Math. Struct. in Comp. Science (2013), *vol.* 23, *pp.* 945–1001. © Cambridge University Press 2013
doi:10.1017/S0960129512000424 First published online 1 March 2013

A logical framework combining model and proof theory

FLORIAN RABE[†]

Computer Science, Jacobs University Bremen,
Campus Ring 1, 28759 Bremen, Germany
Email: f.rabe@jacobs-university.de

Received 12 October 2010; revised 12 December 2011

Mathematical logic and computer science have driven the design of a growing number of logics and related formalisms such as set theories and type theories. In response to this population explosion, logical frameworks have been developed as formal meta-languages in which to represent, structure, relate and reason about logics.
Research on logical frameworks has diverged into separate communities, often with conflicting backgrounds and philosophies. In particular, two of the most important logical frameworks are the framework of institutions, from the area of model theory based on category theory, and the Edinburgh Logical Framework LF, from the area of proof theory based on dependent type theory. Even though their ultimate motivations overlap – for example in applications to software verification – they have fundamentally different perspectives on logic.
In the current paper, we design a logical framework that integrates the frameworks of institutions and LF in a way that combines their complementary advantages while retaining the elegance of each of them. In particular, our framework takes a balanced approach between model theory and proof theory, and permits the representation of logics in a way that comprises all major ingredients of a logic: syntax, models, satisfaction, judgments and proofs. This provides a theoretical basis for the systematic study of logics in a comprehensive logical framework. Our framework has been applied to obtain a large library of structured and machine-verified encodings of logics and logic translations.

1. Introduction

Since the foundational crisis of mathematics in the early twentieth century, logic has been an important research topic in mathematics, and later in 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: for example, even as recently as 2005, the World Congress on Universal Logic held a contest about what a logic is (Béziau 2005). Research areas that were initially connected

[†] This work was developed over the course of five years, during which the author was supported by Ph.D. scholarships from Jacobs University Bremen, the German Academic Exchange Service and the German Merit Foundation. Additional support was received from research grants by the American National Science Foundation (Logosphere grant, number CCR-ITR-0325808) and the German Research Council (LATIN grant, number KO-2428/9-1), and from IT University Copenhagen.

have diverged and evolved into separate fields, and while this specialisation has led to very successful results, it has also created divisions in the research on logic that are sometimes detrimental.

In response to these divisions, logical frameworks have been introduced. They unify different logical formalisms by representing them in a fixed meta-language. Logical frameworks have been used successfully for logic-independent investigations on both the theoretical (for example, in the textbook Diaconescu (2008)) and practical levels (for example, in the proof assistant Isabelle (Paulson 1994)).

However, today we observe that one division remains, and that there are two groups of logical frameworks: *model-theoretical* and *proof-theoretical* ones. While some of these frameworks integrate aspects from the other side, such as the general proof theory developed in Meseguer (1989), almost all of them lean towards either model or proof theory. Often these sides are divided not only by research questions, but also by 'conflicting cultures and attitudes', and 'attempts to bridge these two cultures are rare and rather timid' (quoting an anonymous reviewer of a related paper). This paper 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.

Model-theoretical frameworks are based on mathematical foundations in set theory (for example, Zermelo–Fraenkel set theory (Zermelo 1908; Fraenkel 1922)) or category theory (Mac Lane 1998), and characterise logics model-theoretically in a way that goes back to Tarski's view of logical consequence (Tarski 1933; Tarski and Vaught 1956; Robinson 1950). The central concepts are a *model*, which interprets the non-logical symbols, and the *satisfaction* relation between formulas and models. Category theory and initial models (Goguen *et al.* 1978) are often employed to study the model classes. The most important such framework is that of institutions (Goguen and Burstall 1992).

On the other hand, *proof-theoretical frameworks* are based on mathematical foundations in type theory (for example, the *principia* (Whitehead and Russell 1913) or simple type theory (Church 1940)), and characterise logics proof-theoretically (see, for example, Gödel (1930) and Gentzen (1934)) in a way most prominently expressed in Hilbert's program (Hilbert 1926). The central concept is that of a *proof*, which derives valid *judgments* from axioms using inference rules. The Curry–Howard correspondence (Curry and Feys 1958; Howard 1980) is often employed to study proofs as expressions of a formal language. The most important such frameworks are Automath (de Bruijn 1970), Isabelle (Paulson 1994) and the Edinburgh Logical Framework (LF) (Harper *et al.* 1993).

The ontological intersection of model and proof theory is the syntax of logical formulas and the *consequence* relation between formulas, which includes the study of *theorems*. Thus, both provide ontological frameworks for formulas and consequence, and this intersection will be our central interest in the current paper. But both frameworks also go beyond this intersection. Model theory studies the properties of models in general, such as categoricity and initiality. Similarly, proof theory studies the properties of proofs in general, such as normalisation and program extraction. Our work may provide a starting point for investigating whether these advanced properties of one field have interesting analogues in the other.

Our unified framework picks one of the most successful frameworks from each side and combines them: *viz.* institutions and LF.

Institutions provide an abstract definition of the syntax and model-theoretical semantics of a logic. Research on institutions focuses on signatures, sentences, models and satisfaction, but de-emphasises proofs and derivability. Among the most characteristic features of the framework of institutions are:

(i) It abstracts from the syntax of formulas and only assumes an abstract set of sentences; similarly, it uses an abstract class of models and an abstract satisfaction relation.

(ii) Using a Galois connection between sentences and models, it allows us to view the relation between syntax and (model-theoretical) semantics as a pair of adjoint functors (Lawvere 1969).

(iii) Using category theory (Mac Lane 1998), it provides an abstract notion of translations: both within a logic, where they are called signature or theory morphisms, and between logics, where they are called institution (co)morphisms (Goguen and Rosu 2002; Mossakowski *et al.* 2007).

LF is a dependent type theory related to Martin-Löf's type theory (Martin-Löf 1974) featuring kinded type families and dependent function types. Research on logic encodings in LF focuses on syntax, proofs and provability, and, dually, de-emphasises models and satisfaction. Among the most characteristic features of this framework are:

(i) It represents the logical and non-logical symbols as constants of the type theory using higher-order abstract syntax to represent binders.

(ii) Using the judgments-as-types paradigm (Martin-Löf 1996), it allows us to view the syntax and (proof-theoretical) semantics using judgments and inference systems, and thus without appealing to a foundation of mathematics.

(iii) Using a constructive point of view, translation functions are given as provably terminating programs (see, for example, Naumov *et al.* (2001)).

Both of these frameworks have been strongly influenced by the respective approaches, and this has created complementary strengths and weaknesses. For example, the bias towards an abstraction from the syntax exhibited by institutions is very useful for a logic-independent meta-logical analysis (Diaconescu 2008). But LF's bias towards concrete syntax excels when defining individual logics and extracting programs from individual translations (Pfenning 2001). Therefore, we will focus on combining the respective strengths and motivations: while our work is technically formulated within institution theory, it inherits a strictly formalist justification from LF.

In the literature on individual logics, the combination of proof- and model-theoretical perspective is commonplace, and logics are usually introduced by giving their syntax, model theory and proof theory. The major difference is often the order of the latter two. Sometimes the model theory is given primacy as *the* semantics, and then a proof theory is given and proved sound and complete (see, for example, Barwise (1977) and Smullyan (1995)). And sometimes it is the other way round (see, for example, Andrews (1986) and Lambek and Scott (1986)). This difference often reflects the philosophical inclination of the author.

A similar difference in primacy is found when looking at families of logics. For example, the model-theoretical view takes precedence in description logic (see, for example, Brachman and Schmolze (1985) and Baader *et al.* (2003)), where the model theory is fixed and proof theory is studied chiefly to provide reasoning tools. This view is arguably paramount as it dominates accounts of (classical) first-order logic and most uses of logic in mathematics. On the other hand, the proof-theoretical view takes precedence in, for example, higher-order logic (see Church (1940) for proof theory and Henkin (1950) for model theory); and some logics such as intuitionistic logics (Brouwer 1907) are explicitly defined by their proof theory.

At the level of logical frameworks, proof-theoretical principles have been integrated into model-theoretical frameworks in several ways. Parchments are used in Goguen and Burstall (1986) and Mossakowski *et al.* (1997) to express the syntax of a logic as a formal language. And, for example, in Meseguer (1989), Fiadeiro and Sernadas (1988) and Mossakowski *et al.* (2005), proof theory is formalised in the abstract style of category theory. These approaches are very elegant, but often fail to exploit a major advantage of proof theory – the constructive reasoning over concrete syntax.

Model-theoretical principles have also been integrated into proof-theoretical frameworks, albeit to a lesser extent. For example, models and model classes can be represented using axiomatic type classes in Isabelle (Paulson 1994; Haftmann and Wenzel 2006) or using structures in Mizar (Trybulec and Blair 1985).

However, no logical framework has so far systematically subsumed frameworks from both perspectives, and our contribution is to provide such a framework. Our work is separated into three main parts:

(1) In Section 3, we extend institutions with an abstract notion of proof theory, arriving at what we call *logics*. Our logics are defined in the spirit of institutions, in particular, they retain the abstraction from concrete syntax. Thus, they are very similar to the general logics of Meseguer (1989) and to the proof-theoretic institutions of Mossakowski *et al.* (2005) and Diaconescu (2006). We also discuss the use of meta-logics, which can be seen as a response to Tarlecki (1996), where the use of a meta-institution is explored in general, and the use of LF suggested in particular.

(2) In Section 4, we give a logic $\mathbb{M}$ based on LF, which we will use as our meta-logic. The central idea is that a signature of $\mathbb{M}$ captures the syntax, proof theory and model theory of an object logic. Similarly, $\mathbb{M}$-signature morphisms capture translations between object logics, including the translations of syntax, proof theory and model theory. $\mathbb{M}$ is defined in the spirit of LF, in particular, it retains the focus on concrete syntax. Consequently, judgments are represented as types and proofs as terms. Moreover, inspired by the ideas of Lawvere (Lawvere 1963), models are represented as signature morphisms from the logical theory into a signature representing the foundation of mathematics.

In this paper, we will use the term 'foundation of mathematics' to refer to a fixed language in which mathematical objects are expressed. While mathematicians usually do not specify a foundation explicitly (often implicitly assuming a variant of first-order set theory as the foundation), the choice of foundation is more difficult in computational logic, where, for example, higher-order logic (Church 1940) and the calculus of constructions (Coquand and Huet 1988) are often used instead. By representing such a foundation as an LF signature itself, our framework can formulate model theory without committing to a specific foundation.

(3) In Section 5, we demonstrate how to use $\mathbb{M}$ as a meta-logic. We also show how existing logics and translations can be encoded in $\mathbb{M}$ and how we can reason about the adequacy of such encodings. This section reconciles the type/proof and set/model-theoretical perspectives: institution-based logics are expressed using the syntax of an LF-based meta-logic.

We will exemplify all our definitions by giving a simple logic translation from modal logic to first-order logic as a running example. Moreover, we have implemented our framework and evaluated it in a number of large-scale case studies (Horozal and Rabe 2011; Iancu and Rabe 2011; Codescu *et al.* 2011).

We discuss and evaluate our framework in Section 6 and give our conclusions in Section 7. However, we will begin in Section 2 by introducing the frameworks of institutions and LF.

2. Logical frameworks

2.1. *The model-theoretical framework of institutions*

Institutions were introduced in Goguen and Burstall (1992) as a means of managing the population explosion among logics in use. The key idea is to abstract from the satisfaction relation between the sentences and the models. We will only give an intuitive introduction to the notion of an institution here. A rigorous and more comprehensive treatment is inherent in our definitions in Section 3.

The components of an institution are centred around a class **Sig** of *signatures*. For each signature $\Sigma \in$ **Sig**, there are an associated set $\mathbf{Sen}(\Sigma)$ of *sentences* and class $\mathbf{Mod}(\Sigma)$ of *models*, and a *satisfaction* relation $\models_\Sigma \subseteq \mathbf{Mod}(\Sigma) \times \mathbf{Sen}(\Sigma)$.

Example 2.1 (modal logic). For a simple institution $\mathbb{ML}$ of modal logic, the signatures in $\mathbf{Sig}^{\mathbb{ML}}$ are the finite sets of propositional variables (chosen from some fixed set of symbols). The set $\mathbf{Sen}^{\mathbb{ML}}(\Sigma)$ is the smallest set containing p for all $p \in \Sigma$ and closed under sentence formation $F \supset G$ and $\Box F$.

A model $M \in \mathbf{Mod}^{\mathbb{ML}}(\Sigma)$ is a Kripke model, that is, a tuple $(W, \prec, V)$ for a set W of worlds, an accessibility relation $\prec \subseteq W \times W$ and a valuation $V : \Sigma \times W \to \{0, 1\}$ for the propositional variables. V is extended to a mapping $\mathbf{Sen}(\Sigma) \times W \to \{0, 1\}$ in the usual way, and the satisfaction relation $M \models^{\mathbb{ML}}_\Sigma F$ holds if and only if $V(F, w) = 1$ for all worlds $w \in W$.

This abstract definition yields a rich setting in which institutions can be studied (see, for example, Diaconescu (2008)). Most importantly, we can define an abstract notion of a *theory* as a pair (Σ, Θ) for a set $\Theta \subseteq \mathbf{Sen}(\Sigma)$ of axioms. (See Terminology 3.17 for different uses of the term 'theory'.) A sentence $F \in \mathbf{Sen}(\Sigma)$ is a (Σ, Θ)-*theorem* if and only if for all models $M \in \mathbf{Mod}(\Sigma)$ we have $M \models_\Sigma A$ for all $A \in \Theta$ implies $M \models_\Sigma F$. This is the model-theoretical consequence relation and written as $\Theta \models_\Sigma F$.

The above concepts have a second dimension when all classes are extended to categories (Mac Lane 1998). **Sig** is in fact a category and **Sen** a functor $\mathbf{Sig} \to \mathcal{SET}$. Similarly, **Mod** is a functor $\mathbf{Sig} \to \mathcal{CAT}^{op}$ (see also Notation 3.3). Signature morphisms represent translations between signatures, and the actions of **Sen** and **Mod** on a signature morphism extend these translations to sentences and models. Sentence and model translations go in opposite directions, which corresponds to an adjunction between syntax and semantics. Finally, $\models_\Sigma$ is subject to the satisfaction condition, which guarantees that the satisfaction of a sentence in a model is invariant under signature morphisms.

Example 2.2 (modal logic continued). In the case of modal logic, signature morphisms $\sigma : \Sigma \to \Sigma'$ are substitutions of Σ'-sentences for the propositional variables in Σ, and

$$\mathbf{Sen}^{\mathbb{ML}}(\sigma) : \mathbf{Sen}^{\mathbb{ML}}(\Sigma) \to \mathbf{Sen}^{\mathbb{ML}}(\Sigma')$$

is the induced homomorphic extension. The functor $\mathbf{Mod}^{\mathbb{ML}}(\sigma)$ reduces a model

$$(W, \prec, V') \in \mathbf{Mod}^{\mathbb{ML}}(\Sigma')$$

to the model

$$(W, \prec, V) \in \mathbf{Mod}^{\mathbb{ML}}(\Sigma)$$

by putting

$$V(p, w) = V'(\sigma(p), w).$$

A *theory morphism* $\sigma : (\Sigma, \Theta) \to (\Sigma', \Theta')$ is a signature morphism $\sigma : \Sigma \to \Sigma'$ such that for all axioms $A \in \Theta$, we have that $\mathbf{Sen}(\sigma)(A)$ is a (Σ', Θ')-theorem. This implies that $\mathbf{Sen}(\sigma)$ maps (Σ, Θ)-theorems to (Σ', Θ')-theorems, that is, theory morphisms preserve truth. Adjointly, one can show that σ is a theory morphism if and only if $\mathbf{Mod}(\sigma)$ maps (Σ', Θ')-models to (Σ, Θ)-models.

Institutions can be described elegantly as functors out of a signature category into a fixed category C of *rooms*. This yields a notion of institution translations via the well-known notion of the lax slice category of functors into C. Such translations are called institution *comorphisms* (Meseguer 1989; Tarlecki 1996; Goguen and Rosu 2002). This is the basis of a number of logic translations that have been expressed as translations between institutions (see, for example, Mossakowski *et al.* (2007) for examples).

