Summary: Representing Logics and Logic Translations

Florian Rabe

1 Introduction

Logic is the study of formal languages for propositions and truth. Logics are used both as a foundation of mathematics and as specification languages in mathematics and computer science. Since logic is intricately intertwined with the nature of mathematics, the question how to represent logics in our minds is a constant challenge to our understanding. And only when it is understood can we begin to answer the corresponding question about logic translations. At the same time logics are used to a large extent in computer science to reason about both mathematics and software systems. This brings up the question how logics and their translations can be represented in a computer system. In this text we try to give answers to these fundamental research questions.

Throughout the 20^{th} century several answers have already been provided. Most of them can be grouped into two kinds, which can be denoted by set/model theory and type/proof theory. These two classes represent different research fields with conflicting philosophical and mathematical backgrounds, which makes their conceptualizations ontologically different. However, both fields have developed very sophisticated and strong solutions.

Therefore, we base our investigation on the desire to reconcile these two views. Our focus is on extending the existing notions of logic and logic translation in a way that retains their nature and the accumulated knowledge about them while leading them into a new direction. Our goal is to enrich both research fields by applying them more cogently to and making them more accessible and understandable to one another.

We choose institutions as a set/model theoretical and dependent type theory as a type/proof theoretical representative of the two kinds. We observe that both have complementary advantages that reflect their different backgrounds and devise our own answer by combining them in a way that consequently exploits their respective strengths. Mathematically, our main results can be summarized as follows. We define logics by extending institutions with notions of proof categories and proof theoretical truth that are very much parallel to the model categories and satisfaction relation of institutions. Then we give a concrete simple logic for dependent type theory using models inspired by Kripke models for intuitionistic logic. Finally, we show how to use this logic to define or encode logics and logic translations.

While this answers the question how to represent logics and logic translations in our minds, it is not adequate for the specific constraints and use cases of software systems. Therefore, in a second investigation, we explore how to make our representations more concrete and robust enough to permit a mechanized treatment on a large scale.

We choose OMDoc as a scalable, web-compatible representation language for mathematical knowledge. Because we find that its applicability to logical knowledge is limited, we revise it taking into account the characteristic requirements of logic. While keeping the motivation behind and flavor of OMDoc, we employ a different methodology more suitable to logic, thus effectively reinventing it. Mathematically our main results can be summarized as follows. We provide a formal abstract syntax for modular theory development and a formal semantics for it. Our module system treats logical frameworks, logics, and logical theories as well as the translations between them uniformly as theories and theory morphisms that are related via

the "is meta-language for" relation. And it consequently separates logic-independent and logic-specific language constructs, which lets us derive a logic-independent flattening theorem and constitutes the basis for logic-independent logical knowledge management services.

Taking these two results together, we obtain a triangle of different mathematical communities, research objectives, and philosophies consisting of set/model theory, proof/type theory, and mathematical knowledge management. Our main contribution is to integrate its corners into a coherent framework centered around logic that capitalizes on their comparative advantages.

The thesis consists of three parts. Part I reviews the historical works and the state of the art in the field of logic paying particular attention to logic translations and the representation of logics and logic translations. It describes in detail the relation between set/model and type/proof theory and their respective relations to mathematical knowledge management, and finally develops the research objects of "Combining Model and Proof Theory" and "Logical Knowledge Management", which serve as titles of the following parts. Part I also gives a formally rigorous top-down introduction to the mathematical foundations of logic to make the thesis self-contained, and a tutorial-style bottom-up introduction to support readers with the extremely abstract subject matter.

Part II consists of three chapters. The first one takes the model-theoretical logical framework of institutions and develops it into one that is more balanced between model and proof theory. We call an institution enriched with our proof theoretical structure a *logic*. Similarly, the second chapter takes the proof-theoretical logical framework of Martin-Löf's dependent type theory and adds a model-theoretical semantics to it. This yields a logic (in our sense) for dependent theory. Due to the fundamentally different nature of proof and model theoretical semantics of logics, these two efforts cannot simply meet in the middle. We unify them by using the logic developed in the second chapter as a meta-logic, in which other logics are represented. This is carried out in the third chapter, which also gives example representations of logics and logic translations.

Part III consists of three chapters as well. In the first one, we develop MMT, our simple yet expressive module system for mathematical theories. Being fully formal, foundation-independent, and scalable, it is designed to support logic (translation) representations in practice. In the second chapter, we apply MMT to the representations developed in Part II and show how the foundations of specific type theories and set theories can be recovered in it. In the last chapter, we describe the scalable infrastructure and its implementation that we have developed for MMT.

This summary is structured along the above outline of Parts II and III of the thesis.

2 Part II: Combining Model and Proof Theory

Since the Grundlagenkrise of mathematics logic has been an important research topic in mathematics and computer science. A central issue has always been what a logic actually is (let alone a logic translation). Over the course of the last hundred years researchers have provided very different answers to this question, and research areas that were initially connected have diverged and evolved into separate fields. While this specialization has led to very successful results, it has also created a split in research on logic that is sometimes detrimental.

