes,</td></tr><tr><td>WP1.2</td><td>record types and</td></tr><tr><td>WP1.3</td><td>inductive types.</td></tr></table>(See Section 4.2)</td></tr>
<tr><td>WA2</td><td>Investigating methods for<table><tr><td>WP2.1</td><td>refactoring libraries and the theories contained therein, and</td></tr><tr><td>WP2.2</td><td>integrating formal libraries with each other.</td></tr></table>(See Section 4.3)</td></tr>
<tr><td>WA3</td><td>Evaluating the above methods as to their usefulness and effectiveness by<table><tr><td>WP3.1</td><td>making PVS accessible to MMT; the imported PVS library will serve as a case study for WA1 (See Section 4.4.1), and</td></tr><tr><td>WP3.2</td><td>Evaluating the methods from WA2, using the integrated libraries (PVS, TPS, HOLLight, Mizar, TPTP) as a case study (See Section 4.4.2).</td></tr></table></td></tr>
</table>

4.2 WA1: Extending LF

As mentioned before, LF is based on the dependently typed λ-calculus. While this provides powerful type constructors that allow for efficiently specifying particularly classic "textbook logics", when it comes to certain features which are used in "real-world" logics, such as the foundations of modern proof assistants, our experience has shown that LF is not powerful enough to model these elegantly.

In practice, it turns out that for these kind of type constructors to be nicely specifiable in a logical framework, the same feature has to already exist on the level of the logical framework itself. I therefore want to try to extend LF by providing type constructors and checking rules for these kinds of features. Implementing those in MMT will furthermore necessitate extending its available data structures and abstract syntax by additional term constructors, to accomodate i.e. record field assignments or pattern matching on inductive types (which are basically *constant declarations* on the *object level*, which OMDOC/MMT currently doesn't allow).

One such extension of LF (by finite sequences) and the associated problems in implementing them conveniently (necessitating features such as ellipses, lengths, concatenations, foldings etc.) has been described in [57].

It should be noted, that in extending LF by features such as *predicate subtyping*, we will necessarily lose decidability. However, this will mostly just introduce certain proof obligations for typing judgements, which have to be discharged by the user. Furthermore, since any formal library we want to import that makes use of these features will have the same problem, the necessary typing judgements are *ideally* already present in that library (to which extent will have to be seen). I therefore do not consider this to be a serious obstacle to our goals.

4.2.1 WP1.1: Subtypes

Consider as an example a judgement such as $A < B$ ("A is a subtype of B") in some logic (such as the type theory used by PVS). Since LF itself doesn't support subtyping, the kind "type" in this case has to be "pulled down" to a lower level, e.g. by declaring a new type tp, which serves as the (LF-)type of all "lower-level types", so we can declare both A and B to be of type tp, declare $<$ to be of type $tp \to tp \to type$ and introduce a judgment $AsubtpB : A < B$ (this is basically a *relativization*). But now declaring an element $a : A$ is impossible in LF, since A is not a type, but an element of type tp! So we have to introduce a new type $term$ and a new declaration \$ of type $term \to tp \to type$, and finally declare $a : term$ and $a\_type : a\$A$ to provide the information, that a has the (lower-level) type A.

However, LF now can not do proper type checking on the lower-level types, since now all typed objects have the same LF-type $term$. Furthermore, if we wanted to declare a function of type $f : B \to C$, we would have to declare that as having LF-type $term \to term$, losing the typing-information of f and making it difficult to distinguish between a well-defined expression such as $f(b)$ (where b has type B) and $f(q)$ for some term q on which f *should be* undefined. Furthermore, correctly type checking a valid expression such as $f(a)$ needs to make use of the information, that a is an element of type B, even though it has been declared to be of type A (since A is a subtype of B).