A comorphism from $\mathbb{I}$ to $\mathbb{I}'$ translates signatures and sentences from $\mathbb{I}$ to $\mathbb{I}'$ and models in the opposite direction. The sentence translation preserves the (model-theoretical) consequence relation $\Theta \models_\Sigma F$. If the model translations are surjective (this is called the model expansion property), the consequence relation is also reflected. This is the basis of the important borrowing application (Cerioli and Meseguer 1997), where the consequence relation of one institution is reduced to that of another, for example, one for which an implementation is available.

In fact, very few logic translations can be expressed as comorphisms in this sense – instead, it is often necessary to translate $\mathbb{I}$-signatures to $\mathbb{I}'$-theories. However, such translations can be represented as institution comorphisms from $\mathbb{I}$ to $Th(\mathbb{I}')$. Here $Th(\mathbb{I}')$

is a new institution whose signatures are the theories of $\mathbb{I}'$ – this is a straightforward construction, and is possible for all institutions.

2.2. *The proof-theoretical framework LF*

LF (Harper *et al.* 1993) is a dependent type theory related to Martin-Löf type theory (Martin-Löf 1974). It is the corner of the λ-cube (Barendregt 1992) that extends simple type theory with dependent function types. We will work with the Twelf implementation of LF (Pfenning and Schürmann 1999).

The main use of LF and Twelf is as a *logical framework* in which deductive systems are represented. Typically, *kinded type families* are declared to represent the syntactic classes of the system. For example, to represent higher-order logic, we use a signature HOL with declarations:

$$\begin{array}{lcl} tp & : & \mathtt{type} \\ tm & : & tp \rightarrow \mathtt{type} \\ bool & : & tp \\ ded & : & tm\ bool \rightarrow \mathtt{type}\,. \end{array}$$

Here `type` is the LF-kind of types, and tp is an LF-type whose LF-terms represent the STT-types. $tp \rightarrow$ `type` is the kind of type families, which are indexed by terms of LF-type tp. Then $tm\ A$ is the LF-type whose terms represent the STT-terms of type A. For example, $bool$ represents the HOL-type of propositions such that HOL-propositions are represented as LF-terms of type $tm\ bool$. LF employs the Curry–Howard correspondence to represent proofs-as-term (Curry and Feys 1958; Howard 1980) and extends it to the *judgments-as-types* methodology (Martin-Löf 1996): ded declares the truth judgment on HOL-propositions, and the type $ded\ F$ represents the judgment that F is derivable. HOL-derivations of F are represented as LF-terms of type $ded\ F$, and F is provable in HOL if and only if there is an LF term of type $ded\ F$.

Additionally, *typed constants* are declared to construct expressions of these syntactic classes, that is, types (for example, $bool$ above), terms, propositions and derivations of HOL. For example, consider

$$\begin{array}{lcl} \Rightarrow & : & tm\ bool \rightarrow tm\ bool \rightarrow tm\ bool \\ \Rightarrow_E & : & \{F : bool\}\ \{G : bool\}\ \ ded\ (F \Rightarrow G)\ \ \rightarrow\ \ ded\ F\ \ \rightarrow\ \ ded\ G \\ \forall & : & \{A : tp\}\ \ (tm\ A \rightarrow bool)\ \ \rightarrow\ \ bool\,. \end{array}$$

Here $\Rightarrow$ encodes implication as a binary proposition constructor. $\Rightarrow_E$ encodes the implication elimination rule (modus ponens): it first takes two propositions F and G as arguments, and then two derivations of $F \Rightarrow G$ and F, respectively, and returns a derivation of G. Note that the types of the latter arguments depend on the values F and G of the first two arguments. The encoding of the universal quantifier $\forall$ uses *higher-order abstract syntax* to declare binders as constants: LF terms of type $S \rightarrow T$ are in bijection to LF terms of type T with a free variable of type S. Thus, $\forall$ encodes a proposition constructor that takes a HOL type A and a proposition with a free variable of type $tm\ A$.

We will always use Twelf notation for the LF primitives of binding and application: the type $\Pi_{x:A}B(x)$ of dependent functions taking $x : A$ to an element of $B(x)$ is written

$\{x : A\} B\ x$, and the function term $\lambda_{x:A} t(x)$ taking $x : A$ to $t(x)$ is written $[x : A]\, t\ x$. As usual, we write $A \to B$ instead of $\{x : A\} B$ if x does not occur in B.

This yields the following grammar for the three levels of LF expressions:

$$\begin{array}{lll} \text{Kinds:} & K & ::= \mathtt{type} \mid \{x : A\} K \\ \text{Type families:} & A, B & ::= a \mid A\ t \mid [x : A]\, B \mid \{x : A\} B \\ \text{Terms:} & s, t & ::= c \mid x \mid [x : A]\, t \mid s\ t. \end{array}$$

$\{x : A\} K$ and $\{x : A\} B$ are used to abstract over variables occurring in kinds and types, respectively. Similarly, there are two λ-abstractions and two applications. The three productions for kind-level abstraction are set in grey type.

The *judgments* are $\Gamma \vdash_\Sigma K\ \mathtt{kind}$ for well-formed kinds, $\Gamma \vdash_\Sigma A : K$ for well-kinded type families, and $\Gamma \vdash_\Sigma s : A$ for well-typed terms. They are defined relative to a signature Σ and a context Γ, which declare the symbols and variables, respectively, that can occur in expressions. Moreover, all levels come with an *equality* judgment, and both abstractions respect α, β and η-conversion. All judgments are decidable (Harper *et al.* 1993).

The grammar for signatures and contexts and their translations is as follows:

$$\begin{array}{llll} \text{Signatures} & \Sigma & ::= & \cdot \mid \Sigma,\ c : A \mid \Sigma,\ a : K, \\ \text{Morphisms} & \sigma & ::= & \cdot \mid \sigma,\ c := t \mid \Sigma,\ a := A \\ \text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma,\ x : A \\ \text{Substitutions} & \sigma & ::= & \cdot \mid \gamma,\ x := t. \end{array}$$

An LF *signature* Σ holds the globally available declared names; it is a list of kinded type family declarations $a : K$ and typed constant declarations $c : A$. Similarly, a Σ-*context* holds the locally available declared names; it is a list of typed variable declarations $x : A$.

Both signatures and contexts come with a notion of homomorphic translations. Given two signatures Σ and Σ', a *signature morphism* $\sigma : \Sigma \to \Sigma'$ is a typing- and kinding-preserving map of Σ-symbols to closed Σ'-expressions. Thus, σ maps every constant $c : A$ of Σ to a term $\sigma(c) : \overline{\sigma}(A)$ and every type family symbol $a : K$ to a type family $\sigma(a) : \overline{\sigma}(K)$. Here, $\overline{\sigma}$ is the homomorphic extension of σ to all closed Σ-expressions, and we will write σ instead of $\overline{\sigma}$ from now on.

Signature morphisms preserve typing, kinding and equality of expressions, in other words, if $\cdot \vdash_\Sigma E : F$, then $\cdot \vdash_{\Sigma'} \sigma(E) : \sigma(F)$, and similarly for equality. In particular, because σ must map all axioms or inference rules declared in Σ to proofs or derived inference rules, respectively, over Σ', signature morphisms preserve the provability of judgments. Two signature morphisms are equal if they map the same constants to $\alpha\beta\eta$-equal expressions. Up to this equality, signatures and signature morphisms form a category, which we denote by $\mathbb{LF}$.

$\mathbb{LF}$ has inclusion morphisms $\Sigma \hookrightarrow \Sigma, \Sigma'$ and pushouts along inclusions (Harper *et al.* 1994). Moreover, a coherent system of pushouts along inclusions can be chosen canonically: given $\sigma : \Sigma \to \Sigma'$ and an inclusion $\Sigma \hookrightarrow \Sigma,\ c : A$, the canonical *pushout* is given by

$$\sigma,\ c := c \quad : \quad \Sigma,\ c : A \quad \to \quad \Sigma',\ c : \sigma(A)$$

(except for possibly renaming c if it is not fresh for Σ'). The canonical choices for pushouts along other inclusions are obtained accordingly.

Similarly, given two Σ-contexts Γ and Γ', a *substitution* $\gamma : \Gamma \to \Gamma'$ is a typing-preserving map of Γ-variables to Γ'-expressions (which leaves the symbols of Σ fixed). Thus, σ maps every variable $x : A$ of Γ to a term $\gamma(x) : \overline{\gamma}(A)$. Again, $\overline{\gamma}$ is the homomorphic extension of γ, and we write γ instead of $\overline{\gamma}$. Like signature morphisms, substitutions preserve typing, kinding and equality of expressions in contexts, for example, if $\Gamma \vdash_\Sigma E : F$, then $\Gamma' \vdash_\Sigma \gamma(E) : \gamma(F)$. Two substitutions are equal if they map the same variables to $\alpha\beta\eta$-equal terms. Up to this equality, for a fixed signature Σ, the contexts over Σ and the substitutions between them form a category.

Finally, a signature morphism $\sigma : \Sigma \to \Sigma'$ can be applied component-wise to contexts and substitutions:

— $\sigma(\cdot) = \cdot$ and $\sigma(\Gamma, x : A) = \sigma(\Gamma), x : \sigma(A)$ for contexts;
— $\sigma(\cdot) = \cdot$ and $\sigma(\gamma, x := t) = \sigma(\gamma), x := \sigma(t)$ for substitutions.

We have the invariant that if $\gamma : \Gamma \to \Gamma'$ over Σ and $\sigma : \Sigma \to \Sigma'$, then $\sigma(\gamma) : \sigma(\Gamma) \to \sigma(\Gamma')$. Categorically, $\sigma(-)$ is a functor from the category of Σ-contexts to the category of Σ'-contexts.

3. A comprehensive logical framework

Our primary objective is to encode logics and logic translations in an LF-based meta-logic. Therefore, we must do three things:

(i) define what logics and logic translations are – see Section 3.1 and 3.2, respectively;
(ii) define what it means to encode them in a meta-logic – see Section 3.3;
(iii) give the LF-based meta-logic – see Section 4.

Note that (ii) and (iii) could also be given the other way round. We give (ii) first for an arbitrary meta-logic so that it is possible to substitute other meta-logics later.

3.1. *Logics*

3.1.1. *Model theories and institutions.* In order to give a unified approach to model and proof theory, we will refactor the definition of institutions by splitting an institution into syntax and model theory.

Definition 3.1 (logic syntax). A *logic syntax* is a pair $(\mathbf{Sig}, \mathbf{Sen})$ such that $\mathbf{Sig}$ is a category and $\mathbf{Sen} : \mathbf{Sig} \to \mathcal{SET}$ is a functor.

Definition 3.2 (model theory). For a logic syntax $(\mathbf{Sig}, \mathbf{Sen})$, a *model theory* is a tuple $(\mathbf{Mod}, \models)$ such that $\mathbf{Mod} : \mathbf{Sig} \to \mathcal{CAT}^{op}$ is a functor and $\models_\Sigma \subseteq \mathbf{Sen}(\Sigma) \times |\mathbf{Mod}(\Sigma)|$ is family of relations such that the satisfaction condition holds: for all $\sigma : \Sigma \to \Sigma'$, $F \in \mathbf{Sen}(\Sigma)$ and $M' \in |\mathbf{Mod}(\Sigma')|$, we have $\mathbf{Mod}(\sigma)(M') \models_\Sigma F$ if and only if $M' \models_{\Sigma'} \mathbf{Sen}(\sigma)(F)$.

In particular, institutions are exactly the pairs of a logic syntax and a model theory for it.

Notation 3.3. In the literature, the model theory functor is usually given as $\mathbf{Mod} : \mathbf{Sig}^{op} \to \mathcal{CAT}$. We choose the dual orientation here in order to emphasise the symmetry between model and proof theory later on.

Example 3.4 (modal logic continued). $(\mathbf{Sig}^{\mathbb{ML}}, \mathbf{Sen}^{\mathbb{ML}})$ and $(\mathbf{Mod}^{\mathbb{ML}}, \models^{\mathbb{ML}})$ from Example 2.1 are examples of a logic syntax and a model theory for it. To complete the example, we have to show the satisfaction condition. We assume $\sigma : \Sigma \to \Sigma'$, $F \in \mathbf{Sen}^{\mathbb{ML}}(\Sigma)$ and $(W, \prec, V) \in \mathbf{Mod}^{\mathbb{ML}}(\Sigma')$. The satisfaction condition then follows if we can show that $V(F, w) = V'(\mathbf{Sen}^{\mathbb{ML}}(\sigma)(F), w)$, but this is easy to see.

Note that we have proved that σ preserves truth in all worlds, whereas the satisfaction condition only requires the weaker condition that $\mathbf{Sen}^{\mathbb{ML}}(\sigma)(F)$ holds in all worlds of $(W, \prec, V)$ if and only if F holds in all worlds of $(W, \prec, V')$. This extension of the satisfaction condition to all components of syntax and model theory is typical, but different for each institution. A uniform formulation was given in Aiguier and Diaconescu (2007) using stratified institutions.

3.1.2. *Proof theories and logics.*

As in the case of model theories, for a given logic syntax, we will define proof theories as pairs $(\mathbf{Pf}, \vdash)$. First, we define proof categories – the values of the $\mathbf{Pf}$ functor.

Definition 3.5 (proof categories). A *proof category* is a category P with finite products (including the empty product). $\mathcal{PFCAT}$ is the category of proof categories together with the functors preserving finite products.

Notation 3.6. If a category has the product $(F_1, \ldots, F_n)$, we write it as $(F_i)_1^n$, or simply F_1^n. Similarly, for $p_i : E \to F_i$, we write p_1^n for the universal morphism $(p_1, \ldots, p_n) : E \to F_1^n$.

We think of the objects of P as *judgments* about the syntax that can be assumed and derived. The intuition of a morphism p_1^n from E_1^m to F_1^n is that it provides *evidence* p_i for each judgment F_i under the assumptions $E_1, \ldots, E_m$.

Example 3.7 (modal logic continued). We obtain a proof theory functor $\mathbf{Pf}^{\mathbb{ML}} : \mathbf{Sig}^{\mathbb{ML}} \to \mathcal{PFCAT}$ as follows. The proof category $\mathbf{Pf}^{\mathbb{ML}}(\Sigma)$ is the category of finite families F_1^n for $F_i \in \mathbf{Sen}^{\mathbb{ML}}(\Sigma)$. The morphisms in $\mathbf{Pf}^{\mathbb{ML}}(\Sigma)$ from E_1^m to F_1^n are the families p_1^n such that every p_i is a proof of F_i from assumptions $E_1, \ldots, E_m$. We can choose any calculus to make the notion of proofs precise, for example, a Hilbert calculus with rules for modus ponens and necessitation.

$\mathbf{Pf}^{\mathbb{ML}}(\Sigma)$ has finite products by taking products of families. The action of $\mathbf{Pf}^{\mathbb{ML}}$ on signature morphisms $\sigma : \Sigma \to \Sigma'$ is induced in the obvious way.

Remark 3.8 (proof categories). By assuming that proof categories have products in Definition 3.5, we are implicitly assuming the rules of weakening (given by morphisms from (A, B) to A), contraction (given by morphisms from A to (A, A)), and exchange (given by morphisms from (A, B) to (B, A)). A substructural framework, for example, one akin to linear logic (Girard 1987), could be obtained by weakening the condition on the existence of products.

We only require finite products because deductive systems typically use words over countable alphabets, in which only finite products of judgments can be expressed. This amounts to a restriction to compact deductive systems, which corresponds to the constructive flavour of proof theory.

It is natural to strengthen the definition by further assuming that all objects of a proof category P can be written as products of irreducible objects, in which case proof categories become many-sorted Lawvere categories (Lawvere 1963). Then, up to isomorphism, the judgments can be seen as the finite multi-sets of irreducible objects, and canonical products are obtained by taking unions of multi-sets. This is the typical situation, but we do not assume it here because we will not make use of this additional assumption.