Today we observe that there are two groups of logical frameworks: those based on set theoretical foundations of mathematics that characterize logics model theoretically, and those based on type theoretical foundations that characterize logics proof theoretically. The former go back to Tarski's view of consequence ([Tar33, TV56]) with institutions ([GB92, GR02]) and general logics ([Mes89]) being the most important examples. The latter are usually based on the Curry-Howard correspondence ([CF58, How80]), examples being Automath ([dB70]), Isabelle ([Pau94]), and the Edinburgh Logical Framework (LF, [HHP93]).

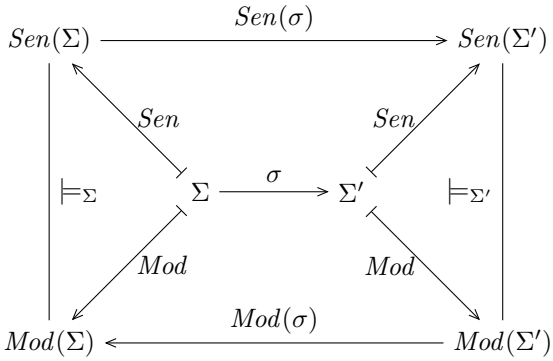
While some of these integrate the other side, such as the general proof theory developed in [Mes89], almost all of them lean towards either one. Often these sides are divided not only by research questions but also by "conflicting cultures and a[t]titudes", and "attempts to bridge these two cultures are rare and rather timid" (quoting an anonymous reviewer of a paper). This part of the thesis makes one such attempt, trying to provide a balanced framework that integrates and subsumes both views on logic in a way that preserves and exploits their respective advantages.

From the point of view of proof theory, we give a model theoretical semantics for logic encodings in dependent type theory by using institutions. And from the point of view of model theory, we give a specification language for institutions by using dependent type theory. From a neutral point of view, these results yield a logical framework that combines model theory and proof theory.

2.1 Extending Institutions with Proof Theory

The framework of institutions provides an abstract definition of the syntax and model theoretical semantics of a logic based on category theory. An institution is a tuple $(\mathit{Sig}, \mathit{Sen}, \mathit{Mod}, \models)$ where Sig is a category of signatures, and $\mathit{Sen} : \mathit{Sig} \to \mathcal{SET}$ and $\mathit{Mod} : \mathit{Sig} \to \mathcal{CAT}^{op}$ assign sentences and models to each signature; then the set of sentences and the class of models of a signature Σ are related via the satisfaction relation $\models_\Sigma$.

For given signatures Σ and Σ' and a signature morphism σ between them, the involved expressions can be visualized as follows:



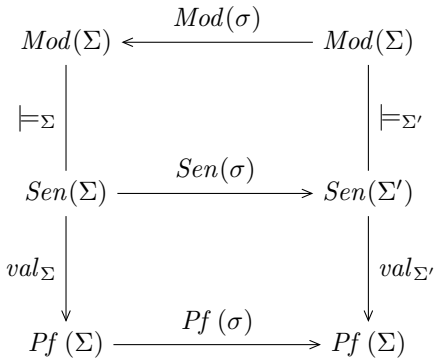
Readers not familiar with institutions should think of Σ and Σ' as the first-order signatures of monoids (i.e., declaring symbols for composition and unit element) and groups (i.e., extending monoids with a symbol for the inverse element), respectively, and of σ as the inclusion mapping from Σ to Σ'. Then $\mathit{Sen}(\Sigma)$ and $\mathit{Sen}(\Sigma')$ are the sets of first-order sentences over the respective signature, and $\mathit{Sen}(\sigma)$ is the mapping translating Σ-sentences to Σ'-sentences. In the monoid-group example, $\mathit{Sen}(\sigma)$ is an inclusion because the first-order language of monoids is contained in the one of groups.

Similarly, $\mathit{Mod}(\Sigma)$ and $\mathit{Mod}(\Sigma')$ are categories of models. Readers not familiar with category theory should think of $\mathit{Mod}(\Sigma)$ as the class of all monoids and $\mathit{Mod}(\Sigma')$ as the class of all groups. $\mathit{Mod}(\sigma)$ expresses the model reduction, a translation that goes against the signature translation. For our example signatures, $\mathit{Mod}(\sigma)$ is the mapping that maps every group to itself qua monoid. Models can often be viewed as mappings from the syntax of a logic to the set theory. For example, a monoid can be regarded as a mapping assigning functions to the symbols of Σ. In that case, reduction of a Σ'-model I along σ can be expressed as the composition $I \circ \sigma$. We come back to this intuition in Sect. 3.1.

Finally, $\models_\Sigma$ is the satisfaction relation: For a given model I and sentence F, it holds if I satisfies F.

We extend institutions to logics. A **logic** consists of a category of signatures Sig, a sentence functor Sen, a model category functor Mod, a proof category functor Pf , and model and proof theoretical definitions of truth $\models$ and val. Pf maps every signature to a category of proofs. The objects of these categories are the judgments – a concept from proof theory that is new in the context of institutions. A judgment can be, for example, a proposition, a sequent, a tableaux branch, or a set of clauses. Then the morphisms between families of judgments are the proofs. Finally, for every signature Σ, val_Σ is a mapping that assigns to every sentence a judgment expressing its validity; for example, if the judgments are sequents, val_Σ maps $A \in Sen(\Sigma)$ to $\vdash A$.