One way of implementing this directly in LF is of course to always carry the lower-level typing judgements as *additional arguments* in functions, i.e. f really has type $\Pi_{b:term}(b\$B) \to term$ and we introduce an additional declaration $f\_type : \Pi_{b:term,p:b\$B} f(b,p)\$C$, but this gets messy very quickly (obfuscating the actual mathematical content) and introduces a lot of overhead, since we "manually" have to carry all the declared *and inferred* (through subtyping) low-level typing judgements everywhere they might have to be used. Furthermore, we would have to take care of *proof irrelevance*, so that $f(b,p) = f(b,q)$ for every two proofs $p, q : b\$B$.

Alternatively, one could introduce an injective *type casting* function $AsubtpB : A \to B$ for every declaration $A < B$, allowing us to use LF-types directly. However, that would force us not just to keep track of any such declaration and "manually" appending the corresponding function(s) whenever we want to use an element of type A as an element of type B, but also to declare every property some $a : A$ might have again for all possible type casts such as $AsubtpB(a)$, again introducing a lot of overhead just to make sure that typecasted elements behave as needed.

A potential solution to this problem might result from *intersection types*[58] or *refinement types* [59], although it is questionable, whether they are powerful enough to allow for specifying e.g. predicate subtyping efficiently. For a discussion of subtyping in higher-order logic, see e.g. [60].

4.2.2 WP1.2: Record Types

An element of a record type is in principle a list of *key-value* pairs. If we ignore the keys, a record type is just a finite product, and the associated elements are tuples of corresponding values; however in ignoring the keys, we lose exactly the thing that makes record types valuable.

Consider as an example the record type {|name:String, age:Int|}. Obviously, that can be modeled as the type String × Int, however, in that case an element ("Steve", 25) would also be an element of the record type {|cityname:String, population:Int|}, which the intended semantics of record types should not allow for. Furthermore, an element of the type $A \times B \times C$ is not also an element of type $A \times B$, whereas the record type {|a:A, b:B, c:C|} should be a subtype of {|a:A, b:B|}, since an element of the former instantiates all the required record fields of the latter, namely a and b.

Specifying record types wih the intended semantics in OMDoc/Mmt is however inconvenient, since a definition of a record type like Person := {|name:String, age:Int|} introduces two new typed symbols, name and age. These are functionally declarations, but on the *object level*, which the current OMDoc/Mmt language does not allow for. The only current way to solve this is to model record types as theories T and elements of a record type as Mmt structures, that import T while providing the desired definitions for the symbols (i.e. record fields) specified in T. Unfortunately, this introduces a lot of undesired overhead, not just in specifying the types, but also because we are not able to use theories like types; i.e. a declaration like Steve:Person, having an anonymous element of type T within a term or binding a variable of that type are now impossible.

4.2.3 WP1.3: Inductive Types

Similar problems arise with inductive types. In principle, type constructors such as W-types (see e.g. [61]) can be used to specify inductive types. They are based on viewing inductive types as the *freely generated* types over some signature, of which only the arities matter. For example, the natural numbers are freely generated by two construction cases: $0 : \mathbb{N}$ (with arity 0) and the successor function $S : \mathbb{N} \to \mathbb{N}$ (with arity 1). Using W-types, the type $\mathbb{N}$ could thus be defined as $\mathbb{N} := W_{x:2}(0, 1)$.

W-types are easy to specify and implement in any sufficient language; but as with record types, we lose the information that the first element of the inductive definition is (supposed to be) called 0 and the second is called S. We can define 0 and S using the W-type for $\mathbb{N}$, but that results in definitions like

$$S := \lambda x_{\mathbb{N}}.sup(\top, \lambda y_{Unit}.x)$$

and

$$plus := \lambda n_{\mathbb{N}}.rec\left(\lambda a_{Bool}.\lambda u_{\beta(a)\to\mathbb{N}}.\lambda g_{\beta(a)\to\mathbb{N}}.((\lambda x_{Unit}.n) + (\lambda x_{Unit}.sup(\top, \lambda y_{Unit}.g(x))))(a)\right),$$

which are difficult to read and even more difficult to write, contrary to the ease of definitions like

$$plus := \lambda n.\lambda x.\ x\ match\ [case\ 0 => n;\ case\ S(m) => S(n + m)].$$

However, that would necessitate having the two cases 0 and S being intrinsically linked to the definition of the type $\mathbb{N}$, which – as with record types – calls for having additional declarations (of 0 and S) on the object level, namely in the definition of $\mathbb{N}$.