An even stronger definition could require that the irreducible objects of $\mathbf{Pf}\,(\Sigma)$ are just the sentences in $\mathbf{Sen}(\Sigma)$. This is the case in Example 3.7, where the irreducible objects are the singleton multi-sets, and thus essentially the sentences. But it is desirable to avoid this assumption in order to include deductive systems where auxiliary syntax is introduced, such as hypothetical judgments (as used in natural deduction – see Example 3.9) or signed formulas (as used in tableaux calculi).

Example 3.9. The proof theory of the first-order natural deduction calculus arises as follows. For a $\mathbb{FOL}$-signature Σ, an atomic Σ-judgment is of the form $\Gamma;\Delta \vdash F$ where:

— Γ is a context declaring some free variables that may occur in Δ and F.
— Δ is a list of Σ-formulas.
— F is a Σ-formula.

The objects of $\mathbf{Pf}^{\,\mathbb{FOL}}(\Sigma)$ are the multi-sets of atomic Σ-judgments, and products are given by unions of multi-sets. A morphism in $\mathbf{Pf}^{\,\mathbb{FOL}}(\Sigma)$ from J_1^m to a singleton multi-set (K) is a natural deduction proof tree whose root is labelled with K and whose leaves are labelled with one element of $J_1, \ldots, J_m$. Morphisms with other codomains are obtained from the universal property of the product.

Giving a proof category $\mathbf{Pf}\,(\Sigma)$ for a signature Σ is not sufficient to define a proof-theoretical semantics of Σ – we also have to relate the Σ sentences and the Σ-judgments. To do this, we use a map $\vdash_\Sigma : \mathbf{Sen}(\Sigma) \to \mathbf{Pf}\,(\Sigma)$ assigning to every sentence its truth judgment. In particular, $\mathbf{Pf}\,(\Sigma)$-morphisms from the empty family () to $\vdash_\Sigma F$ represent the proofs of F. Corresponding to the satisfaction condition, the truth judgment should be preserved under signature morphisms. This yields the following definition.

Definition 3.10 (proof theory). For a logic syntax $(\mathbf{Sig}, \mathbf{Sen})$, a *proof theory* is a tuple $(\mathbf{Pf}\,, \vdash)$ such that $\mathbf{Pf}\; : \mathbf{Sig} \to \mathcal{PFCAT}$ is a functor and $\vdash_\Sigma$ is a mapping from $\mathbf{Sen}(\Sigma)$ to $\mathbf{Pf}\,(\Sigma)$ such that for all $\sigma : \Sigma \to \Sigma'$ and $F \in \mathbf{Sen}(\Sigma)$, we have $\mathbf{Pf}\,(\sigma)(\vdash_\Sigma F) = \vdash_{\Sigma'} \mathbf{Sen}(\sigma)(F)$.

Finally, we can put together syntax, model theory and proof theory to form a logic.

Definition 3.11 (logics). A *logic* is a tuple $\mathbb{I} = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models, \mathbf{Pf}\,, \vdash)$ such that $(\mathbf{Sig}, \mathbf{Sen})$ is a logic syntax and $(\mathbf{Mod}, \models)$ and $(\mathbf{Pf}\,, \vdash)$ are a model theory and a proof theory, respectively, for it.

Example 3.12 (modal logic continued). We obtain a proof theory and thus a logic (in our formal sense) for modal logic by using $\mathbf{Pf}^{\,\mathbb{ML}}$ as in Example 3.7 and putting $\vdash_\Sigma^{\mathbb{ML}} F = (F)$ where (F) is the family containing a single element F. Indeed, we have

$$\mathbf{Pf}^{\,\mathbb{ML}}(\sigma)(\vdash_\Sigma^{\mathbb{ML}} F) = (\mathbf{Sen}^{\mathbb{ML}}(\sigma)(F)) = \vdash_{\Sigma'}^{\mathbb{ML}} \mathbf{Sen}^{\mathbb{ML}}(\sigma)(F).$$

Thus, a logic consists of a category of signatures **Sig**, a sentence functor **Sen**, a model theory functor **Mod**, a proof theory functor **Pf** , and model and proof-theoretical definitions of truth $\models$ and $\vdash$, respectively. This is depicted below for a signature morphism $\sigma : \Sigma \to \Sigma'$:

$$\begin{tikzcd}
\mathbf{Mod}(\Sigma) \arrow[d, dash, "\models_\Sigma"'] & \mathbf{Mod}(\Sigma') \arrow[l, "\mathbf{Mod}(\sigma)"'] \arrow[d, dash, "\models_{\Sigma'}"] & \mathcal{CAT} \\
\mathbf{Sen}(\Sigma) \arrow[r, "\mathbf{Sen}(\sigma)"] \arrow[d, "\vdash_\Sigma"'] & \mathbf{Sen}(\Sigma') \arrow[d, "\vdash_{\Sigma'}"] & \mathcal{SET} \\
\mathbf{Pf}\,(\Sigma) \arrow[r, "\mathbf{Pf}\,(\sigma)"] & \mathbf{Pf}\,(\Sigma') & \mathcal{PFCAT}
\end{tikzcd}$$

Technically, this is not a diagram in the sense of category theory. But if we treat the sets $\mathbf{Sen}(\Sigma)$ and $\mathbf{Sen}(\Sigma')$ as discrete categories, the lower half is a commuting diagram of categories. Moreover, if we forget morphisms and treat functors as special cases of relations between classes, then the upper half is a commutative diagram in the category of classes and relations. This is made more precise in the following remark.

Remark 3.13. The definition of logics can be phrased more categorically, which also makes the formal symmetry between model and proof theory more apparent. Let $\mathcal{CLASS}$ and $\mathcal{REL}$ be the categories of classes with mappings and relations, respectively, and let $|-| : \mathcal{CAT}^{op} \to \mathcal{CLASS}$ and $|-|^r : \mathcal{CAT}^{op} \to \mathcal{REL}$ be the functors forgetting morphisms. Let us also identify $\mathcal{SET}$ with its two inclusions into $\mathcal{REL}$ and $\mathcal{PFCAT}$. Then $\models$ and $\vdash$ are natural transformations $\models : \mathbf{Sen} \to |-|^r \circ \mathbf{Mod}$ and $\vdash : \mathbf{Sen} \to |-| \circ \mathbf{Pf}$.

Then a logic $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models, \mathbf{Pf}\,, \vdash)$ yields the following diagram in the category $\mathcal{CAT}$, where double arrows indicate natural transformations between the functors making up the surrounding rectangles:

$$\begin{tikzcd}
 & \mathcal{CAT}^{op} \arrow[r, "|-|^r"] & \mathcal{REL} \\
\mathbf{Sig} \arrow[ru, "\mathbf{Mod}"] \arrow[r, "\mathbf{Sen}"] \arrow[rd, "\mathbf{Pf}"'] & \mathcal{SET} \arrow[u, Rightarrow, "\models"'] \arrow[d, Rightarrow, "\vdash"] \arrow[ru, hook] \arrow[rd, hook] & \\
 & \mathcal{PFCAT} \arrow[r, "|-|"'] & \mathcal{CLASS}
\end{tikzcd}$$

Remark 3.14. Even more categorically than in Remark 3.13, we can assume

$$U : \mathcal{SET} \to \mathcal{REL} \times \mathcal{CLASS}$$

maps a set S to (S, S), and

$$V : \mathcal{CAT}^{op} \times \mathcal{PFCAT} \to \mathcal{REL} \times \mathcal{CLASS}$$

maps a pair of a model category and a proof category to itself, but forgetting morphisms. Then logics are the functors from some category of signatures **Sig** to the comma category $(U \downarrow V)$.

3.1.3. *Theories and consequence.* While the notion of theories only depends on a logic syntax, both the model-theoretical semantics and the proof-theoretical semantics induce one consequence relation each. As usual, we write $\models$ and $\vdash$ for the model and proof-theoretical consequence, respectively. Consequently, we obtain two notions of theory morphisms.

Definition 3.15 (theories). Given a logic syntax $(\mathbf{Sig}, \mathbf{Sen})$, a *theory* is a pair (Σ, Θ) for $\Sigma \in \mathbf{Sig}$ and $\Theta \subseteq \mathbf{Sen}(\Sigma)$. The elements of Θ are called the *axioms* of (Σ, Θ).

Definition 3.16 (consequence). Assume a logic $\mathbb{I} = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models, \mathbf{Pf}, \vdash)$. For a theory (Σ, Θ) and a Σ-sentence F, we define *consequence* as follows:

— Θ *entails* F proof-theoretically, written $\Theta \vdash^{\mathbb{I}}_{\Sigma} F$, if and only if there exist a finite subset $\{F_1, \ldots, F_n\} \subseteq \Theta$ and a $\mathbf{Pf}(\Sigma)$-morphism from $(\vdash^{\mathbb{I}}_{\Sigma} F_i)_1^n$ to $\vdash^{\mathbb{I}}_{\Sigma} F$,

— Θ *entails* F model-theoretically, written $\Theta \models^{\mathbb{I}}_{\Sigma} F$, if and only if every model $M \in \mathbf{Mod}(\Sigma)$ satisfying all sentences in Θ also satisfies F.

We will drop the superscript $\mathbb{I}$ if it is clear from the context.

A signature morphism $\sigma : \Sigma \to \Sigma'$ is called a model-theoretical (or proof-theoretical) *theory morphism* from (Σ, Θ) to (Σ', Θ') if for all $F \in \Theta$, we have $\Theta' \models_{\Sigma'} \mathbf{Sen}(\sigma)(F)$ (or $\Theta' \vdash_{\Sigma'} \mathbf{Sen}(\sigma)(F)$ in the case of proof-theoretical theory morphisms).

Terminology 3.17. In the literature, the term *theory* is sometimes used only for a set of sentences that is closed under the consequence relation. In that case, theories in our sense are called *presentations*.

Theory morphisms preserve the respective consequence relation.

Lemma 3.18 (truth preservation). Given a model-theoretical or proof-theoretical theory morphism $\sigma : (\Sigma, \Theta) \to (\Sigma', \Theta')$ and a sentence $F \in \mathbf{Sen}(\Sigma)$, we have:

(i) $\Theta \models_{\Sigma} F$ implies $\Theta' \models_{\Sigma'} \mathbf{Sen}(\sigma)(F)$ for the model-theoretical case.
(ii) $\Theta \vdash_{\Sigma} F$ implies $\Theta' \vdash_{\Sigma'} \mathbf{Sen}(\sigma)(F)$ for the proof-theoretical case.

The well-known concepts of soundness and completeness relate the two consequence relations.

Definition 3.19 (soundness and completeness). A logic is *sound* if and only if $\varnothing \vdash_{\Sigma} F$ implies $\varnothing \models_{\Sigma} F$ for all signatures Σ and all sentences F. It is *strongly sound* if and only if $\Theta \vdash_{\Sigma} F$ implies $\Theta \models_{\Sigma} F$ for all theories (Σ, Θ) and all sentences F. Similarly, a logic is *strongly complete* or *complete* if the respective converse implication holds.

In particular, if a logic is strongly sound and strongly complete, then proof and model-theoretical consequence and theory morphisms coincide.

3.2. *Logic translations*

3.2.1. *Logic translations.* We will now define translations between two logics. Since our framework is based on institutions, we will build upon the existing notion of an institution *comorphism.*

Definition 3.20 (logic comorphism). Given two logics

$$\mathbb{I} = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models, \mathbf{Pf}, \vdash)$$
$$\mathbb{I}' = (\mathbf{Sig}', \mathbf{Sen}', \mathbf{Mod}', \models', \mathbf{Pf}', \vdash'),$$

a *logic comorphism* from $\mathbb{I}$ to $\mathbb{I}'$ is a tuple $(\Phi, \alpha, \beta, \gamma)$ consisting of a functor $\Phi : \mathbf{Sig} \to \mathbf{Sig}'$ and natural transformations

$$\alpha : \mathbf{Sen} \to \mathbf{Sen}' \circ \Phi$$
$$\beta : \mathbf{Mod} \to \mathbf{Mod}' \circ \Phi$$
$$\gamma : \mathbf{Pf} \to \mathbf{Pf}' \circ \Phi$$

such that

(1) for all $\Sigma \in \mathbf{Sig}$, $F \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}'(\Phi(\Sigma))$,

$$\beta_\Sigma(M') \models_\Sigma F \quad \text{iff} \quad M' \models'_{\Phi(\Sigma)} \alpha_\Sigma(F);$$

(2) for all $\Sigma \in \mathbf{Sig}$ and $F \in \mathbf{Sen}(\Sigma)$,

$$\gamma_\Sigma(\vdash_\Sigma F) \;=\; \vdash'_{\Phi(\Sigma)} \alpha_\Sigma(F).$$

With the obvious choices for identity and composition, we obtain the category $\mathcal{LOG}$ of logics and comorphisms.

Recall that β_Σ is a morphism in the category $\mathcal{CAT}^{op}$ and thus the same as a functor

$$\beta_\Sigma : \mathbf{Mod}'(\Phi(\Sigma)) \to \mathbf{Mod}(\Sigma),$$

so $\beta_\Sigma(M')$ is a well-formed functor application. Note that the syntax and the proof theory are translated from $\mathbb{I}$ to $\mathbb{I}'$, whereas the model theory is translated in the opposite direction. The two conditions on comorphisms are truth preservation conditions: The model and proof theory translations must preserve model and proof-theoretical truth, respectively.

In just the same way as logics can be decomposed into syntax, model theory and proof theory, a logic comorphism consists of:

— a syntax translation $(\Phi, \alpha) : (\mathbf{Sig}, \mathbf{Sen}) \to (\mathbf{Sig}', \mathbf{Sen}')$;
— for a given syntax translation (Φ, α), a model translation $\beta : (\mathbf{Mod}, \models) \to (\mathbf{Mod}', \models')$ which must satisfy condition (1);
— for a given syntax translation (Φ, α), a proof translation $\gamma : (\mathbf{Pf}, \vdash) \to (\mathbf{Pf}', \vdash')$ which must satisfy condition (2).

Remark 3.21. Continuing Remark 3.14, we observe that if logics are slices $\mathbb{I}_i : \mathbf{Sig}_i \to (U \downarrow V)$, then a logic comorphisms from $\mathbb{I}_1$ to $\mathbb{I}_2$ is a functor $\Phi : \mathbf{Sig}_1 \to \mathbf{Sig}_2$ together with a natural transformation from $\mathbb{I}_1$ to $\mathbb{I}_2 \circ \Phi$. In this way, $\mathcal{LOG}$ can be seen as the lax slice category of objects over $(U \downarrow V)$.

3.2.2. *Comorphism modifications.* The category of logics and logic comorphisms can be turned into a 2-category. Again we are inspired by the existing notion for institutions, for which the 2-cells are called institution *comorphism modifications* (Diaconescu 2002):

Definition 3.22 (modifications). Given two logic comorphisms

$$\mu^i = (\Phi^i, \alpha^i, \beta^i, \gamma^i) : \mathbb{I} \to \mathbb{I}'$$

for $i = 1, 2$, a *comorphism modification* from μ^1 to μ^2 is a natural transformation $m : \Phi_1 \to \Phi_2$ such that the following diagrams commute:

$$\begin{tikzcd}
 & \mathbf{Sen}'(\Phi^1(\Sigma)) \arrow[dd, "\mathbf{Sen}'(m_\Sigma)"] \\
\mathbf{Sen}(\Sigma) \arrow[ur, "\alpha^1_\Sigma"] \arrow[dr, "\alpha^2_\Sigma"'] & \\
 & \mathbf{Sen}'(\Phi^2(\Sigma))
\end{tikzcd}$$

$$\begin{tikzcd}
 & \mathbf{Mod}'(\Phi^1(\Sigma)) \arrow[dl, "\beta^1_\Sigma"'] \\
\mathbf{Mod}(\Sigma) & \\
 & \mathbf{Mod}'(\Phi^2(\Sigma)) \arrow[ul, "\beta^2_\Sigma"] \arrow[uu, "\mathbf{Mod}'(m_\Sigma)"']
\end{tikzcd}$$

$$\begin{tikzcd}
 & \mathbf{Pf}'(\Phi^1(\Sigma)) \arrow[dd, "\mathbf{Pf}'(m_\Sigma)"] \\
\mathbf{Pf}(\Sigma) \arrow[ur, "\gamma^1_\Sigma"] \arrow[dr, "\gamma^2_\Sigma"'] & \\
 & \mathbf{Pf}'(\Phi^2(\Sigma))
\end{tikzcd}$$

where the diagram for the model theory is drawn in $\mathcal{CAT}$.