This can be summarized by the following diagram, also for a signature morphism $\sigma : \Sigma \to \Sigma'$, where $Pf\,(\sigma)$ translates Σ-judgements and proofs to Σ'. All arrows represent maps between classes, and the lines without arrow tips denote relations between classes. (For the sake of simplicity in this summary, this is not quite a diagram in the strict sense of category theory.)



Our definitions preserve the elegance and abstraction of institutions while paving the way for a natural integration of proof theoretical logics. Especially when viewed formally, we obtain a very elegant symmetry between the model theory $(Mod, \models)$ consisting of model category and model-theoretical definition of truth, and the proof theory $(Pf\,, val)$ consisting of proof category and proof-theoretical definition of truth.

Similar efforts have been undertaken before. In fact our definitions can be regarded (roughly) as a special case of the general logics of [Mes89]. Then our contribution lies in the specific choice that preserves the level of abstraction while elegantly incorporating more explicit proof theoretical notions. Furthermore, our definition of provability corresponds to the entailment systems of [Mes89]. Also, our work was directly motivated by the proof theoretic institutions of [MGDT05] and [Dia06].

2.2 A Model Theory for Dependent Type Theory

Martin-Löf type theory, MLTT, is a dependent type theory ([ML74]). The main characteristic is that there are type-valued function symbols that take terms as input and return types as output. This is enriched with further type constructors such as dependent sum and product. The syntax of dependent type theory is significantly more complex than that of simple type theory, because well-formed types and terms and both their equalities must be defined in a single joint induction.

The semantics of MLTT is similarly complicated. In [See84], the connection between MLTT and locally cartesian closed (LCC) categories was first established. LCC categories interpret contexts Γ as objects $[\![\Gamma]\!]$, types in context Γ as objects in the slice category over $[\![\Gamma]\!]$, substitution as pullback, and dependent sum and product as left and right adjoint to pullback. But there is a difficulty, namely that these three operations are not independent: Substitution of terms into types is associative and commutes with sum and product formation, which is not necessarily the

case for the choices of pullbacks and their adjoints. This is known as the coherence or strictness problem and has been studied extensively. In incoherent models, equal types are interpreted as isomorphic, but not necessarily equal objects such as in [Cur89]. In [Car86], coherent models for MLTT were given using categories with attributes. And in [Hof94], a category with attributes is constructed for every LCC category. Several other model classes and their coherence properties have been studied in, e.g., [Str91] and [Jac90]. In [Pit00], an overview is given.

These model classes all have in common that they are rather abstract and have a more complicated structure than general LCC categories. It is clearly desirable to have simpler, more concrete models. But it is a hard problem to equip a given LCC category with choices for pullbacks and adjoints that are both natural and coherent. Our motivation is to find a simple concrete class of LCC categories for which such a choice can be made, and which is still general enough to be complete for MLTT.

Mathematically, our main results can be summarized very simply: Using a theorem from topos theory, it can be shown that MLTT is complete with respect to – not necessarily coherent – models in the LCC categories of the form $\mathcal{SET}^P$ for posets P. And for these rather simple models, we can give a solution to the coherence problem. $\mathcal{SET}$ can be equipped with a coherent choice of pullback functors, and hence the categories $\mathcal{SET}^P$ can be as well. Deviating subtly from the well-known constructions, we can also make coherent choices for the required adjoints to pullback. Finally, rather than working in the various slices $\mathcal{SET}^P/A$, we use the isomorphism $\mathcal{SET}^P/A \cong \mathcal{SET}^{\int_P A}$, where $\int_P A$ is the Grothendieck construction: Thus we can formulate the semantics of dependent types uniformly in terms of the simple categories of indexed sets $\mathcal{SET}^Q$ for various posets Q.

In addition to being easy to work with, this has the virtue of capturing the idea that a dependent type S in context Γ is in some sense a type-valued function on Γ: Our models interpret Γ as a poset $[\![\Gamma]\!]$ and S as an indexed set $[\![\Gamma|S]\!] : [\![\Gamma]\!] \to \mathcal{SET}$. We speak of Kripke models because these models are a natural extension of the well-known Kripke models for intuitionistic first-order logic ([Kri65]). Such models are based on a poset P of worlds, and the universe is given as a P-indexed set (possibly equipped with P-indexed structure). This can be seen as the special case of our semantics when there is only one base type.

In fact, our results are also interesting in the special case of simple type theory ([Chu40]). Contrary to Henkin models [Hen50, MS89], and the models given in [MM91], which like ours use indexed sets on posets, our models are standard: The interpretation $[\![\Gamma|S \to S']\!]$ of the function type is the exponential of $[\![\Gamma|S]\!]$ and $[\![\Gamma|S']\!]$. And contrary to the models in [Fri75, Sim95], our completeness result holds for theories with more than only base types and terms.

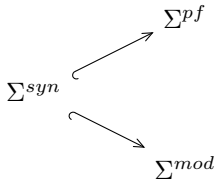
Due to the strong mathematical prerequisites needed for this chapter, we refrain from going into more details, and only sketch how we obtain a logic for dependent type theory. *Sig* consists of the signatures and signature morphisms of dependent type theory. For a given signature Σ, $Sen(\Sigma)$ contains the types over this signature – following the propositions-as-types paradigm. $Mod(\Sigma)$ contains our Kripke models of Σ, the judgments in $Pf\ (\Sigma)$ are the contexts of dependent type theory, and the proofs between two such contexts are the substitutions. For a type $S \in Sen(\Sigma)$, both $\models_\Sigma$ and val_Σ are given by inhabitation of S: Whereas the former is given model-theoretically by inhabitation of S in a model (i.e., by whether $[\![S]\!]$ is non-empty), the latter is given proof-theoretically by inhabitation S in the type theory.