4.2.4 A Potential Solution to the Problems With Record and Inductive Types

As we have seen, the problems with specifying both record types and inductive types can be solved, if we are allowed to have declarations on the object level. One possible solution is therefore to extend the OMDoc/Mmt language by a new term type (preliminarily called) OML which wraps around a

declaration. To do so, the exact semantics of these terms will have to be worked out and the MMT system will have to be extended to allow for them.

4.2.5 Preliminary Results

So far, I have extended LF by coproducts, Σ-types, finite types (using coproducts, the empty type and the Unit type) and W-types and specified the corresponding inference and computation rules in a new MMT plugin. I do not expect them to be useful for **WP3.1**–**WP3.3**, however doing so has increased the depth of my understanding of the necessary MMT API methods. Furthermore, they might provide interesting toy examples for evaluating the resulting methods from **WA2** (e.g. by translating inductive types in some library to the corresponding W-types).

4.3 WA2: Refactoring and Integrating Libraries

As a second (and main) goal, I will look for useful refactoring methods of theories, such as *theory intersections* (see Section 4.3.1), to increase modularity, expose heterogeneity and ultimately investigate possible methods to efficiently identify overlap between and move knowledge across libraries.

The specific techniques to close in on that goal are very much an open question. Consequently, it is difficult to predict, what kind of difficulties I might encounter in doing so, what kind of results can be expected and what amount of work this will take.

Methods that might be useful in refactoring theories are theory intersections and methods for finding (partial) theory morphisms (see Section 4.3.1). Apart from improving and extending those, I want to look into the concept of **interface theories and logics** [38]: Assume, we are given e.g. a theory of real numbers implemented in some logic using some construction principle. We know, that the real numbers can be (up to cardinality and isomorphisms) uniquely described as the theory of topologically complete ordered archimedian fields – ergo, to state the necessary axioms, second-order predicate logic is sufficient. Moreover, any theory that *uses* the real numbers will (most likely) only use their second-order properties. We would therefore benefit from finding a way to seperate the concrete implementation of real numbers in our original logic from the *"actual"* theory of real numbers, which we would call the *interface theory* of the reals (of which our starting theory is just an extension), and the intricate construction principles used from the *interface logic*, that is actually used in asserting the relevant axioms, namely second-order predicate logic.

Implementing methods to (if possible) automatically find and extract interface theories (and refactor them to use the corresponding interface logic) could help identifying the heterogeneous part of a theory and enable **aligning** libraries by matching equivalent theories in different libraries automatically. Due to its modular nature, the LATIN library can provide the corrseponding interface logics.

Another potentially useful concept for translating content across libraries are *declaration patterns* [57], that – if consistently used in the import method for some library – might make it easier to match equivalent theories using different foundations.

4.3.1 Preliminary Results

So far, I have developed the following two methods, that might be useful for refactoring and aligning libraries:

Viewfinder The Viewfinder is an MMT method that tries to find useful partial morphisms between two theories. It has been tested on the LATIN library with experimental success, resulting in 20 – 3000 (depending on the configuration) morphisms on approximately 75000 pairs of theories. Since the LATIN library is modular by design, the low number of found theory morphisms is not too suprising; on inspection, most the found morphisms seem to be between theories that are similar-but-not-equal by design, such as Curry-Howard morphisms or morphisms between Church-style and Curry-style type

theories, where the corresponding theories deliberately do not share a common inclusion theory for historical and semantic reasons. Its algorithm is similar to a method described in [62].