Thus, a comorphism modification m is a family $(m_\Sigma)_{\Sigma\in\mathbf{Sig}}$ of $\mathbb{I}'$-signature morphisms $m_\Sigma : \Phi^1(\Sigma) \to \Phi^2(\Sigma)$, which can be understood as modifying μ_1 in order to make it equal to μ_2. For example, the sentence translations

$$\alpha^1_\Sigma : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}'(\Phi^1(\Sigma))$$
$$\alpha^2_\Sigma : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}'(\Phi^2(\Sigma))$$

may be quite different. Yet, by composing α^1_Σ with $\mathbf{Sen}'(m_\Sigma)$, the difference can be bridged. We obtain a comorphism modification if syntax, model and proof translation can be bridged uniformly, that is, by using the respective translations induced by m_Σ.

3.3. *Meta-logics*

Logic comorphisms $\mathbb{I} \to \mathbb{I}'$ allow the representation of a logic $\mathbb{I}$ in a logic $\mathbb{I}'$. An important special case arises when this representation is *adequate* in the intuitive sense that the semantics of $\mathbb{I}$ is preserved when representing it in $\mathbb{I}'$. Our understanding of $\mathbb{I}'$ can then be applied to study properties of $\mathbb{I}$. If we use a fixed logic $\mathbb{M} = \mathbb{I}'$, in which multiple other logics are represented adequately, we refer to them as *logic encodings* in a *meta-logic*. Occasionally, the term ‘universal logic’ is used instead of ‘meta-logic’ (for example, in Tarlecki (1996)), but we do not use it here in order to prevent any confusion with the general field of ‘universal logic’, which investigates common structures of all logics.

Alternatively, instead of giving a logic comorphism from a given logic $\mathbb{I}$ to $\mathbb{M}$, we can start with a partial logic – for example, an institution or even just a category **Sig** – and translate only that into $\mathbb{M}$. The missing components of a logic can then be inherited – often called *borrowed* – from $\mathbb{M}$. In this case, we refer to *logic definitions.*

Similarly, a meta-logic can be applied to define or encode logic comorphisms so that we have four cases in the end: defining/encoding a logic/logic comorphism. In the following, we will derive the general properties of these four concepts for an arbitrary meta-logic $\mathbb{M}$. Then, in Section 4, we will give the concrete definition of our meta-logic $\mathbb{M}$ as well as examples.

3.3.1. *Defining logics.*

Given a signature category **Sig**, to *define a logic* in a meta-logic $\mathbb{M}$ means to relate **Sig** to $\mathbb{M}$ in such a way that the syntax, model theory and proof theory of $\mathbb{M}$ can be borrowed to extend **Sig** to a logic.

Notation 3.23. If $\models$ is a family $(\models_\Sigma)_{\Sigma\in\mathbf{Sig}}$ and $\Phi : \mathbf{Sig}' \to \mathbf{Sig}$ is a functor, we write $\models \circ \Phi$ for the family $(\models_{\Phi(\Sigma)})_{\Sigma\in\mathbf{Sig}'}$. We also define $\vdash \circ \Phi$ accordingly. In the light of Remark 3.13, this is simply the composition of a functor and a natural transformation.

Definition 3.24. Let $\mathbb{M}$ be a logic. A *logic definition* in $\mathbb{M}$ consists of a category **Sig** and a functor $\Phi : \mathbf{Sig} \to \mathbf{Sig}^{\mathbb{M}}$. Such a logic definition induces a logic

$$\mathbb{M} \circ \Phi = (\mathbf{Sig}, \mathbf{Sen}^{\mathbb{M}} \circ \Phi, \mathbf{Mod}^{\mathbb{M}} \circ \Phi, \models^{\mathbb{M}} \circ \Phi, \mathbf{Pf}^{\,\mathbb{M}} \circ \Phi, \vdash^{\mathbb{M}} \circ \Phi).$$

Remark 3.25. With the notation of Remark 3.13, this yields the diagram

$$\begin{tikzcd}
& & \mathcal{CAT}^{op} \arrow[r, "|-|^r"] & \mathcal{REL} \\
\mathbf{Sig} \arrow[r, "\Phi"] & \mathbf{Sig}^{\mathbb{M}} \arrow[ur, "\mathbf{Mod}^{\mathbb{M}}"] \arrow[r, "\mathbf{Sen}^{\mathbb{M}}"] \arrow[dr, "\mathbf{Pf}^{\mathbb{M}}"'] & \mathcal{SET} \arrow[u, Rightarrow, "\models^{\mathbb{M}}"'] \arrow[d, Rightarrow, "\vdash^{\mathbb{M}}"] \arrow[ur, hook] \arrow[dr, hook] & \\
& & \mathcal{PFCAT} \arrow[r, "|-|"'] & \mathcal{CLASS}
\end{tikzcd}$$

Moreover, Φ induces a canonical comorphism $\mathbb{M} \circ \Phi \to \mathbb{M}$.

Theorem 3.26 (logic definitions). $\mathbb{M} \circ \Phi$ is indeed a logic. Moreover, it is (strongly) sound or (strongly) complete if $\mathbb{M}$ is.

Proof. The fact that $\mathbb{M} \circ \Phi$ is a logic follows immediately from the diagram in Remark 3.25. In particular, $\models^{\mathbb{M}} \circ \Phi$ and $\vdash^{\mathbb{M}} \circ \Phi$ are natural transformation because they arise by composing a natural transformation with a functor.

It is inherent in Definition 3.24 that $\Theta \vdash^{\mathbb{M}\circ\Phi}_{\Sigma} F$ if and only if $\Theta \vdash^{\mathbb{M}}_{\Phi(\Sigma)} F$, and similarly for the entailment relation. This then yields the (strong) soundness/completeness result. □

Remark 3.27. In Remark 3.14, we stated that institutions are functors into $(U \downarrow V)$, and thus, in particular, $\mathbb{M} : \mathbf{Sig}^{\mathbb{M}} \to (U \downarrow V)$. So $\mathbb{M} \circ \Phi$ is indeed the composition of two functors (which justifies our notation).

3.3.2. *Encoding logics.* While a logic definition starts with a signature category and borrows all other notions from $\mathbb{M}$, a *logic encoding* starts with a whole logic

$$\mathbb{I} = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models, \mathbf{Pf}, \vdash)$$

and a functor

$$\Phi : \mathbf{Sig} \to \mathbf{Sig}^{\mathbb{M}}.$$

Definition 3.28 (logic encoding). An *encoding* of a logic $\mathbb{I}$ in $\mathbb{M}$ is a logic comorphism

$$(\Phi, \alpha, \beta, \gamma) : \mathbb{I} \to \mathbb{M}.$$

(Equivalently, we can say that $(id, \alpha, \beta, \gamma)$ is a logic comorphism $\mathbb{I} \to \mathbb{M} \circ \Phi$.)

When encoding logics in a meta-logic, a central question is *adequacy*. Intuitively, adequacy means that the encoding is correct with respect to the encoded logic. More formally, a logic encoding yields two logics $\mathbb{I}$ and $\mathbb{M} \circ \Phi$, and adequacy is a statement about the relation between them.

Ideally, α, β and γ are isomorphisms, in which case $\mathbb{I}$ and $\mathbb{M} \circ \Phi$ are isomorphic in the category $\mathcal{LOG}$. However, it is often sufficient to show that the consequence relations in $\mathbb{I}$ and $\mathbb{M} \circ \Phi$ coincide. This motivates the following definition and theorem.

Definition 3.29 (adequate encodings). In the situation of Definition 3.28, the encoding is said to be *adequate* if:

— each β_Σ is surjective on objects (model-theoretical adequacy);
— each γ_Σ is 'surjective' on morphisms in the sense that for any A, B, if there is a morphism from $\gamma_\Sigma(A)$ to $\gamma_\Sigma(B)$, then there is also one from A to B (proof-theoretical adequacy).

Model-theoretical adequacy is just the *model expansion property* (see, for example, Cerioli and Meseguer (1997)). Intuitively, this means that a property that holds for all models of $\mathbf{Mod}^{\mathbb{M}}(\Phi(\Sigma))$ can be used to establish the corresponding property for all models of Σ. Proof-theoretical adequacy means that proofs found in $\mathbb{M} \circ \Phi$ give rise to corresponding proofs in $\mathbb{I}$. More precisely, we have the following theorem.

Theorem 3.30 (adequacy). Given an adequate logic encoding $(\Phi, \alpha, \beta, \gamma)$ of $\mathbb{I}$ in $\mathbb{M}$, we have for all $\mathbb{I}$-theories (Σ, Θ) and all Σ-sentences F,

$$\Theta \models^{\mathbb{I}}_{\Sigma} F \quad \text{iff} \quad \alpha_\Sigma(\Theta) \models^{\mathbb{M}}_{\Phi(\Sigma)} \alpha_\Sigma(F)$$

$$\Theta \vdash^{\mathbb{I}}_{\Sigma} F \quad \text{iff} \quad \alpha_\Sigma(\Theta) \vdash^{\mathbb{M}}_{\Phi(\Sigma)} \alpha_\Sigma(F).$$

In particular, $\mathbb{I}$ is strongly sound or complete if $\mathbb{M}$ is.

Proof. It is easy to show that the left-to-right implications hold for any logic comorphism.

The right-to-left implications are independent and only require the respective adequacy assumption. The model-theoretical result is essentially the known borrowing result for institutions (see Cerioli and Meseguer (1997)). For the proof-theoretical result, recall that γ_Σ is a functor $\mathbf{Pf}^{\mathbb{I}}(\Sigma) \to \mathbf{Pf}^{\mathbb{M}}(\Phi(\Sigma))$. Since provability is defined by the existence of morphisms in the proof category, the proof-theoretical adequacy then yields the result.

The soundness and completeness results then follow from Theorem 3.26. □

3.3.3. *Mixing defining and encoding.* Logic definitions, where we start with **Sig** and inherit all other components from $\mathbb{M}$, and logic encodings, where we start with a logic and encode all components in $\mathbb{M}$, are opposite extremes. We can also start, for example, with an institution $\mathbb{I} = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$, and give an institution comorphism $(\Phi, \alpha, \beta) : \mathbb{I} \to \mathbb{M}$, which encodes the syntax and model theory of $\mathbb{I}$ in $\mathbb{M}$ and inherits the proof theory. Dually, we can start with syntax and proof theory $\mathbb{I} = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Pf}, \vdash)$, and give a comorphism $(\Phi, \alpha, \gamma) : \mathbb{I} \to \mathbb{M}$, which encodes the syntax and proof theory and inherits the model theory.

The first of these is often used to apply an implementation of the proof-theoretical consequence relation of $\mathbb{M}$ to reason about the model-theoretical consequence relation of $\mathbb{I}$, which is a technique known as *borrowing* (Cerioli and Meseguer 1997). The borrowing theorem can be recovered as a special case of the above, and we briefly state it as follows in our notation.

Theorem 3.31. Let $\mathbb{I}$ be an institution and $\mathbb{M}$ be a logic, and let $\mu = (\Phi, \alpha, \beta)$ be an institution comorphism from $\mathbb{I}$ to $\mathbb{M}$ (*qua* institution). Then we can extend $\mathbb{I}$ to a logic $\mathbb{I}'$ by putting $\mathbf{Pf}' := \mathbf{Pf}^{\mathbb{M}} \circ \Phi$ and $\vdash' := (\vdash^{\mathbb{M}} \circ \Phi) \circ \alpha$. Moreover, we have that $(\Phi, \alpha, \beta, id) : \mathbb{I}' \to \mathbb{M}$ is a logic comorphism that is an adequate encoding if all β_Σ are surjective on objects.

Proof. Basic properties of category theory mean that $\mathbb{I}'$ is indeed a logic. Model-theoretical adequacy holds due to the surjectivity assumption, and proof-theoretical adequacy follows because the proof translation is the identity. □

3.3.4. *Defining logic comorphisms.* We now turn to the treatment of logic comorphisms. To define a logic comorphism, we first give a family of signature morphisms in $\mathbb{M}$.

Definition 3.32 (defining comorphisms). Given two logic definitions

$$\Phi^i : \mathbf{Sig}^i \to \mathbf{Sig}^{\mathbb{M}}$$

for $i = 1, 2$, a *comorphism definition* consists of a functor $\Phi : \mathbf{Sig}^1 \to \mathbf{Sig}^2$ and a natural transformation $m : \Phi^1 \to \Phi^2 \circ \Phi$. Such a comorphism definition induces a logic comorphism

$$(\Phi, \mathbb{M} \circ m) = (\Phi, \mathbf{Sen}^{\mathbb{M}} \circ m, \mathbf{Mod}^{\mathbb{M}} \circ m, \mathbf{Pf}^{\mathbb{M}} \circ m)$$

from $\mathbb{M} \circ \Phi^1$ to $\mathbb{M} \circ \Phi^2$.

Remark 3.33. With the notation of Remark 3.25, this yields the diagram

$$\begin{tikzcd}
\mathbf{Sig}^1 \arrow[dd, "\Phi"'] \arrow[drr, "\Phi^1", ""{name=A, below}] & & & \mathcal{CAT}^{op} \arrow[r, "|-|^r"] & \mathcal{REL} \\
& & \mathbf{Sig}^{\mathbb{M}} \arrow[ur, "\mathbf{Mod}^{\mathbb{M}}"] \arrow[r, "\mathbf{Sen}^{\mathbb{M}}"] \arrow[dr, "\mathbf{Pf}^{\mathbb{M}}"'] & \mathcal{SET} \arrow[u, Rightarrow, "\models^{\mathbb{M}}"'] \arrow[d, Rightarrow, "\vdash^{\mathbb{M}}"] \arrow[ur, hook] \arrow[dr, hook] & \\
\mathbf{Sig}^2 \arrow[urr, "\Phi^2"', ""{name=B, above}] & & & \mathcal{PFCAT} \arrow[r, "|-|"'] & \mathcal{CLASS}
\arrow[Rightarrow, from=A, to=B, "m"']
\end{tikzcd}$$

where double arrows again indicate natural transformations. So m_Σ is a family of $\mathbf{Sig}^{\mathbb{M}}$-morphisms

$$m_\Sigma : \Phi^1(\Sigma) \to \Phi^2(\Phi(\Sigma))$$

for $\Sigma \in \mathbf{Sig}^1$. Sentence, proof and model translation are then given by $\mathbf{Sen}^{\mathbb{M}}(m_\Sigma)$, $\mathbf{Mod}^{\mathbb{M}}(m_\Sigma)$ and $\mathbf{Pf}^{\mathbb{M}}(m_\Sigma)$, respectively.

Theorem 3.34 (defining comorphisms). $(\Phi, \mathbb{M} \circ m)$ is indeed a logic comorphism.

Proof. It is clear from the diagram in Remark 3.33 that the types of the components of $(\Phi, \mathbb{M} \circ m)$ are correct because they arise by composing functors and natural transformations.

It remains to show that Conditions (1) and (2) in Definition 3.20 hold. For the latter, we substitute the respective definitions for $\vdash$, $\vdash'$, α and γ to give

$$(\mathbf{Pf}^{\mathbb{M}} \circ m)_\Sigma\big((\vdash^{\mathbb{M}} \circ \Phi^1)_\Sigma\ F\big) \;=\; (\vdash^{\mathbb{M}} \circ \Phi^2)_{\Phi(\Sigma)}\ (\mathbf{Sen}^{\mathbb{M}} \circ m)_\Sigma(F).$$

This simplifies to

$$\mathbf{Pf}^{\mathbb{M}}(m_\Sigma)\,(\vdash^{\mathbb{M}}_{\Phi^1(\Sigma)}\ F) \;=\; \vdash^{\mathbb{M}}_{\Phi^2(\Phi(\Sigma))}\ \mathbf{Sen}^{\mathbb{M}}(m_\Sigma)(F),$$

which is exactly the condition from Definition 3.10 for the morphism m_Σ.