2.3 Logic Representations in a Meta-Logic

In this chapter, we use the logic we obtained for dependent type theory before as a meta-logic, in which other logics and logic translations are represented. Our definition of a meta-logic for dependent type theory can be regarded as the continuation of work undertaken in [HST94] and [Tar96]. Our work adds a model theoretic perspective to the former and carries out the use of a meta-institution for dependent type theory suggested in the latter. Our definitions are

chosen carefully to permit to extend the available proof theoretic logic encodings in LF (e.g., [HHP93, HST94, AHMP92, Pfe01, Pfe00, AHMP98]) with encodings of their model theoretic counterparts.

The crucial idea of these encodings is that dependent type theory is expressive enough to represent proof and model theory of other logics. Intuitively (and somewhat simplified compared to the thesis), to encode a logic L, every signature Σ of L is represented as a triple $(\Sigma^{syn}, \Sigma^{pf}, \Sigma^{mod})$ of signatures of dependent type theory as in the diagram on the right. These signatures represent syntax, proof theory, and model theory of Σ.



The sentences of Σ are represented as terms of a distinguished type o over Σ^{syn}. The judgments of Σ are represented by using distinguished types over Σ^{pf}. The Σ-models are represented as Kripke models of Σ^{mod}. As a very simple example, we give an encoding of a fragment of propositional logic.

$$
\begin{array}{lcl}
PL^{syn} & = & o{:}\mathtt{type},\ \supset{:}o \to o \to o,\ \mathit{true}:o \to \mathtt{type} \\
PL^{pf} & = & PL^{syn}, \\
 & & \supset I{:}(\mathit{true}\ A \to \mathit{true}\ B) \to \mathit{true}\ (A \supset B), \\
 & & \supset E{:}\mathit{true}\ (A \supset B) \to \mathit{true}\ A \to \mathit{true}\ B \\
PL^{mod} & = & PL^{syn}, \\
 & & 0{:}o,\ \ 1{:}o,\ \ \mathit{desig}0{:}\mathit{true}\ 0 \to \mathtt{void},\ \ \mathit{desig}1{:}\mathit{true}\ 1, \\
 & & \mathit{boole}{:}\Pi_{A:o}\,((A=0)+(A=1)), \\
 & & \supset 1{:}((A=0)+(B=1)) \to (A \supset B = 1), \\
 & & \supset 0{:}((A=1)*(B=0)) \to (A \supset B = 1)
\end{array}
$$

Here $S \to T$, $S+T$, and $S*T$ are the function, union, and product type of S and T, $\Pi_{x:S}\,T$ is the dependent product, $\mathtt{void}$ is the empty type, and $(A = A')$ is the identity type of A and A'. PL^{syn} declares a type o for sentences, a constant $\supset$ for implication, and a type family true where $\mathit{true}\ F$ is the truth judgment of F, i.e., the type whose inhabitation represents the truth of F. PL^{pf} adds natural deduction introduction and elimination rules to PL^{syn}. Finally PL^{mod} axiomatizes models of propositional logic. PL^{mod} first declares elements 0 and 1 representing the truth values. Then $\mathit{desig}0$ and $\mathit{desig}1$ axiomatize the non-inhabitation of $\mathit{true}\ 0$ (i.e., $[\![\mathit{true}\ 0]\!] = \varnothing$) and the inhabitation of $\mathit{true}\ 1$ (i.e., $[\![\mathit{true}\ 1]\!] \neq \varnothing$) – in other words they represent that 1 is the designated truth value. Then boole is an axiom that guarantees $[\![o]\!]$ is a two-element set, i.e., the usual Boolean lattice. Finally, $\supset 1$ and $\supset 0$ axiomatize the model-theoretical definition of the truth of implication: For example, $\supset 1$ implies $[\![\mathit{true}\ A \supset B]\!] = [\![\mathit{true}\ 1]\!] \neq \varnothing$ whenever the union type $(A = 0) + (B = 1)$ is inhabited, and that is the case when $[\![A]\!] = [\![0]\!]$ or $[\![B]\!] = [\![1]\!]$.

Finally, we give general results how logics and logic translations can be both defined and represented in our framework. For example, the soundness of L can be proved by giving a signature morphism from Σ^{pf} to Σ^{mod}, and such morphisms can be found partially and verified fully mechanically. In the thesis, we worked out the translation of modal logic into first-order logic as a running example. Larger case studies have been carried out since then (see, e.g., [RH09]) – using implementations of the module system developed in Part III.

3 Part III: Logical Knowledge Management

3.1 MMT: A Scalable Module System for Mathematical Theories

Mathematical knowledge is at the core of science, engineering, and economics, and we are seeing a trend towards employing computational systems like semi-automated theorem provers, model checkers, computer algebra systems, constraint solvers, or concept classifiers to deal with it. It is a characteristic feature of these systems that they either have mathematical knowledge implicitly encoded in their critical algorithms or (increasingly) manipulate explicit

representations of the relevant mathematical knowledge often in the form of logical formulas. Unfortunately, these systems have differing domains of applications, foundational assumptions, and input languages, which makes them non-interoperable and difficult to compare and evaluate in practice. Moreover, the quantity of mathematical knowledge is growing faster than our ability to formalize and organize it, aggravating the problem that mathematical software systems cannot share knowledge representations.