It takes as arguments two MMT theories S, T, flattens them with respect to their theory inclusions and computes a datatype for every constant $c \in S \cup T$, consisting of two objects: An integer hash of the term structure of the type A of c and a list of constants (considered as *variables*) occuring in A. A check, whether two constants c and d are (superficially) compatible now reduces to an equality check on the integer hashes of c and d. To match c to d, the Viewfinder only needs to iterate on the respective lists of variables occuring in their types.

By choosing which constants are pulled into the integer hash of the term structure and which are considered variables, the user has some degree of control over which constants should remain fixed and which are allowed to be mapped to some other constant in the resulting morphisms.

Furthermore, by only trying to match those constants whose list of variables has some minimal length (given as a parameter), the user can trade the number of morphisms found for a relative increase of efficiency – the larger the minimal length, the smaller the number of morphisms and the faster the algorithm. Also, a certain minimal length avoids "meaningless" morphisms, that do not preserve at least some of the properties of a constant c (provided by respective axioms with some minimal number of occuring constants containing c).

Theory Intersections By theory intersection we mean, given a partial theory morphism σ between two theories S, T (as in Figure 4 on the left), extracting a common subtheory N, of which both S and T can be thought of as extensions (as in Figure 4 on the right, where N is isomorphic to the theories C, D, ρ_1, ρ_2 are MMT structures and S', T' are the refactored versions of S and T). e.g. the theory intersection between $(\mathbb{Z}, +, \cdot, <)$ and $(\mathbb{Q}, +, \cdot, <)$ along the obvious morphism would yield something like the theory of linearly ordered integral domain rings.

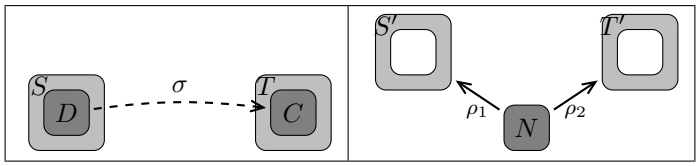

Figure 4: Theory Intersections

This allows for refactoring a given library in a modular way according to the little theories approach.

The theory intersection method takes as parameters two theories S, T, a name of the intersection theory and a list of tuples consisting of some constant in S, some constant in T, a name for the corresponding constant in the intersection and (optionally) a notation. The list of tuples can be computed from either a single morphism $S \to T$ or a pair of *compatible* morphisms $\sigma : S \to T$, $\delta : T \to S$, where σ and δ are considered compatible if for all $c \in S, d \in T$ we have $\delta(\sigma(c)) = c$ and $\sigma(\delta(d)) = d$, whenever these are well-defined.

Using the Viewfinder, the theory intersection method can in theory be automated and run over a whole library without any user input, provided we accept generated names for the resulting theories (the names for the constants in the intersection theory can be taken from one of the input theories).

I have integrated both the Viewfinder and the theory intersection method in the MMT plugin for jEdit as part of a new refactoring panel. An annotated video demonstration of the refactoring panel and its components can be found online [63].

4.4 WA3: Evaluation

4.4.1 WP3.1: Making PVS Accessible to MMT

My goal is to write an MMT plugin to allow for importing PVS libraries into the MMT system, using the extended LF from **WA1** as a logical framework. Doing so will entail three things:

1. Writing an MMT plugin that allows for importing .xml files generated by PVS into the MMT API (this has already been implemented, mostly by Florian Rabe with minor corrections by me) and translating these into MMT theories,
2. writing an MMT theory that provides the underlying foundational logic of PVS and
3. extending and improving the MMT language and -api, as well as LF, to provide syntactical constructors for the specific peculiarities of PVS (see **WA1**).

PVS [64] is a specification language integrated with support tools and a theorem prover. It is based on an extension of higher-order logic and is under active development at the Computer Science Laboratory of SRI International in Menlo Park, California. There exists a large library of mathematical theories for PVS, maintained by NASA's Langley Formal Methods Team [32], providing a particular incentive to integrate PVS into the OAF project. In addition, Sam Owre at SRI has provided us with an XML exporter for PVS[2], which makes reading library content into the MMT API a lot more convenient.