Condition (1) simplifies in the same way to the satisfaction condition from Definition 3.2. □

Remark 3.35. Recalling Remark 3.14 and continuing Remark 3.27, note that $\mathbb{M} \circ m$ is a natural transformation $\mathbb{M} \circ \Phi^1 \to \mathbb{M} \circ \Phi^2 \circ \Phi$ between functors $\mathbf{Sig} \to (U \downarrow V)$. Thus,

$$(\Phi, \mathbb{M} \circ m) : \mathbb{M} \circ \Phi^1 \to \mathbb{M} \circ \Phi^2$$

is indeed a morphism in the lax slice category of objects over $(U \downarrow V)$ (which justifies our notation). This yields a simpler, more abstract proof of Theorem 3.34.

3.3.5. *Encoding logic comorphisms.* Finally, we study the encoding of existing logic comorphisms.

Definition 3.36 (encoding comorphisms). Given a logic comorphism $\mu : \mathbb{I}^1 \to \mathbb{I}^2$ and two logic encodings $\mu^i : \mathbb{I}^i \to \mathbb{M}$, an *encoding* of μ relative to μ_1 and μ_2 is a logic comorphism modification $\mu^1 \to \mu^2 \circ \mu$.

To understand this definition better, consider a comorphism

$$\mu = (\Phi, \alpha, \beta, \gamma) : \mathbb{I}^1 \to \mathbb{I}^2$$

and two encodings

$$\mu^i = (\Phi^i, \alpha^i, \beta^i, \gamma^i) : \mathbb{I}^i \to \mathbb{M}.$$

Then an encoding m of μ is a natural transformation $\Phi^1 \to \Phi^2 \circ \Phi$ such that the following diagrams commute for all $\Sigma \in \mathbf{Sig}^1$:

$$\begin{tikzcd}
\mathbf{Sen}^1(\Sigma) \arrow[r, "\alpha_\Sigma"] \arrow[d, "\alpha^1_\Sigma"'] & \mathbf{Sen}^2(\Phi(\Sigma)) \arrow[d, "\alpha^2_{\Phi(\Sigma)}"] \\
\mathbf{Sen}^{\mathbb{M}}(\Phi^1(\Sigma)) \arrow[r, "\mathbf{Sen}^{\mathbb{M}}(m_\Sigma)"] & \mathbf{Sen}^{\mathbb{M}}(\Phi^2(\Phi(\Sigma)))
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbf{Mod}^1(\Sigma) & \mathbf{Mod}^2(\Phi(\Sigma)) \arrow[l, "\beta_\Sigma"'] \\
\mathbf{Mod}^{\mathbb{M}}(\Phi^1(\Sigma)) \arrow[u, "\beta^1_\Sigma"] & \mathbf{Mod}^{\mathbb{M}}(\Phi^2(\Phi(\Sigma))) \arrow[l, "\mathbf{Mod}^{\mathbb{M}}(m_\Sigma)"'] \arrow[u, "\beta^2_{\Phi(\Sigma)}"']
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbf{Pf}^{\,1}(\Sigma) \arrow[r, "\gamma_\Sigma"] \arrow[d, "\gamma^1_\Sigma"'] & \mathbf{Pf}^{\,2}(\Phi(\Sigma)) \arrow[d, "\gamma^2_{\Phi(\Sigma)}"] \\
\mathbf{Pf}^{\,\mathbb{M}}(\Phi^1(\Sigma)) \arrow[r, "\mathbf{Pf}^{\,\mathbb{M}}(m_\Sigma)"] & \mathbf{Pf}^{\,\mathbb{M}}(\Phi^2(\Phi(\Sigma)))
\end{tikzcd}$$

Intuitively, $m_\Sigma : \Phi^1(\Sigma) \to \Phi^2(\Phi(\Sigma))$ is a signature morphism in the meta-logic such that $\mathbf{Sen}^{\mathbb{M}}(m_\Sigma)$ has the same effect as α_Σ. This is particularly intuitive in the typical case where α^1_Σ and $\alpha^2_{\Phi(\Sigma)}$ are bijections. Similarly, $\mathbf{Mod}^{\mathbb{M}}(m_\Sigma)$ and $\mathbf{Pf}^{\,\mathbb{M}}(m_\Sigma)$ must have the same effect as β_Σ and γ_Σ.

4. A meta-logic

We will now define a specific logic

$$\mathbb{M} = (\mathbf{Sig}^{\mathbb{M}}, \mathbf{Sen}^{\mathbb{M}}, \mathbf{Mod}^{\mathbb{M}}, \models^{\mathbb{M}}, \mathbf{Pf}^{\,\mathbb{M}}, \vdash^{\mathbb{M}}),$$

which we will use as our meta-logic. $\mathbb{M}$ is based on the dependent type theory of LF.

We could define $\mathbb{M}$ such that the signatures of $\mathbb{M}$ are the LF signatures. There are several choices for the sentences of such a logic. The most elegant one uses the types as the sentences, in which case the proof categories are the categories of contexts, and models are usually based on locally cartesian closed categories (Seely 1984; Cartmell 1986). For example, we gave a (sound and complete) logic in this style, though not using the terminology used here, in Awodey and Rabe (2011).

However, this is not our intention here. We want to use $\mathbb{M}$ as a logical framework in which *all* components of an object logic are represented in terms of the syntax of the meta-logic. In order to capture all aspects of a logic, the $\mathbb{M}$-signatures must be more complex.

We will first define the syntax, proof theory and model theory of $\mathbb{M}$ in Section 4.1, 4.2 and 4.3, respectively, and then define $\mathbb{M}$ and discuss its basic properties in Section 4.4. We use propositional modal logic as a running example. $\mathbb{M}$ is foundation independent: object logic models are represented using the syntax of the meta-logic in a way that does not commit to a particular foundation of mathematics. We will discuss the use of foundations in more detail in Section 4.5.

4.1. *Syntax*

The encoding of logical syntax in LF has been well studied, and we will motivate our definitions of $\mathbb{M}$ signatures by analysing existing logic encodings in LF:

(1) Logic encodings in LF usually employ a distinguished type `form` for the formulas and a distinguished judgment `ded` : `form` $\to$ `type` for the truth judgment. But the names of these symbols may differ, and in some cases `form` is not even a symbol. Therefore, we fix a signature *Base* as follows:

```
form  :  type
ded   :  form → type
```

Now, by considering morphisms out of *Base*, that is, pairs of a signature Σ and a morphism $base : Base \to \Sigma$, we can specify these distinguished types explicitly. *base*(`form`) is a type over Σ representing the syntactic class of formulas. And *base*(`ded`) is a type family over Σ representing the truth of formulas. By investigating morphisms out of *Base*, we are restricting attention to those LF signatures that define formulas and truth – the central properties that separate logic encodings from other LF signatures.

(2) We want to represent models of a signature Σ as signature morphisms out of Σ. However, models often have components that are not mentioned in the syntax, for example, the set of worlds in a Kripke model. Therefore, we will use two LF-signatures Σ^{syn} and Σ^{mod} to represent the syntax and model theory separately.
Σ^{syn} declares the syntax of a logical language, that is, LF symbols that represent syntactic classes (for example, terms or formulas), sorts, functions, predicates, and so on. Σ^{mod} declares the model theory, that is, LF symbols that axiomatise the properties of models, typically in terms of a foundational semantic language like set theory, type theory or category theory. A morphism $\mu : \Sigma^{syn} \to \Sigma^{mod}$ interprets the syntax in terms of the model theory and represents the inductive interpretation function that translates syntax into the semantic realm. Finally, individual models are represented as morphisms out of Σ^{mod}.

(3) Because of the symmetry between model and proof theory, it is elegant to split syntax and proof theory as well by introducing an LF-signature Σ^{pf} and a morphism $\pi : \Sigma^{syn} \to \Sigma^{pf}$.
Σ^{pf} declares the proof theory, that is, LF symbols that represent judgments about the syntax and inference rules for them. It typically differs from Σ^{syn} by declaring auxiliary syntax such as signed formulas in a tableaux calculus. In many cases, π is an inclusion.

Hence, we arrive at the following diagram:

$$\begin{tikzcd}
 & & \Sigma^{pf} \\
Base \arrow[r, "base"] & \Sigma^{syn} \arrow[ur, "\pi"] \arrow[dr, "\mu"'] & \\
 & & \Sigma^{mod}
\end{tikzcd}$$

The symbol `ded` plays a crucial duplicate role. *Proof-theoretically*, the Σ^{pf}-type

$$\pi(base(\texttt{ded})\ F)$$

is the type of proofs of F. A proof from assumptions Γ is a Σ^{pf}-term in context Γ. This yields the well-known Curry–Howard representation of proofs as terms (Martin-Löf 1996; Pfenning 2001). *Model-theoretically*, the Σ^{mod}-type $\mu(base(\texttt{form}))$ is the type of truth values, and $\mu(base(\texttt{ded}))$ is a predicate on it giving the designated truth values. Below, we will define models as morphisms $m : \Sigma^{mod} \to M$, and (M, m) satisfies F if and only if the type

$$m(\mu(base(\texttt{ded})\ F))$$

is inhabited over M. This yields the well-known representation of models as morphisms out of an initial object in a suitable category (Lawvere 1963; Goguen *et al.* 1978). Thus, ded is the mediator between proof and model theory, and Σ^{pf} and Σ^{mod} encode the two different ways to give meaning to Σ^{syn}.

We summarise all this in the following definition.

Definition 4.1 (signatures). The *signatures* of $\mathbb{M}$ are tuples

$$\Sigma = (\Sigma^{syn}, \Sigma^{pf}, \Sigma^{mod}, base, \pi, \mu)$$

forming a diagram of LF signatures as given above. $base(\texttt{ded})$ must be a constant.

The restriction of $base(\texttt{ded})$ to constants is a bit inelegant. But it is not harmful in practice and permits us to exclude some degenerate cases that would make later definitions more complicated.

As expected, sentences are the LF terms of type form over Σ^{syn}.

Definition 4.2 (sentences). Given an $\mathbb{M}$ signature Σ as in Definition 4.1, we define the *sentences* by

$$\mathbf{Sen}^{\mathbb{M}}(\Sigma) = \{F \mid \cdot \vdash_{\Sigma^{syn}} F : base(\texttt{form})\}.$$

Example 4.3 (modal logic continued). For our simple modal logic $\mathbb{ML}$ from Example 2.1, we use a signature ML^{syn} as follows:

form	:	type
$\supset$	:	form $\to$ form $\to$ form
$\Box$	:	form $\to$ form
ded	:	form $\to$ type

Here $\supset$ and $\Box$ are formula constructors for implication and necessity. The morphism $base : Base \to ML^{syn}$ is simply an inclusion. If we want to declare propositional variables, we can add them using declarations $p : \texttt{form}$. Let PQ^{syn} be the extension of ML^{syn} with $p : \texttt{form}$ and $q : \texttt{form}$. An example sentence is $T = \Box\ p \supset p \in \mathbf{Sen}^{\mathbb{M}}(PQ)$, which is an instance of the axiom scheme T. As we have done here for $\supset$, we will use intuitive infix and bracket elimination rules when giving example expressions.

Definition 4.4 (signature morphism). An $\mathbb{M}$-*signature morphism* $\sigma : \Sigma \to \Sigma'$ is a tuple $(\sigma^{syn}, \sigma^{pf}, \sigma^{mod})$ such that:

— the following diagram commutes:

$$\begin{tikzcd}
 & \Sigma^{pf} \arrow[rr, "\sigma^{pf}"] & & \Sigma'^{pf} \\
\Sigma^{syn} \arrow[ur, "\pi"] \arrow[rr, "\sigma^{syn}"] \arrow[dr, "\mu"'] & & \Sigma'^{syn} \arrow[ur, "\pi'"'] \arrow[dr, "\mu'"] & \\
 & \Sigma^{mod} \arrow[rr, "\sigma^{mod}"] & & \Sigma'^{mod}
\end{tikzcd}$$

— for every $F \in \mathbf{Sen}^{\mathbb{M}}(\Sigma)$, there is an F' such that $\sigma^{syn}(base(\mathtt{ded})\ F) = base'(\mathtt{ded})\ F'$.

Identity and composition for $\mathbf{Sig}^{\mathbb{M}}$-morphisms are defined component-wise. σ is *simple* if additionally $\sigma^{syn} \circ base = base'$.

Remark 4.5. The diagram in Definition 4.4 omits *Base* and thus does not imply $\sigma^{syn} \circ base = base'$, which only holds for simple morphisms, though it is natural to restrict attention to simple morphisms. In particular, the second condition in Definition 4.4 is redundant for simple morphisms as we always have $F' = \sigma^{syn}(F)$.

However, our slightly weaker definition is crucial for the representation of many logic translations, as we will see in Example 5.13 and discuss in Remark 5.15.

Definition 4.4 is just strong enough to guarantee the existence of a *sentence translation.*

Definition 4.6 (sentence translation). In the situation of Definition 4.4, we put

$$\mathbf{Sen}^{\mathbb{M}}(\sigma)(F) = \text{ the } F' \text{ such that } \sigma^{syn}(base(\mathtt{ded})\ F) = base'(\mathtt{ded})\ F'.$$

This is well defined because F' exists by Definition 4.4; and if it exists, it is necessarily unique.

Remark 4.7. In particular, sentence translation along simple morphisms is just the application of the morphism σ^{syn}: $\sigma^{syn}(base(o)) = base'(o)$ and $\mathbf{Sen}^{\mathbb{M}}(\sigma)(F) = \sigma^{syn}(F)$.

Example 4.8 (modal logic continued). A simple $\mathbf{Sig}^{\mathbb{M}}$-morphism $w : PQ \to PQ$ can contain the component

$$w^{syn} \quad = \quad id,\ q := \Box q,\ p := q$$

where *id* is the identity morphism for PQ^{syn}. Then we have

$$w^{syn}(\mathtt{form}) = \mathtt{form}$$
$$w^{syn}(\mathtt{ded}) = \mathtt{ded}$$
$$w^{syn}(\mathtt{ded}\ T) = \mathtt{ded}\ \Box\ q \supset q.$$

Thus, we obtain,

$$\mathbf{Sen}^{\mathbb{M}}(w)(T) = \Box q \supset q.$$

4.2. *Proof theory*

The idea behind the proof theory of $\mathbb{M}$ is the Curry–Howard correspondence (Curry and Feys 1958; Howard 1980): judgments $(E_1, \ldots, E_n)$ correspond to contexts $x_1 : E_1, \ldots, x_n :$

E_n, where the variables act as names for hypotheses. Moreover, a morphism from $\Gamma = x_1 : E_1, \ldots, x_m : E_m$ to $\Gamma' = x_1 : F_1, \ldots, x_n : F_n$ provides one proof over Γ for every judgment assumed in Γ', that is, it is a tuple p_1^n of terms in context $\Gamma \vdash_{\Sigma^{pf}} p_i : F_i$. Such a tuple is simply a substitution from Γ' to Γ, and the proof categories are the well-understood categories of contexts and substitutions. The finite products are given by concatenations of contexts (possibly using α-renaming to avoid duplicate variables).

Definition 4.9 (proof categories). Given an $\mathbb{M}$ signature Σ as in Definition 4.1, we define the proof category $\mathbf{Pf}^{\mathbb{M}}(\Sigma)$ as the dual of the category of contexts and substitutions over Σ^{pf}:

— the objects are the contexts over Σ^{pf};
— the morphisms from Γ to Γ' are the substitutions from Γ' to Γ.

Example 4.10 (modal logic continued). For our modal logic, we can define ML^{pf} by extending ML^{syn} with a calculus for provability. Thus, π_{ML} is an inclusion. There are a number of ways to do this (see Avron *et al.* (1998)); for a simple Hilbert-style calculus, ML^{pf} consists of axioms for propositional logic and the declarations

$$\begin{array}{lll} mp & : & \{x:\mathtt{form}\}\{y:\mathtt{form}\}\,\mathtt{ded}\ x \supset y \to \mathtt{ded}\ x \to \mathtt{ded}\ y \\ nec & : & \{x:\mathtt{form}\}\,\mathtt{ded}\ x \to \mathtt{ded}\ \Box\, x \\ K & : & \{x:\mathtt{form}\}\{y:\mathtt{form}\}\,\mathtt{ded}\ \Box(x \supset y) \supset (\Box x \supset \Box y) \end{array}$$

Note how the Π-binder $\{x : \mathtt{form}\}$ is used to declare schema variables.