The work reported in this paper focuses on developing an exchange format between math-knowledge based systems. We concentrate on a foundationally unconstrained framework for knowledge representation that allows to represent the meta-theoretic foundations of the mathematical knowledge in the same format and to interlink the foundations at the meta-logical level. In particular, the logical foundations of domain representations for the mathematical knowledge can be represented as modules themselves and can be interlinked via meta-morphisms. This "logics-as-theories" approach makes systems behavior as well as their represented knowledge interoperable and thus comparable at multiple levels. The explicit representation of epistemic foundations also benefits systems whose mathematical knowledge is only implicitly embedded into the algorithms. Here, the explicit representation can serve as a documentation of the system interface as well as a basis for verification or testing attempts.

Of course communication by translation to the lowest common denominator logic – the current state of the art – is always possible. But such translations lose the very structural properties of the knowledge representation that drive computation and led to the choice of logical system in the first place. Therefore our format incorporates a module system geared to support flexible reuse of knowledge items via theory morphisms. This module system is the central part of the proposed format – emphasizing interoperability between theorem proving systems, and the exchange and reusability of mathematical facts across different systems. In contrast to the formula level, which needs to be ontologically unconstrained, the module system level must have a clear semantics (relative to the semantics of the formula level) and be expressive enough to encode current structuring practice so that systems can communicate without losing representational structure.

On a practical level, an exchange format must be *scalable* in the sense that it supports the distribution of resources (theories, conjectures, proofs, etc.) over the internet, so that they can be managed collaboratively. In the current web architecture this means that all (relevant) resources must be addressable by a URI-based naming scheme [BLFM05]. Note that in the presence of a complex modularity and reusability infrastructure, this may mean that resources have to be addressable, even though they are only virtually induced by the inheritance structure.

Finally the design, implementation, and maintenance of large scale logical knowledge management services will realistically only pay off if the same framework can be reused for different foundations of mathematics. Therefore, an interface layer is needed between the logical-mathematical core of a mathematical foundation and the needs of a foundation-independent knowledge management service that has to preserve the semantics of the knowledge it operates on without knowing it.

MMT was designed as such a representation language. It represents logical knowledge on three levels: the module, symbol, and object level. On the module level, we build on modular representation languages for logical knowledge such as OBJ [GWM+93], ASL [SW83], development graphs [AHMS99], and CASL [CoF04]. In particular, we carry on the rigorous use of theories and theory morphism as the primitive modular concepts and the distinction between definitional and postulated links as the concepts relating theories. This distinguishes MMT from module systems that have been developed for type theories such as PVS [ORS92], Isabelle [Pau94], Coq [BC04], or Nuprl [CAB+86].

On the symbol level, MMT has in common with type theories the use of typed, possibly defined constants as the primitive concept. In particular, MMT uses the Curry-Howard correspondence ([CF58, How80]) to represent axioms and theorem as constants, and proofs as terms. This distinguishes MMT from the logic-oriented module systems mentioned above.

Finally, on the object level, MMT uses the formal grammar of OPENMATH [BCC+04] to represent terms without committing to a specific formal foundation. Specific foundations are defined by defining judgments for typing and equality of terms, in which MMT is parametric. This distinguishes MMT from the logical and type theoretical languages above, which are either designed for one foundation (such as type theories) or pose heavyweight requirements on the foundation (such as being an institution in the case of CASL).

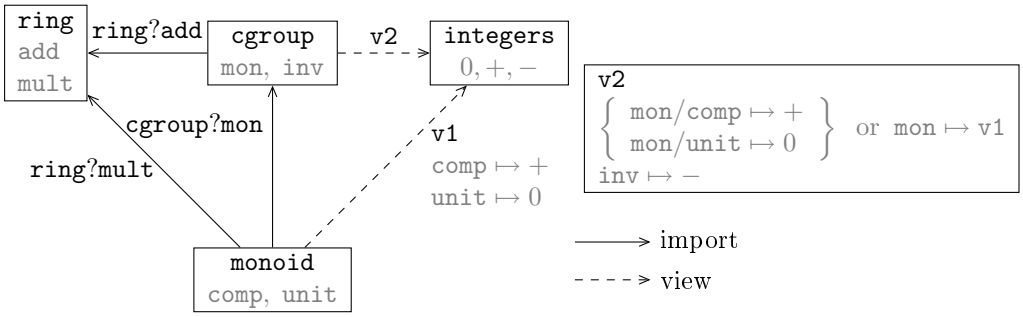


Figure 1: Example

To illustrate the features and feel of MMT, we give a simple example that shows some of its crucial features: Fig. 1 gives a theory graph for a portion of the algebraic hierarchy. The bottom node represents the theory of monoids, which declares constants for composition and unit. (We omit the axioms and the types here.) The theory `cgroup` for commutative groups arises by importing from `monoid` and adding a constant `inv` for the inverse element. This theory does not have to declare constants for composition and unit again because they are imported from the theory of monoids. The import is named `mon`, and it can be referenced by the qualified name `cgroup?mon`; it induces a theory morphism from monoids to commutative groups.