PVS has a very powerful (in general undecidable) type system and a rich language with many primitives, to make formalizing theorems and proofs as easy as possibly. In contrast, MMT is designed to be as flexible as possible, and consequently has very few primitives to allow the user to define them as needed. This will make the translation process from PVS to MMT possibly difficult, at the very least non-trivial. PVS offers e.g. record types, predicate subtyping, inductive and coinductive types, parametric theories etc., all of which will have to be specified in (or translated using) the corresponding MMT theory. As such, PVS can serve as a suitable case study for the extended version of LF from **WA1**.

The goal is to have the resulting import method run over the whole NASA library without errors and type checking correctly; both internally to the MMT API as well as after exporting its contents (at least) into the MMT language.

4.4.2 WP3.2: Evaluating the methods from WA2

WA2 is the most difficult part of my PhD; both with respect to the subject matter as well as in evaluating the results. Since this is very much an open problem, it is difficult to foresee what is feasible and what is not, as well as which specific challenges will occur.

Preliminarily, I will consider this part of my PhD a success, if we will be able to automatically identify at least some of the basic domains of mathematical discourse (booleans, natural numbers, rationals, reals,...) in two or more different library formats. The theory refactoring methods can furthermore be evaluated on the basis of resulting size reductions of theories, increase of the number of theories (as a measure of increased modularity) and inclusion relations and whether the input and output theory graphs are equal after flattening (with respect to theory inclusions and MMT structures).

The PVS NASA library as well as the Mizar, TPS and HOLLight libraries, which have already been integrated into MMT, will serve as a basis for this assessment.

4.5 Preliminary Schedule

The projected scheduling of these work areas (going forward) is visualized in Figure 5.

My initial focus will be on work area **WA1** and the corresponding case study **WP3.1**. In the process, I will furthermore (in collaboration with Florian Rabe) attempt to write a generic tutorial on how to do the same for arbitrary formal systems. We hope that this will ultimately enable other people outside of our research group to import libraries in e.g. their own theorem prover system into

[2] without proofs, which we deliberately do not include at this stage. Adding proofs to the goals presented in this proposal would add further major difficulties.

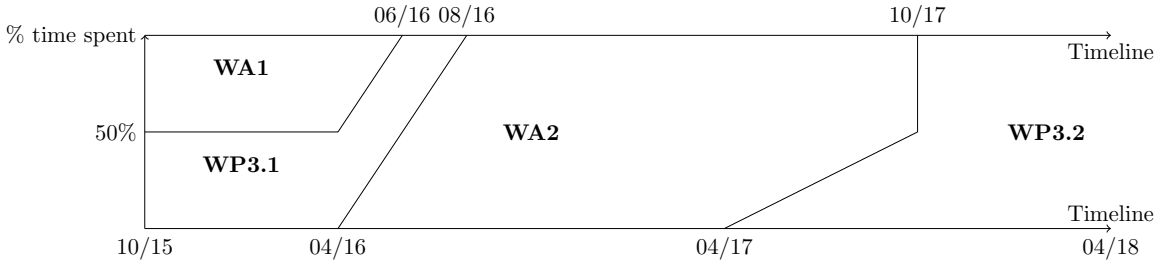

Figure 5: Preliminary Schedule

MMT. This might increase the number of available libraries faster than if someone intimately familiar with MMT is needed. The current status of this tutorial can be found in the MMT repository[3].

5 Conclusion

I have layed out the details of the incompatibility problem with respect to formal libraries. It should be clear, that – even though there are good reasons as to why the current automated theorem provers are sympathetic to the homogeneous method, despite of its negative aspects – the current state of the art is unsatisfactory if the ultimate goal is to have a *unified, general archive of formal mathematics.*

I believe that trying to take on the integration and incompatibility problem is consequently a worthwhile endeavour, and given the previous projects and results at the KWARC research group, as well as the availabilty of several different formal libraries within MMT, I think we are in a unique position to achieve at least partial solutions to this problem.