Similarly, we define PQ^{pf} by extending ML^{pf} with $p : \mathtt{form}$ and $q : \mathtt{form}$. Then the proof category $\mathbf{Pf}^{\mathbb{M}}(PQ)$ contains objects like

$$\Gamma_1 \quad = \quad x : \mathtt{ded}\ \Box(p \supset q),\ y : \mathtt{ded}\ \Box p$$

and

$$\Gamma_2 \quad = \quad z : \mathtt{ded}\ \Box q.$$

A $\mathbf{Pf}^{\mathbb{M}}(PQ)$-morphism from Γ_1 to Γ_2 is the substitution

$$\gamma_1 \quad = \quad z := mp\ \underline{\Box p}\ \underline{\Box q}\ \big(mp\ \underline{\Box(p \supset q)}\ \underline{\Box p \supset \Box q}\ (K\ \underline{p}\ \underline{q})\ x\big)\ y.$$

This gives a proof of the goal labelled z in terms of the two assumptions labelled x and y. To enhance readability, we have underlined the arguments that instantiate the schema variables.

Just as the proof categories are the categories of contexts and substitutions over the LF signature Σ^{pf}, the proof translation functor is given by the application of the LF morphism σ^{pf} to contexts and substitutions.

Definition 4.11 (proof translation). Given an $\mathbb{M}$-signature morphism $\sigma : \Sigma \to \Sigma'$ as in Definition 4.4, the proof translation functor $\mathbf{Pf}^{\mathbb{M}}(\sigma) : \mathbf{Pf}^{\mathbb{M}}(\Sigma) \to \mathbf{Pf}^{\mathbb{M}}(\Sigma')$ is defined by

$$\begin{array}{ll} \mathbf{Pf}^{\mathbb{M}}(\sigma)(\Gamma) = \sigma^{pf}(\Gamma) & \text{for } \Gamma \in |\mathbf{Pf}^{\mathbb{M}}(\Sigma)| \\ \mathbf{Pf}^{\mathbb{M}}(\sigma)(\gamma) \ = \sigma^{pf}(\gamma) & \text{for } \gamma \in \mathbf{Pf}^{\mathbb{M}}(\Sigma)(\Gamma, \Gamma'). \end{array}$$

Example 4.12 (modal logic continued). Reusing the morphism w and the object Γ_1 from our running example, we obtain

$$\mathbf{Pf}^{\,\mathbb{M}}(w)(\Gamma_1) \quad = \quad x : \mathtt{ded}\ \Box(q \supset \Box q),\ y : \mathtt{ded}\ \Box q$$

and

$$\mathbf{Pf}^{\,\mathbb{M}}(w)(\gamma_1) \quad = \quad z := mp\ \underline{\Box q}\ \underline{\Box\Box q}\ \big(mp\ \underline{\Box(q \supset \Box q)}\ \underline{\Box q \supset \Box\Box q}\ (K\ \underline{q}\ \underline{\Box q})\ x\big)\ y,$$

which is a $\mathbf{Pf}^{\,\mathbb{M}}(PQ)$-morphism from $\mathbf{Pf}^{\,\mathbb{M}}(w)(\Gamma_1)$ to $\mathbf{Pf}^{\,\mathbb{M}}(w)(\Gamma_2)$.

As expected, the truth judgment is given by the image of $\mathtt{ded}$ in Σ^{pf}:

Definition 4.13 (truth judgment). Given an $\mathbb{M}$ signature Σ as in Definition 4.1, the truth judgment $\vdash^{\mathbb{M}}_{\Sigma} F$ is defined as the context $x : \pi(base(\mathtt{ded})\ F)$ for an arbitrary fixed variable x.

Example 4.14 (modal logic continued). We have

$$\begin{aligned} \vdash^{\mathbb{M}}_{PQ} \Box(p \supset q) \times \vdash^{\mathbb{M}}_{PQ} \Box p \cong &\quad \Gamma_1 \\ \vdash^{\mathbb{M}}_{PQ} \Box q \cong &\quad \Gamma_2. \end{aligned}$$

The projections out of the product Γ_1 are just inclusion substitutions. And the isomorphism in the second case is a variable renaming. Thus, the morphism γ_1 proves

$$\{\Box(p \supset q), \Box p\} \vdash^{\mathbb{M}}_{PQ} \Box q.$$

Similarly, the morphism $\mathbf{Pf}^{\,\mathbb{M}}(w)(\gamma_1)$ proves

$$\{\Box(q \supset \Box q), \Box q\} \vdash^{\mathbb{M}}_{PQ} \Box\Box q.$$

4.3. *Model theory*

The definition of Σ^{mod} is the most difficult part because Σ^{mod} must declare symbols for all components that are present in a model. Since a model may involve any mathematical object, Σ^{mod} must be expressive enough to define arbitrary mathematical objects. Therefore, Σ^{mod} must include a representation of the whole foundation of mathematics. The foundation is flexible: for example, we have represented Zermelo–Fraenkel set theory, Mizar's set theory, and higher-order logic HOL as foundations in the applications of our framework (Horozal and Rabe 2011; Iancu and Rabe 2011; Codescu *et al.* 2011). For simplicity, we will use higher-order logic as the foundation here. We start with our running example.

Example 4.15 (modal logic continued). To describe the model theory of modal logics, we reuse the signature HOL that was introduced as a running example in Section 2.2. We assume it also provides a declaration

$$\forall \quad : \quad (tm\ A \to tm\ bool) \to tm\ bool$$

for universal quantification over elements of A.

Then ML^{mod} can be given by extending HOL with

$$
\begin{array}{lcl}
\mathit{worlds} & : & \mathit{tp} \\
\mathit{acc} & : & \mathit{tm\ worlds} \to \mathit{tm\ worlds} \to \mathit{tm\ bool}
\end{array}
$$

where *worlds* and *acc* represent the set of words and the accessibility relation of a Kripke model. The morphism $\mu_{ML} : ML^{syn} \to ML^{mod}$ can now be defined as

$$
\begin{array}{rcl}
\mathtt{form} & := & \mathit{tm\ worlds} \to \mathit{tm\ bool} \\
\supset & := & [f : \mathit{tm\ worlds} \to \mathit{tm\ bool}]\,[g : \mathit{tm\ worlds} \to \mathit{tm\ bool}] \\
 & & [w : \mathit{tm\ worlds}]\,(f\ w \Rightarrow g\ w) \\
\Box & := & [f : \mathit{tm\ worlds} \to \ \mathit{tm\ bool}] \\
 & & [w : \mathit{tm\ worlds}]\,\forall\,[w' : \mathit{tm\ worlds}]\,(\mathit{acc}\ w\ w' \Rightarrow f\ w') \\
\mathtt{ded} & := & [f : \mathit{tm\ worlds} \to \mathit{tm\ bool}]\,\mathit{ded}\ (\forall\,[w : \mathit{tm\ worlds}]\,f\ w).
\end{array}
$$

μ_{ML} formalises the fact that formulas are interpreted as functions from the set of worlds to the set of booleans. Consequently, $\mu_{ML}(\supset)$ takes two and returns one such function, and $\mu_{ML}(\Box)$ takes one and returns one such function. These formalise the interpretation of formulas in Kripke models in the usual way. Finally, $\mu_{ML}(\mathtt{ded})$ formalises the satisfaction relation: a formula holds in a model if it is true in all worlds.

To avoid confusion, it is useful to keep in mind that ded formalises truth in the object-logic ML, whereas *ded* formalises truth in the mathematical foundation HOL, which is used to express the model theory. As common in model-theoretical definitions of truth, the truth of formulas is defined in terms of the truth of the mathematical foundation.

The signature PQ^{mod} arises by extending ML^{mod} with declarations $p : \mu(\mathtt{form})$, that is, $p : \mathit{tm\ worlds} \to \mathit{tm\ bool}$, and similarly for q. The morphism $\mu_{PQ} : PQ^{syn} \to PQ^{mod}$ is given by $\mu,\ p := p,\ q := q$.

Now a model of PQ must provide specific values for *worlds*, *acc*, p and q in terms of HOL. Thus, we can represent models as morphisms from PQ^{mod} to HOL.

Then we arrive at the following definition of model categories.

Definition 4.16 (model categories). Given an $\mathbb{M}$ signature Σ as in Definition 4.1, we define the model category $\mathbf{Mod}^{\mathbb{M}}(\Sigma)$ as the slice category of objects under Σ^{mod}:

— objects are the pairs (M, m) for an LF-signature morphism $m : \Sigma^{mod} \to M$;
— morphisms from (M, m) to (M', m') are LF-signature morphisms $\varphi : M \to M'$ such that $\varphi \circ m = m'$;
— the identity morphism of (M, m) is id_M;
— the composition of $\varphi : (M, m) \to (M', m')$ and $\varphi' : (M', m') \to (M'', m'')$ is $\varphi' \circ \varphi$.

Here we use an arbitrary LF signature as the codomain of the models to avoid a commitment to a particular foundation. We will return to this in Remark 4.22.

The model translation functor is given by composition with σ^{mod}.

Definition 4.17 (model translation). Given an $\mathbb{M}$-signature morphism $\sigma : \Sigma \to \Sigma'$ as in Definition 4.4, we define $\mathbf{Mod}^{\mathbb{M}}(\sigma) : \mathbf{Mod}^{\mathbb{M}}(\Sigma') \to \mathbf{Mod}^{\mathbb{M}}(\Sigma)$ by

$$\mathbf{Mod}^{\mathbb{M}}(\sigma)(M, m) = \left(M, m \circ \sigma^{mod}\right)$$
$$\mathbf{Mod}^{\mathbb{M}}(\sigma)(\varphi) = \varphi.$$

The following commutative diagram illustrates the functor $\mathbf{Mod}^{\mathbb{M}}$:

$$\begin{tikzcd}
\Sigma^{mod} \arrow[r, "\sigma^{mod}"] \arrow[d, "m"'] & \Sigma'^{mod} \arrow[d, "m'"] \\
M \arrow[r, "\varphi"] & M'
\end{tikzcd}$$

Example 4.18 (modal logic continued). We can give an example model

$$(HOL, m_1) \in \mathbf{Mod}^{\mathbb{M}}(PQ)$$

by defining the LF-signature morphism m_1 to map:

— all symbols of HOL to themselves;
— *worlds* to the set $\mathbb{N}$;
— *acc* to the $\leqslant$ relation on $\mathbb{N}$;
— p and q to the predicates *odd* and *nonzero* on $\mathbb{N}$ that are true for odd and non-zero numbers, respectively.

The model-theoretical truth, that is, the satisfaction relation, is given by the inhabitation of the type $base(\mathtt{ded})\ F$ in a model.

Definition 4.19 (satisfaction). For a Σ-model (M, m) and a Σ-sentence F, we define

$$(M, m) \models^{\mathbb{M}}_{\Sigma} F \qquad \text{iff} \qquad \text{there exists } t \text{ such that } \cdot \vdash_M t : m(\mu(base(\mathtt{ded})\ F)).$$

Example 4.20 (modal logic continued). We can assume that HOL has expressions $true$: $tm\ bool$ and $false : tm\ bool$ such that $ded\ true$ is inhabited but $ded\ false$ is empty. Thus, we obtain for the satisfaction in the model (HOL, m_1) from above

$$(HOL, m_1) \models^{\mathbb{M}}_{PQ} F \qquad \text{iff} \qquad \text{exists } t \text{ such that } \cdot \vdash_{HOL} t : ded\ (\forall[w : tm\ \mathbb{N}]\ m_1(F)\ w).$$

This holds if and only if $m_1(F)$ is the constant function returning $true$ (up to provable equality in HOL). For example, if we assume that HOL contains proof rules to derive common HOL-theorems, we obtain

$$(HOL, m_1) \models^{\mathbb{M}}_{PQ} p \supset \Box q$$

because we can simplify

$$m_1(p \supset \Box q) \ \ w$$

to the formula

$$odd\ w \Rightarrow \forall[w' : tm\ \mathbb{N}]\ ((w \leqslant w') \Rightarrow nonzero\ w'),$$

which is true for all w.

Remark 4.21 (lax model categories). Definition 4.16 can be generalised by using lax slice categories under Σ^{mod}. Then model morphisms from (M, m) to (M', m') are pairs (φ, r) for an LF-signature morphism $\varphi : M \to M'$ and a 2-cell $r : \varphi \circ m \Rightarrow m'$. Model reduction can be generalised accordingly. However, this is problematic because there is no natural way to turn the category of LF signatures into a 2-category. The same remark applies when other type theories are used instead of LF.

We have recently developed a theory of logical relations for LF as relations between LF signature morphisms (Sojakova 2010). Binary logical relations do not form a 2-category either, but they do behave like 2-cells in many ways. We expect that our present definition can be improved along those lines but omit the details here.

Remark 4.22 (restricted model categories). This definition of models as morphisms (M, m) is very general, permitting, for example, the trivial model $(\Sigma^{mod}, id_{\Sigma^{mod}})$. It is often desirable to restrict attention to certain models or certain model morphisms.

Most importantly, we are usually interested in those models where the codomain is a fixed LF-signature $\mathcal{F}$ representing the foundation of mathematics, such as $\mathcal{F} = HOL$ above. Then the LF signature morphism $m \circ \mu$ maps the syntax into the foundation $\mathcal{F}$, that is, every syntactical expression to its interpretation in the language of mathematics. We will return to this in Section 4.5.

More generally, we can supplement our definition of signatures with a formal language in which structural properties of models and morphisms can be formulated. For example, Σ^{mod} might contain a declaration that requires m to be an elementary embedding. The key unsolved difficulty here is to design a formal language that is reasonably expressive without making the framework overly complex.

4.4. *Collecting the pieces together*

We conclude the definition of $\mathbb{M}$ with the following result.

Theorem 4.23. $\mathbb{M} = (\mathbf{Sig}^{\mathbb{M}}, \mathbf{Sen}^{\mathbb{M}}, \mathbf{Mod}^{\mathbb{M}}, \models^{\mathbb{M}}, \mathbf{Pf}^{\mathbb{M}}, \vdash^{\mathbb{M}})$ is a logic.

Proof. We have to prove a number of conditions:

— **Sig** is a category:
The only non-obvious aspect here is that the existence of F' in the definition of signature morphisms is preserved under composition. It is easy to see.

— $\mathbf{Sen}^{\mathbb{M}}$ is a functor:
This is straightforward.

— $\mathbf{Mod}^{\mathbb{M}}$ is a functor:
This is a standard result of category theory for slice categories (see, for example, Mac Lane (1998)).

— $\mathbf{Pf}^{\mathbb{M}}$ is a proof theory functor:
It is a standard result for the category of contexts in type theories that $\mathbf{Pf}^{\mathbb{M}}$ is a functor $\mathbf{Sig}^{\mathbb{M}} \to \mathcal{CAT}$ (see, for example, Pitts (2000)). Thus, we only need to show that $\mathbf{Pf}^{\mathbb{M}}(\Sigma)$ has finite products and that $\mathbf{Pf}^{\mathbb{M}}(\sigma)$ preserves these products. The products are given by $\Gamma_1 \times \ldots \times \Gamma_n = \Gamma'_1, \ldots, \Gamma'_n$, where Γ'_i arises from Γ_i by α-renaming to eliminate duplicate variable names. The empty product (terminal object) is the empty context.