Then the theory of rings can be formed by importing from `monoid` (via an import named `mult` that provides the multiplicative structure) and from `cgroup` (via an import named `add` that provides the additive structure). Because all imports are named, different import paths can be distinguished: By concatenating import and symbol names, the theory `ring` can access the symbols `ring?add/mon/comp` (addition), `ring?add/mon/unit` (zero), `ring?add/inv` (additive inverse), `ring?mult/comp` (multiplication), and `ring?mult/unit` (one).

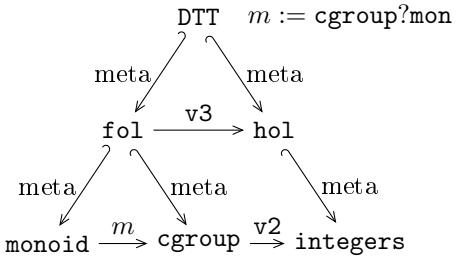
MMT represents models as theory morphisms. The node on the right side of the graph represents a theory for the integers declaring the constants 0, +, and −. The fact that the integers are a monoid is represented by the view `v1`. It is a theory morphism that is explicitly given by its interpretations of `comp` as + and of `unit` as 0. (If we did not omit axioms, this view would also have to interpret all the axioms of `monoid` as – using Curry-Howard representation – proof terms.)

The view `v2` is particularly interesting because there are two ways to represent the fact that the integers are a commutative group. Firstly, all operations of `cgroup` can be interpreted as terms over `integers`: This means to interpret `inv` as − and the two imported constants `mon/comp` and `mon/unit` as + and 0, respectively. Secondly, `v2` can be constructed along the modular structure of `cgroup` and use the existing view `v1` to interpret all constants imported by `mon`. In MMT, this can be expressed elegantly by the interpretation $\texttt{mon} \mapsto \texttt{v1}$, which interprets a named import with a theory morphism. The intuition behind such an interpretation is that it makes the right triangle commute: $\texttt{v2} \circ \texttt{cgroup?mon} = \texttt{v1}$. Clearly, both ways lead to the same theory morphism; the second one is conceptually more complex but eliminates redundancy. (This redundancy is especially harmful when axioms are considered, which must be interpreted as expensive-to-find proofs.)

3.2 Representing Foundations and Logics in MMT

A central structuring mechanism of MMT is the meta-theory relation, which is a binary relation between theories. The intuition is that the meta-theory M of T provides the syntactic material that is used to define the semantics of the symbols declared in T.

For example, we can add meta-theories to the above example as in the figure on the right. `fol` is a theory for first-order logic, in which monoids and groups are axiomatized, `hol` is a theory for higher-order logic, in which the integers are defined. We can iterate this step and define both `fol` and `hol` using our framework of dependent type theory introduced in Part II as a further meta-theory.



Because M is always included into T, any theory morphism with domain T must also map the symbols of M. This is the role of the meta-morphism from M to the codomain of the theory morphism (or its meta-theory). For `cgroup?mon`, this is the identity $id_{\texttt{fol}}$ because domain and codomain have the same meta-theory. For `v2`, we need to give a meta-morphism `v3` from `fol` to `hol`.

In MMT, the topmost meta-theories are called the foundations. Foundation-independence of MMT means that the semantics of any MMT theory is determined by the semantics of its meta-theory. The semantics of the foundations can be easily specified for a variety of set and type theories. We have given examples for Zermelo-Fraenkel set theory and our DTT meta-logic.

In fact, we can go even further by representing translations between different foundations, e.g., with a view from DTT to Isabelle ([Pau94]). For example, the integers as defined in Isabelle can be used as a model of monoids. In that case MMT is able to bridge between different mathematical foundations.

3.3 A Web-Scale Infrastructure

MMT is designed to provide a web-scalable infrastructure for the management of logical knowledge. For example, its concrete syntax is an XML language incorporating established web-scalable formats for mathematics ([BCC+04, ABC+03, Koh06]), its identifiers are URIs ([BLFM05]), and its typing system specifies a RESTful HTTP interface to an MMT server ([Fie00]).

Using notation definitions, MMT documents can be rendered into arbitrary human- or machine-oriented formats. Finally the design and implementation of management of change, versioned persistent storage, user model-driven document presentation, and collaborative editing of MMT documents are under way in collaboration with other members of the Kwarc research group at Jacobs University Bremen (see, e.g., [KLR07, KMR08, GLR09]).

Just recently, we integrated the special case of MMT for a foundation for LF ([HHP93, PS99], a proof theoretical framework based on dependent type theory) into the Twelf implementation of LF ([RS09]). Conversely, a plugin for LF has been implemented for the MMT implementation. A similar integration with the Hets implementation of CASL ([CoF04, MML07], a model theoretical logical framework based on institutions) is planned.

This infrastructure enables us to design and represent large logics and logic translations efficiently and with web-scale machine-support while retaining the formal reliability assurances of logic.

4 Conclusion

Our work was motivated by the goals of representing logics and logic translations in our minds and in machines. Regarding the first goal, we made a significant contribution to the foundations

of computer science and logic. By integrating the often competing approaches of proof and model theory, we obtained a seminal new perspective on logic. And we provided a logical framework that formalizes and exploits this perspective.