It should be noted that I am only proposing to work on formal "results", in the sense of theories containing definitions, theorems, axioms, but deliberately not the corresponding proofs. Extending the goals described in this proposal to proofs would introduce far greater challenges and vastly increase the difficulties in getting actually useful results, but might be a very worthwile endeavor subsequent to the work presented herein.

References

[1] Thomas Hales, Mark Adams, et al. *A formal proof of the Kepler conjecture*. 2015. URL: http://arxiv.org/abs/1501.02155.

[2] Ron Solomon. "On Finite Simple Groups and Their Classification". In: *Notices of the AMS* (Feb. 1995), pp. 231–239.

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

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

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

[6] Michael Kohlhase and Florian Rabe. "QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge". In: *Journal of Formalized Reasoning* (2015). in press. URL: http://kwarc.info/kohlhase/papers/qed20.pdf.

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

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

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

[10] 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.

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

[3] https://github.com/KWARC/MMT/tree/stable/doc/introduction/tutorial

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

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

[14] 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.

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

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

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

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

[19] 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.

[20] 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.

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

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

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

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

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

[26] 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.

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

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

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

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

[31] 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.

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

[33] 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).

[34] 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.

[35] 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.

[36] 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.

[37] Serge Autexier and Dieter Hutter. "Structure Formation in Large Theories". In: *Intelligent Computer Mathematics 2015*. LNCS. submitted. Springer, 2015, pp. 155–170.

[38] Michael Kohlhase, Florian Rabe, and Claudio Sacerdoti Coen. "A Foundational View on Integration Problems". In: *Intelligent Computer Mathematics*. Ed. by James Davenport, William Farmer, Florian Rabe, and Josef Urban. LNAI 6824. Springer Verlag, 2011, pp. 107–122. URL: http://kwarc.info/kohlhase/papers/cicm11-integration.pdf.

[39] 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.

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

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] M. Boespflug, Q. Carbonneaux, and O. Hermant. "The λΠ-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.

[46] 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.

[47] 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.

[48] 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.

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

[50] 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.

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

[52] Michael Kohlhase. "The OMDoc2 Language Design". KWARC Blue Note. 2013. URL: http://gl.kwarc.info/omdoc/blue/raw/master/design/note.pdf.

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

[54] 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.

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

[56] *The OAF Project & System*. URL: http://oaf.mathhub.info (visited on 04/23/2015).

[57] "A Framework for Defining Declarative Languages". PhD thesis. Jacobs University Bremen, Nov. 2014. URL: https://svn.kwarc.info/repos/fhorozal/pubs/phd-thesis.pdf.

[58] Frank Pfenning. *Intersection Types for a logical Framework*. POP-Report 92–106. miko: Carnegie Mellon University, 1992.

[59] Frank Pfenning. "Refinement Types for Logical Frameworks". In: *Informal Proceedings of the 1993 Workshop on Types for Proofs and Programs*. Ed. by Herman Geuvers. Nijmegen, The Netherlands: University of Nijmegen, May 1993, pp. 285–301.

[60] Michael Kohlhase. "A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle". PhD thesis. Universität des Saarlandes, 1994. URL: http://kwarc.info/kohlhase/papers/diss.pdf.

[61] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: http://homotopytypetheory.org/book, 2013.

[62] Immanuel Normann. "Automated Theory Interpretation". PhD thesis. Bremen, Germany: Jacobs University, 2008. URL: https://svn.eecs.jacobs-university.de/svn/eecs/archive/phd-2008/normann.pdf.

[63] *Theory Intersections in MMT*. URL: https://www.youtube.com/watch?v=qXKaGuV7kLY.

[64] 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.