— $\models^{\mathbb{M}}$ meets the satisfaction condition:
We assume $\sigma : \Sigma \to \Sigma'$, $F \in \mathbf{Sen}^{\mathbb{M}}(\Sigma)$ and $(M, m) \in \mathbf{Mod}^{\mathbb{M}}(\Sigma')$. Then

$$\begin{aligned}
&\mathbf{Mod}^{\mathbb{M}}(\sigma)(M,m) \models^{\mathbb{M}}_{\Sigma} F \\
&\qquad \text{iff there is an } M\text{-term of type } (m \circ \sigma^{mod})\big(\mu(base(\mathtt{ded})\ F)\big) \\
&\qquad \text{iff there is an } M\text{-term of type } m\big(\mu'\big(\sigma^{syn}(base(\mathtt{ded})\ F)\big)\big) \\
&\qquad \text{iff there is an } M\text{-term of type } m\big(\mu'\big(base'(\mathtt{ded})\ \mathbf{Sen}^{\mathbb{M}}(\sigma)(F)\big)\big) \\
&\qquad \text{iff } (M,m) \models^{\mathbb{M}}_{\Sigma'} \mathbf{Sen}^{\mathbb{M}}(\sigma)(F).
\end{aligned}$$

— $\vdash^{\mathbb{M}}$ is preserved:
We assume $\sigma : \Sigma \to \Sigma'$ and $F \in \mathbf{Sen}^{\mathbb{M}}(\Sigma)$. Then

$$\begin{aligned}
\mathbf{Pf}^{\mathbb{M}}(\sigma)(\vdash^{\mathbb{M}}_{\Sigma} F) &= x : \sigma^{pf}(\pi(base(\mathtt{ded})\ F)) \\
&= x : \pi'(\sigma^{syn}(base(\mathtt{ded})\ F)) \\
&= x : \pi'(base'(\mathtt{ded})\ \mathbf{Sen}^{\mathbb{M}}(\sigma)(F)) \\
&= \vdash^{\mathbb{M}}_{\Sigma'} \mathbf{Sen}^{\mathbb{M}}(\sigma)(F).
\end{aligned}$$
□

4.4.1. *Soundness and completeness.* $\mathbb{M}$ is intentionally neither sound nor complete because it is designed to be a meta-logic in which other possibly unsound or incomplete logics are represented. However, we do have the following soundness criterion.

Definition 4.24. An $\mathbb{M}$-signature $(\Sigma^{syn}, \Sigma^{pf}, \Sigma^{mod}, base, \pi, \mu)$ is said to be *sound* if there is an LF signature morphism $\psi : \Sigma^{pf} \to \Sigma^{mod}$ such that the following diagram commutes:

$$\begin{tikzcd}
 & \Sigma^{pf} \arrow[dd, "\psi"] \\
\Sigma^{syn} \arrow[ur, "\pi"] \arrow[dr, "\mu"'] & \\
 & \Sigma^{mod}
\end{tikzcd}$$

This definition is justified by the following theorem.

Theorem 4.25. The logic arising from $\mathbb{M}$ by considering only the sound $\mathbb{M}$-signatures is strongly sound.

Proof. We assume:

(i) Σ is a sound signature as in Definition 4.24, Θ is a set of Σ-sentences and F is a Σ-sentence.
(ii) $\Theta \vdash^{\mathbb{M}}_{\Sigma} F$, so we need to show $\Theta \models^{\mathbb{M}}_{\Sigma} F$.
(iii) we have an arbitrary model $(M, m) \in \mathbf{Mod}^{\mathbb{M}}(\Sigma)$ such that $(M, m) \models^{\mathbb{M}}_{\Sigma} A$ for every $A \in \Theta$, so we need to show $(M, m) \models^{\mathbb{M}}_{\Sigma} F$.

By (ii), there must be a substitution from $\vdash^{\mathbb{M}}_{\Sigma} F$ to $\vdash^{\mathbb{M}}_{\Sigma} F_1 \times \ldots \times \vdash^{\mathbb{M}}_{\Sigma} F_n$ for some $\{F_1, \ldots, F_n\} \subseteq \Theta$. Using the LF type theory, it is easy to show that this is equivalent to having a closed Σ^{pf}-term t of type

$$\pi\big(base(\mathtt{ded})\ F_1 \to \ldots \to base(\mathtt{ded})\ F_n \to base(\mathtt{ded})\ F\big).$$

By (i), we obtain a closed M-term $m(\psi(t))$ of type

$$m\big(\mu\big(base(\mathtt{ded})\ F_1 \to \ldots \to base(\mathtt{ded})\ F_n \to base(\mathtt{ded})\ F\big)\big).$$

Now, by (iii), there are M-terms t_i of type $m(\mu(base(\mathtt{ded})\ F_i))$. Thus, there is also an M-term $m(\psi(t))\ t_1\ \ldots\ t_n$ of type $m(\mu(base(\mathtt{ded})\ F))$, so $(M, m) \models^{\mathbb{M}}_{\Sigma} F$. □

We will refine this into a criterion to establish the soundness of a logic in Theorem 5.10.

Remark 4.26 (completeness). It does not make sense to use the analogue of Definition 4.24 for completeness: Σ^{mod} is typically much stronger than Σ^{pf} – indeed, it is often as strong as mathematics as a whole, as in Section 4.5 – so there can be no signature morphism $\Sigma^{mod} \to \Sigma^{syn}$. Moreover, even if we had such a morphism, it would not yield completeness because the proof of Theorem 4.25 crucially uses the composition $m \circ \psi$.

This is not surprising since a soundness result is naturally proved by showing inductively that the interpretation function maps every derivable syntactic statement to a true semantic statement, which is exactly what a signature morphism does, while a completeness result is usually proved by exhibiting a canonical model, which has a very different flavour.

However, our framework should still facilitate the carrying out of completeness proofs since the canonical model is usually formed from the syntax, which is already formally represented in Σ^{syn}. A framework like LF is ideal for building canonical models as abstract data types. However, to show completeness, we still have to reflect these LF-level syntactical models into Σ^{mod}, for example, by interpreting an LF type as the set of LF terms of that type. This technique is related to the definition of abstract data types in Isabelle/HOL (Nipkow *et al.* 2002) and the quotations of Chiron (Farmer 2010). We will leave further investigation of this to future work.

4.4.2. *Theories.* Because of the Curry–Howard representation of proofs, there is no significant conceptual difference between signatures and theories from an LF perspective. In fact, $\mathbb{M}$-signatures already subsume the finite theories.

Notation 4.27. If $\Theta = \{F_1, \ldots, F_n\}$, the $\mathbb{M}$-theory (Σ, Θ) behaves in the same way as the $\mathbb{M}$-signature arising from Σ by appending $a_1 : base(\mathtt{ded})\ F_1$, $\ldots$, $a_n : base(\mathtt{ded})\ F_n$ to Σ^{syn}. We write $(\Sigma, \Theta)^{syn}$ for this signature. Accordingly, we obtain (by pushout along $\Sigma^{syn} \to (\Sigma, \Theta)^{syn}$) the signatures $(\Sigma, \Theta)^{pf}$ and $(\Sigma, \Theta)^{mod}$.

Correspondingly, we can express $\mathbb{M}$-theory morphisms as LF-signature morphisms.

Theorem 4.28. Assume an $\mathbb{M}$-signature morphism $\sigma : \Sigma \to \Sigma'$ and finite theories (Σ, Θ) and (Σ', Θ').

(1) The following properties of σ are equivalent:

— σ is a proof-theoretical theory morphism $(\Sigma, \Theta) \to (\Sigma', \Theta')$.

— There is an LF-signature morphism $(\sigma, \vartheta)^{pf}$ such that the following diagram commutes:

$$\begin{tikzcd}
\Sigma^{pf} \arrow[r, "\sigma^{pf}"] \arrow[d, hook] & \Sigma'^{pf} \arrow[d, hook] \\
(\Sigma, \Theta)^{pf} \arrow[r, "(\sigma, \vartheta)^{pf}"] & (\Sigma', \Theta')^{pf}
\end{tikzcd}$$

(2) The following properties of σ are equivalent:

— σ is a model-theoretical theory morphism $(\Sigma, \Theta) \to (\Sigma', \Theta')$.

— There is an LF-signature morphism $(\sigma, \vartheta)^{mod}$ such that the following diagram commutes:

$$\begin{tikzcd}
\Sigma^{mod} \arrow[r, "\sigma^{mod}"] \arrow[d, hook] & \Sigma'^{mod} \arrow[d, hook] \\
(\Sigma, \Theta)^{mod} \arrow[r, "(\sigma, \vartheta)^{mod}"] & (\Sigma', \Theta')^{mod}
\end{tikzcd}$$

Proof.

(1) The fact that σ is a proof-theoretical theory morphism is equivalent to there being proof terms $p_i : \sigma^{syn}(\pi(base'(\mathtt{ded})\ F_i))$ over the LF-signature $(\Sigma', \Theta')^{pf}$ for every $F_i \in \Theta$, that is, for every $a_i : base(\mathtt{ded})\ F_i$ in $(\Sigma, \Theta)^{syn}$. Then, mapping every a_i to p_i yields the LF-signature morphism $(\sigma, \vartheta)^{pf} : (\Sigma, \Theta)^{pf} \to (\Sigma', \Theta')^{pf}$. The inverse direction is proved similarly.

(2) Note that from the definition of satisfaction, a Σ-model (M, m) satisfies all axioms in Θ if and only if $m : \Sigma^{mod} \to M$ can be factored through $(\Sigma, \Theta)^{mod}$ (in other words, m can be extended to a signature morphism $(\Sigma, \Theta)^{mod}$).
Thus, given $(\sigma, \vartheta)^{mod}$, models of Σ' yield models of Σ simply by composition, which proves σ is a model-theoretical theory morphism. Conversely, we get $(\sigma, \vartheta)^{mod}$ by factoring the Σ-model $((\Sigma', \Theta')^{mod}, \sigma^{mod})$ through $(\Sigma, \Theta)^{mod}$. □

The criterion for proof-theoretical theory morphisms is quite intuitive and useful: the Curry–Howard representation is used to map axioms to proof terms. The model-theoretical criterion is much less useful because there are many models in $\mathbb{M}$, which makes it a very strong condition. It guarantees that σ is a model-theoretical theory morphism for any foundation. However, the property of being a model-theoretical theory morphism often depends on the chosen foundation. In Section 4.5, we will look at a weaker, foundation-specific condition.

4.5. *Foundations*

We say $\mathbb{M}$ is *foundation-independent* because it is not committed to a fixed foundation in which the model theory of object logics is defined. Thus, we are able to express logics using different mathematical foundations and even translations between them. However, this is also a drawback because, for any specific logic, we are usually only interested in certain models (M, m), namely in those where M is the semantic realm of all mathematical objects. Therefore, we restrict $\mathbb{M}$ accordingly using a fixed LF-signature $\mathcal{F}$ that represents the foundation of mathematics.

The intuition is as follows. $\mathcal{F}$ is a meta-language, which is used by Σ^{mod} to specify models. Thus, Σ^{mod} includes $\mathcal{F}$ and adds symbols to it that represent the free parameters of a model, for example, the universe and the function and predicate symbols for first-order models. Models must interpret these added symbols as expressions of the foundation in a way that preserves typing (and thus, through Curry–Howard, in a way that satisfies the axioms). Therefore, we can represent models as signature morphisms $m : \Sigma^{mod} \to \mathcal{F}$.

However, representing a foundation in LF can be cumbersome due to the size of the theory involved. The representation of ZFC we used in Horozal and Rabe (2011) required several thousand declarations just to build up the basic notions of ZFC set theory to a degree that supports simple examples. Moreover, the automation of foundations is a very difficult problem, and type checking or automated reasoning for untyped set theories is much harder than for most logics.

We can remedy these drawbacks by taking a slightly more general approach. We use two LF-signatures $\mathcal{F}_0$ and $\mathcal{F}$ with a morphism $P : \mathcal{F}_0 \to \mathcal{F}$. The idea is that $\mathcal{F}_0$ is a fragment or an approximation of the foundation $\mathcal{F}$ that is refined into $\mathcal{F}$ using the morphism P. It turns out that manageably simple choices of $\mathcal{F}_0$ are expressive enough to represent the model theory of most logics.

This has the additional benefit that the same $\mathcal{F}_0$ can be used together with different values for P and $\mathcal{F}$. This corresponds to the mathematical practice of not detailing the foundation unless necessary. For example, standard constructions and results like the real numbers are usually assumed without giving or relying on a specific definition. For example, in Horozal and Rabe (2011), we use a simply typed higher-order logic for $\mathcal{F}_0$ and ZFC set theory as $\mathcal{F}$, and P interprets types as sets and terms as elements of these sets. $\mathcal{F}_0$ can be written down in LF within a few minutes, and strong implementations are available (Gordon 1988; Nipkow *et al.* 2002; Harrison 1996).

If we combine this approach with a module system for LF – as we do, for example, in Codescu *et al.* (2011) – this gives rise to the paradigm of *little foundations*. It corresponds to the little theories used implicitly in Bourbaki (1974) and explicitly in Farmer *et al.* (1992). Using little foundations, we can specify models in $\mathcal{F}_0$ making as few foundational assumptions as possible and refine it to a stronger foundation when needed. In fact, Σ^{syn} can be viewed as the extreme case of making no assumptions about the foundation, and $\mu : \Sigma^{syn} \to \Sigma^{mod}$ as a first refinement step. At the opposite end of the refinement chain, we might find the extension $ZF \hookrightarrow ZFC$, which adds the axiom of choice.

We formalise these intuitions as follows.

Definition 4.29 (foundations). Given an LF-signature $\mathcal{F}_0$, we define $\mathbf{Sig}^{\mathbb{M}}_{\mathcal{F}_0}$ to be the subcategory of $\mathbf{Sig}^{\mathbb{M}}$ that is restricted to:
— those signatures Σ for which there is an inclusion from $\mathcal{F}_0$ into Σ^{mod};
— those signature morphisms σ for which σ^{mod} is the identity on $\mathcal{F}_0$.

Thus, $\mathbf{Sig}^{\mathbb{M}}_{\mathcal{F}_0}$ is the full subcategory of inclusions of the slice category of objects under $\mathcal{F}_0$. Note that we recover $\mathbf{Sig}^{\mathbb{M}}$ when $\mathcal{F}_0$ is the empty signature.

Definition 4.30 (founded models). For an LF-signature morphism $P : \mathcal{F}_0 \to \mathcal{F}$, we define $\mathbf{Mod}^{\mathbb{M}}_P$ as the functor $\mathbf{Sig}^{\mathbb{M}}_{\mathcal{F}_0} \to \mathcal{CAT}^{op}$ that maps:
— signatures Σ to the subcategory of $\mathbf{Mod}^{\mathbb{M}}(\Sigma)$ restricted to the models $(\mathcal{F}, m)$ for which m agrees with P on $\mathcal{F}_0$;
— signature morphisms in the same way as $\mathbf{Mod}^{\mathbb{M}}$.

These definitions lead to the following commutative diagram for a Σ'-model $(\mathcal{F}, m)$ translated along σ:

$$\begin{tikzcd}
\Sigma^{mod} \arrow[rr, "\sigma^{mod}"] \arrow[ddr, "m \circ \sigma^{mod}"'] & & \Sigma'^{mod} \arrow[ddl, "m"] \\
& \mathcal{F}_0 \arrow[ul, hook'] \arrow[ur, hook] \arrow[d, "P"'] & \\
& \mathcal{F} &
\end{tikzcd}$$

And these specialisations do indeed yield a logic.

Theorem 4.31. For arbitrary $P : \mathcal{F}_0 \to \mathcal{F}$, we obtain a logic

$$\mathbb{M}_P = \left(\mathbf{Sig}^{\mathbb{M}}_{\mathcal{F}_0}, \mathbf{Sen}^{\mathbb{M}}, \mathbf{Mod}^{\mathbb{M}}_P, \models^{\mathbb{M}}, \mathbf{Pf}^{\mathbb{M}}, \vdash^{\mathbb{M}}\right)$$

(where $\mathbf{Sen}^{\mathbb{M}}$, $\models^{\mathbb{M}}$, $\mathbf{Pf}^{\mathbb{M}}$, and $\vdash^{\mathbb{M}}$ are the appropriate restrictions according to $\mathbf{Sig}^{\mathbb{M}}_{\mathcal{F}_0}$ and $\mathbf{Mod}^{\mathbb{M}}_P$).

Proof. This follows immediately from Theorem 4.23. The only non-trivial aspect is to show that $\mathbf{Mod}^{\mathbb{M}}_P(\sigma)$ is well defined, which follows from the commutativity of the above diagram. □

Example 4.32 (modal logic continued). We have already used $\mathbb{M}_P$-models in Example 4.15, where we used $\mathcal{F}_0 = \mathcal{F} = HOL$ and P was the identity.