There are some challenging open theoretical problems regarding our framework, most notably to investigate generalized completeness proofs in it. However, the most important next research task is more practical, namely to apply the framework to the representation of large scale logics and logic translations. By large scale, we mean logics with sophisticated tool support and libraries that are successfully used in software engineering such as PVS ([ORS92]) or Isabelle ([Pau94]), but for which interoperability is a big problem.

An important issue that will come up is the need to formalize and represent incomplete translations that are used in implementations. Recently such translations have been employed very successfully to borrow automated reasoning support from other tools, e.g., in Leo ([BPTF07]) or Isabelle. These translations often employ non-trivial encoding steps to match (parts of) the logics of the two systems. Due to their ad hoc nature, logical frameworks have not been used for them so far.

A long term goal is to build a universal logic graph that represents the most important logics and their interrelations. This will serve both as a documentation platform for research and education and as a translation network to be applied in human, interactive, and automated reasoning. A short term step towards this goal is an integration or our framework and the Hets system ([MML07]).

Regarding the second goal, we explored a research area that we called logical knowledge management. Here we investigated the boundaries between logic research and applications, for which correctness and reliability are paramount, on one side, and knowledge management, which considers also scalability and interoperability, on the other side. With MMT, we introduced an interface language at this boundary that integrates both sides' methods, assumptions, and priorities.

MMT focuses on choosing the right primitive knowledge representation concepts and providing a scalable architecture on top of them. Here architecture refers both to the formal definitions and the implementation. MMT permits to represent all aspects of logics and logic translations in one unified framework, and offers a simple and expressive module system to do so in a scalable way.

A central question of current and future research is how to improve representations in MMT with respect to non-structural and partial logic translations. Specialized frameworks such as Twelf ([PS99]) and Hets do that by using general purpose logic or functional programming languages, respectively, whereas MMT is designed to be declarative. We find the conception of a dedicated logic translation language based on MMT the right compromise.

A long term goal of MMT is to evolve into the representation format for the universal logic graph mentioned above. Furthermore, MMT will provide the interface layer between the formal mathematical semantics of the graph and the knowledge management services employed to edit, maintain, search, and view the graph.

References

[ABC+03] R. Ausbrooks, S. Buswell, D. Carlisle, S. Dalmas, S. Devitt, A. Diaz, M. Froumentin, R. Hunter, P. Ion, M. Kohlhase, R. Miner, N. Poppelier, B. Smith, N. Soiffer, R. Sutor, and S. Watt. Mathematical Markup Language (MathML) Version 2.0 (second edition). Technical report, World Wide Web Consortium, 2003. See http://www.w3.org/TR/MathML2. 9

[AHMP92] A. Avron, F. Honsell, I. Mason, and R. Pollack. Using typed lambda calculus to implement formal systems on a machine. *Journal of Automated Reasoning*, 9(3):309–354, 1992. 6

[AHMP98] B. Avron, F. Honsell, M. Miculan, and C. Paravano. Encoding modal logics in logical frameworks. *Studia Logica*, 60(1):161–208, 1998. 6

[AHMS99] S. Autexier, D. Hutter, H. Mantel, and A. Schairer. Towards an Evolutionary Formal Software-Development Using CASL. In D. Bert, C. Choppy, and P. Mosses, editors, *WADT*, volume 1827 of *Lecture Notes in Computer Science*, pages 73–88. Springer, 1999. 7

[BC04] Y. Bertot and P. Castéran. *Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004. 7

[BCC+04] S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaetano, and M. Kohlhase. The Open Math Standard, Version 2.0. Technical report, The Open Math Society, 2004. See http://www.openmath.org/standard/om20. 8, 9

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force, 2005. 7, 9

[BPTF07] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. The LEO-II Project. In *Automated Reasoning Workshop*, 2007. 10

[CAB+86] R. Constable, S. Allen, H. Bromley, W. Cleaveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. Sasaki, and S. Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, 1986. 7

[Car86] J. Cartmell. Generalized algebraic theories and contextual category. *Annals of Pure and Applied Logic*, 32:209–243, 1986. 5

[CF58] H. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958. 2, 7

[Chu40] A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5(1):56–68, 1940. 5

[CoF04] CoFI (The Common Framework Initiative). *Casl Reference Manual*, volume 2900 (IFIP Series) of *LNCS*. Springer, 2004. 7, 9

[Cur89] P. Curien. Alpha-Conversion, Conditions on Variables and Categorical Logic. *Studia Logica*, 48(3):319–360, 1989. 5

[dB70] N. de Bruijn. The Mathematical Language AUTOMATH. In M. Laudet, editor, *Proceedings of the Symposium on Automated Demonstration*, volume 25 of *Lecture Notes in Mathematics*, pages 29–61. Springer, 1970. 2

[Dia06] R. Diaconescu. Proof systems for institutional logic. *Journal of Logic and Computation*, 16(3):339–357, 2006. 4

[Fie00] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. 9

[Fri75] H. Friedman. Equality Between Functionals. In R. Parikh, editor, *Logic Colloquium*, pages 22–37. Springer, 1975. 5

[GB92] J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992. 2

[GLR09] J. Giceva, C. Lange, and F. Rabe. Integrating Web Services into Active Mathematical Documents. 2009. Accepted at Conference on Mathematical Knowledge Management (MKM), see http://kwarc.info/frabe/Research/AErabe_jobad_09.pdf. 9

[GR02] J. A. Goguen and G. Rosu. Institution morphisms. *Formal Aspects of Computing*, 13:274–307, 2002. 2

[GWM+93] J. Goguen, Timothy Winkler, J. Meseguer, K. Futatsugi, and J. Jouannaud. Introducing OBJ. In Joseph Goguen, editor, *Applications of Algebraic Specification using OBJ*. Cambridge, 1993. 7

[Hen50] L. Henkin. Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15(2):81–91, 1950. 5

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993. 2, 6, 9

[Hof94] M. Hofmann. On the Interpretation of Type Theory in Locally Cartesian Closed Categories. In *CSL*, pages 427–441. Springer, 1994. 5

[How80] W. Howard. The formulas-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 479–490. Academic Press, 1980. 2, 7

[HST94] R. Harper, D. Sannella, and A. Tarlecki. Structured presentations and logic representations. *Annals of Pure and Applied Logic*, 67:113–160, 1994. 5, 6

[Jac90] B. Jacobs. *Categorical Type Theory*. PhD thesis, Catholic University of the Netherlands, 1990. 5

[KLR07] M. Kohlhase, C. Lange, and F. Rabe. Presenting Mathematical Content with Flexible Elisions. In *Proceedings of the OpenMath/JEM workshop*, 2007. 9

[KMR08] M. Kohlhase, C. Müller, and F. Rabe. Notations for Living Mathematical Documents. In S. Autexier and J. Campbell and J. Rubio and V. Sorge and M. Suzuki and F. Wiedijk, editor, *Mathematical Knowledge Management*, volume 5144 of *Lecture Notes in Computer Science*, pages 504–519, 2008. 9

[Koh06] M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer, 2006. 9

[Kri65] S. Kripke. Semantical Analysis of Intuitionistic Logic I. In J. Crossley and M. Dummett, editors, *Formal Systems and Recursive Functions*, pages 92–130. North-Holland, 1965. 5

[Mes89] J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium, 1987*, pages 275–329. North-Holland, 1989. 2, 3, 4

[MGDT05] T. Mossakowski, J. Goguen, R. Diaconescu, and A. Tarlecki. What is a logic? In J. Béziau, editor, *Logica Universalis*, pages 113–133. Birkhäuser Verlag, 2005. 4

[ML74] P. Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In *Proceedings of the '73 Logic Colloquium*. North-Holland, 1974. 4

[MM91] J. Mitchell and E. Moggi. Kripke-style Models for Typed Lambda Calculus. *Annals of Pure and Applied Logic*, 51(1–2):99–124, 1991. 5

[MML07] T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522, 2007. 9, 10

[MS89] J. Mitchell and P. Scott. Typed lambda calculus and cartesian closed categories. In *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 301–316. Amer. Math. Society, 1989. 5

[ORS92] S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Springer, 1992. 7, 10

[Pau94] L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994. 2, 7, 9, 10

[Pfe00] F. Pfenning. Structural cut elimination: I. intuitionistic and classical logic. *Information and Computation*, 157(1-2):84–141, 2000. 6

[Pfe01] F. Pfenning. Logical frameworks. In *Handbook of automated reasoning*, pages 1063–1147. Elsevier, 2001. 6

[Pit00] A. Pitts. Categorical Logic. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, chapter 2, pages 39–128. Oxford University Press, 2000. 5

[PS99] F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. *Lecture Notes in Computer Science*, 1632:202–206, 1999. 9, 10

[RH09] F. Rabe and Fulya Horozal. A formal proof of the soundness of first-order logic, 2009. See `https://svn.kwarc.info/repos/twelf/fol-soundness.cfg`. 6

[RS09] F. Rabe and C. Schürmann. A Practical Module System for LF. Submitted to Conference on Automated Deduction (CADE), see `http://kwarc.info/frabe/Research/lf.pdf`, 2009. 9

[See84] R. Seely. Locally cartesian closed categories and type theory. *Math. Proc. Cambridge Philos. Soc.*, 95:33–48, 1984. 4

[Sim95] A. Simpson. Categorical completeness results for the simply-typed lambda-calculus. In M. Dezani-Ciancaglini and G. Plotkin, editor, *Typed Lambda Calculi and Applications*, pages 414–427, 1995. 5

[Str91] T. Streicher. *Semantics of Type Theory*. Springer-Verlag, 1991. 5

[SW83] D. Sannella and M. Wirsing. A Kernel Language for Algebraic Specification and Implementation. In M. Karpinski, editor, *Fundamentals of Computation Theoryational*, pages 413–427. Springer, 1983. 7

[Tar33] A. Tarski. Pojęcie prawdy w językach nauk dedukcyjnych. *Prace Towarzystwa Naukowego Warszawskiego Wydzial III Nauk Matematyczno-Fizycznych*, 34, 1933. English title: The concept of truth in the languages of the deductive sciences. 2

[Tar96] A. Tarlecki. Moving between logical systems. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specifications. 11th Workshop on Specification of Abstract Data Types*, volume 1130 of *Lecture Notes in Computer Science*, pages 478–502. Springer Verlag, 1996. 5

[TV56] A. Tarski and R. Vaught. Arithmetical extensions of relational systems. *Compositio Mathematica*, 13:81–102, 1956. 2