We can now revisit Theorem 4.28, which stated that model-theoretical theory morphisms

$$\sigma : (\Sigma, \Theta) \to (\Sigma', \Theta')$$

in $\mathbb{M}$ are equivalent to LF-signature morphisms

$$(\sigma, \vartheta)^{mod} : (\Sigma, \Theta)^{mod} \to (\Sigma', \Theta')^{mod}.$$

However, if we restrict attention to $\mathbb{M}_P$, the situation is a bit more complicated.

Theorem 4.33. Given an $\mathbb{M}_P$-signature morphism $\sigma : \Sigma \to \Sigma'$ and finite theories (Σ, Θ) and (Σ', Θ'), the following properties of σ are equivalent:
— σ is a model-theoretical theory morphism $(\Sigma, \Theta) \to (\Sigma', \Theta')$ in the logic $\mathbb{M}_P$.

— For every LF-signature morphism $m' : (\Sigma', \Theta')^{mod} \to \mathcal{F}$, there is an LF-signature morphism $m : (\Sigma, \Theta)^{mod} \to \mathcal{F}$ such that the following diagram commutes:

$$\begin{tikzcd}
\Sigma^{mod} \arrow[r, "\sigma^{mod}"] \arrow[d, hook] & \Sigma'^{mod} \arrow[d, hook] \\
(\Sigma, \Theta)^{mod} \arrow[dr, "m"'] & (\Sigma', \Theta')^{mod} \arrow[d, "m'"] \\
 & \mathcal{F}
\end{tikzcd}$$

Proof. The statement follows immediately because σ is a model-theoretical theory morphism if and only if $\mathbf{Mod}_P^{\mathbb{M}}(\sigma)$ maps Θ'-models to Θ-models. □

Using Theorem 4.33, extending σ^{mod} to an LF-signature morphism

$$(\sigma, \vartheta)^{mod} : (\Sigma, \Theta)^{mod} \to (\Sigma', \Theta')^{mod}$$

is sufficient to make σ a model-theoretical $\mathbb{M}_P$-theory morphism. But it is not a necessary condition. Counter-examples arise when $\mathcal{F}_0$ is too weak and cannot prove all theorems of $\mathcal{F}$. Horozal and Rabe (2011) showed that it is a necessary condition in the special case $\mathcal{F}_0 = \mathcal{F} = ZFC$, but the general case is open.

5. Defining and encoding logics

Innthis section we combine the results of Sections 3 and 4 to define logics and logic translations in our framework. Logics are defined or represented using functors $\Phi : \mathbf{Sig} \to \mathbf{Sig}^{\mathbb{M}}$, but the formal details can be cumbersome in practice: $\mathbb{M}$-signatures are 6-tuples and morphisms are 3-tuples with some commutativity constraints. Because of this, we introduce the notion of uniform logic representations, which are the typical situation in practice and can be given conveniently.

Uniform logics are generated by a fixed $\mathbf{Sig}^{\mathbb{M}}$ signature L. The intuition is that L^{syn} represents the logical symbols, and extensions $L^{syn} \hookrightarrow \Sigma^{syn}$ add one declaration for each non-logical symbols. This is very similar to the use of 'uniform' in Harper *et al.* (1994), from which we borrow the name. Proof and model theory are then given in general by L^{pf} and L^{mod}, and these induce extensions Σ^{pf} and Σ^{mod} for the proof and model theory for a specific choice of non-logical symbols. Similarly, translations interpret logical symbols in terms of logical symbols and non-logical ones in terms of non-logical ones.

5.1. *Non-logical symbols*

Recall from Section 2.2 that we write $\mathbb{LF}$ for the category of LF signatures and LF signature morphisms. Because $\mathbf{Sig}^{\mathbb{M}}$ morphisms are triples of $\mathbb{LF}$ morphisms and composition is defined component-wise, $\mathbf{Sig}^{\mathbb{M}}$ inherits inclusions and pushouts from $\mathbb{LF}$. In particular, we have the following lemma.

Lemma 5.1. $\mathbf{Sig}^{\mathbb{M}}$ has inclusion morphisms in the sense that the subcategory of $\mathbf{Sig}^{\mathbb{M}}$ containing

— all objects; and
— only those morphisms $(\sigma^{syn}, \sigma^{pf}, \sigma^{mod}) : \Sigma \to \Sigma'$ for which σ^{syn}, σ^{pf} and σ^{mod} are inclusions in $\mathbb{LF}$

is a partial order. We write $\Sigma \hookrightarrow \Sigma'$ for inclusion morphisms from Σ to Σ'.

Proof. Morphisms in $\mathbf{Sig}^{\mathbb{M}}$ are triples, and composition is defined component-wise, so the properties of inclusions follow immediately from those of inclusions in $\mathbb{LF}$. $\square$

Remark 5.2. More generally, the inclusions of $\mathbb{LF}$ induce a weak inclusion system in the sense of Căzănescu and Roşu (1997). This property is also inherited by $\mathbf{Sig}^{\mathbb{M}}$.

We will use inclusions $L \hookrightarrow \Sigma$ to represent a logic L together with non-logical symbols given by Σ. When giving the non-logical symbols, it is not necessary to give Σ; instead, we can just give an $\mathbb{LF}$ inclusion $L^{syn} \hookrightarrow \Sigma^{syn}$ and obtain the remaining components of Σ by pushout constructions. We make this precise in the following definitions.

Definition 5.3. An inclusion $L^{syn} \hookrightarrow \Sigma^{syn}$ is said to be *compatible* with L if Σ^{syn} does not add declarations for names that are also declared in L^{pf} or L^{mod}.

Definition 5.4. We define the category $\mathbb{M} + \mathbb{LF}$ as follows:

— The objects are the pairs (L, Σ^{syn}) of an $\mathbb{M}$ signature L and an $\mathbb{LF}$ inclusion $L^{syn} \hookrightarrow \Sigma^{syn}$ that is compatible with L.
— The morphisms from (L, Σ^{syn}) to (L', Σ'^{syn}) are the commuting rectangles

$$\begin{tikzcd}
L'^{syn} \arrow[r, hook] & \Sigma'^{syn} \\
L^{syn} \arrow[u, "l^{syn}"] \arrow[r, hook] & \Sigma^{syn} \arrow[u, "\sigma^{syn}"']
\end{tikzcd}$$

The restriction to compatible inclusions in Definition 5.4 is a technicality needed for Definition 5.5: if Σ^{syn} added a name that is also declared in L^{pf} or L^{mod}, the respective canonical pushout in $\mathbb{LF}$ would not exist. This restriction is harmless in practice because namespaces can be used to ensure the uniqueness of names.

For compatible inclusions, the canonical pushouts exist and every $\mathbb{M} + \mathbb{LF}$ object (L, Σ^{syn}) induces an $\mathbb{M}$ signature and every $\mathbb{M} + \mathbb{LF}$ morphism (l, σ^{syn}) induces an $\mathbb{M}$ signature morphism.

Definition 5.5. We define a functor $\overline{\cdot} : \mathbb{M} + \mathbb{LF} \to \mathbf{Sig}^{\mathbb{M}}$ as follows:

— For objects (L, Σ^{syn}), we define the $\mathbb{M}$ signature

$$\overline{(L, \Sigma^{syn})} = (\Sigma^{syn}, \Sigma^{pf}, \Sigma^{mod}, base_{\Sigma}, \pi_{\Sigma}, \mu_{\Sigma})$$

such that the following diagram commutes and Σ^{pf} and Σ^{mod} are canonical pushouts in $\mathbb{LF}$:

$$\begin{tikzcd}
 & & L^{pf} \arrow[rr, hook] & & \Sigma^{pf} \\
Base \arrow[r, "base"] \arrow[rrr, bend right, "base_\Sigma"'] & L^{syn} \arrow[ur, "\pi"] \arrow[dr, "\mu"'] \arrow[rr, hook] & & \Sigma^{syn} \arrow[ur, "\pi_\Sigma"'] \arrow[dr, "\mu_\Sigma"] & \\
 & & L^{mod} \arrow[rr, hook] & & \Sigma^{mod}
\end{tikzcd}$$

— For morphisms

$$(l, \sigma^{syn}) : (L, \Sigma^{syn}) \to (L', \Sigma'^{syn}),$$

we define the $\mathbb{M}$ signature morphism

$$\overline{(l, \sigma^{syn})} = (\sigma^{syn}, \sigma^{pf}, \sigma^{mod}) : \overline{(L, \Sigma^{syn})} \to \overline{(L', \Sigma'^{syn})}$$

such that σ^{pf} and σ^{mod} are the unique morphisms that make the following $\mathbb{LF}$ diagrams commute:

$$\begin{tikzcd}
 & L'^{pf} \arrow[rr, hook] & & \Sigma'^{pf} \\
L'^{syn} \arrow[ur, "\pi'"] \arrow[rr, hook] & & \Sigma'^{syn} \arrow[ur, "\pi_{\Sigma'}"] & \\
 & L^{pf} \arrow[uu, "l^{pf}"] \arrow[rr, hook] & & \Sigma^{pf} \arrow[uu, "\sigma^{pf}"'] \\
L^{syn} \arrow[uu, "l^{syn}"] \arrow[ur, "\pi"'] \arrow[rr, hook] & & \Sigma^{syn} \arrow[uu, "\sigma^{syn}"'] \arrow[ur, "\pi_\Sigma"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & L'^{mod} \arrow[rr, hook] & & \Sigma'^{mod} \\
L'^{syn} \arrow[ur, "\mu'"] \arrow[rr, hook] & & \Sigma'^{syn} \arrow[ur, "\mu_{\Sigma'}"] & \\
 & L^{mod} \arrow[uu, "l^{mod}"] \arrow[rr, hook] & & \Sigma^{mod} \arrow[uu, "\sigma^{mod}"'] \\
L^{syn} \arrow[uu, "l^{syn}"] \arrow[ur, "\mu"'] \arrow[rr, hook] & & \Sigma^{syn} \arrow[uu, "\sigma^{syn}"'] \arrow[ur, "\mu_\Sigma"'] &
\end{tikzcd}$$

5.2. *Uniform logics*

We obtain a uniform logic $\mathbb{M}_P^L$ for every $\mathbb{M}$ signature L by pairing L with all possible choices of non-logical symbols. In $\mathbb{M}_P^L$, we also fix a foundation for the model theory.

Definition 5.6 (uniform logics). Given an LF signature morphism $P : \mathcal{F}_0 \to \mathcal{F}$ and a $\mathbf{Sig}^{\mathbb{M}}_{\mathcal{F}_0}$-signature L, and writing $L + \mathbb{LF}$ for the subcategory of $\mathbb{M} + \mathbb{LF}$ containing the objects (L, Σ^{syn}) and the morphisms (id_L, σ^{syn}), we obtain a logic

$$\mathbb{M}_P^L = \mathbb{M}_P \circ \overline{\cdot}|_{L+\mathbb{LF}},$$

where

$$\overline{\cdot}|_{L+\mathbb{LF}} : L + \mathbb{LF} \to \mathbf{Sig}^{\mathbb{M}}$$

is the restriction of $\overline{\cdot}$ to $L + \mathbb{LF}$.

Uniform logics are defined completely within LF – signatures, sentences, proofs and models are given as syntactical entities of LF. We already know how to represent existing logics in LF, namely using Theorem 3.28. Uniform logic encodings arise when we use $\mathbb{M}_P^L$ as the meta-logic.

Example 5.7 (modal logic continued). Taking all the examples from Section 4 together, we obtain a uniform encoding of $\mathbb{ML}$ using the $\mathbb{M}$-signature

$$ML = (ML^{syn}, ML^{pf}, ML^{mod}, incl, incl, \mu_{ML})$$

(where *incl* denotes inclusion morphisms). We define a functor

$$\Phi^{\mathbb{ML}} : \mathbf{Sig}^{\mathbb{ML}} \to ML + \mathbb{LF}$$

as follows. For a modal logic signature

$$\Sigma = \{p_1, \ldots, p_n\} \in |\mathbf{Sig}^{\mathbb{ML}}|$$

and a signature morphism

$$\sigma : \Sigma \to \Sigma',$$

we define

$$\Phi^{\mathbb{ML}}(\Sigma) = (ML, \Sigma^{syn})$$
$$\Phi^{\mathbb{ML}}(\sigma) = (id_{,}\sigma^{syn})$$

with

$$\Sigma^{syn} = ML^{syn},\ p_1 : \mathtt{form},\ \ldots,\ p_n : \mathtt{form}$$
$$\sigma^{syn} = id_{ML^{syn}},\ p_1 := \sigma(p_1),\ \ldots,\ p_n := \sigma(p_n)$$

This yields

$$\overline{\Phi^{\mathbb{ML}}(\Sigma)} = (\Sigma^{syn}, \Sigma^{pf}, \Sigma^{mod}, incl, incl, \mu_\Sigma)$$
$$\overline{\Phi^{\mathbb{ML}}(\sigma)} = (\sigma^{syn}, \sigma^{pf}, \sigma^{mod})$$

with

$$\begin{aligned}
\Sigma^{pf} &= ML^{pf},\ p_1 : \mathtt{form},\ \ldots,\ p_n : \mathtt{form}\\
\Sigma^{mod} &= ML^{mod},\ p_1 : tm\ worlds \to tm\ bool,\ \ldots,\ p_n : tm\ worlds \to tm\ bool\\
\mu_\Sigma &= \mu_{ML},\ p_1 := p_1,\ \ldots,\ p_n := p_n\\
\sigma^{pf} &= id_{ML^{pf}},\ p_1 := \sigma(p_1),\ \ldots,\ p_n := \sigma(p_n)\\
\sigma^{mod} &= id_{ML^{mod}},\ p_1 := \sigma(p_1),\ \ldots,\ p_n := \sigma(p_n).
\end{aligned}$$

In particular, we have

$$\overline{\Phi^{\mathbb{ML}}(PQ)} = \left(PQ^{syn}, PQ^{pf}, PQ^{mod}, incl, incl, \mu_{PQ}\right)$$
$$\overline{\Phi^{\mathbb{ML}}(w)} = \left(w^{syn}, w^{pf}, w^{mod}\right)$$

as defined throughout the running example.

Moreover, it is straightforward to show that $\Phi^{\mathbb{ML}}$ can be extended to a logic comorphism $(\Phi^{\mathbb{ML}}, \alpha, \beta, \gamma)$ from $\mathbb{ML}$ to $\mathbb{M}_P^{ML}$ where P is as in Example 4.32, where α_Σ is an obvious bijection, β_Σ maps every LF-signature morphism $m : \Sigma^{mod} \to HOL$ to a Kripke-model of Σ and γ_Σ is a straightforward encoding of judgments as types and proofs as terms.

It is very easy to show the proof-theoretical adequacy of γ. The discussion of the model-theoretical adequacy of β is complicated as it depends on which sets are eligible as Kripke frames in ML-models (see Remark 5.9). If HOL can express all sets that are eligible as Kripke frames, we obtain model-theoretical adequacy.

Remark 5.8 (declaration patterns). Definition 5.6 uses all extensions of L^{syn} as the signatures of $\mathbb{M}_P^L$. However, we often want to work with a subcategory instead. For example, for modal logic, we would like to use only those extensions with declarations of the form p : `form`. This requires an extension of LF with what we call *declaration patterns* (similar to the *block* declarations already present in Twelf (Pfenning and Schürmann 1999)). Such an extension is currently designed in joint work with Fulya Horozal.

Remark 5.9 (model-theoretical adequacy). Example 5.7 raises the question of when model-theoretical adequacy holds. Let us call a mathematical object *definable* if it can be denoted by an expression of the foundational language. For example, since there can only be countably many definable objects, a set theory with uncountably many objects has undefinable objects. But even if we can prove the existence of undefinable objects, we can never actually give one. On the other hand, it is easy to see that a model can be represented as an LF signature morphism if and only if all its components are definable.

Therefore, the surjectivity of β_Σ, and thus model-theoretical adequacy, depends on the philosophical point of view we take.

The type theoretical school often rejects a commitment to set theory – recall that LF, and thus the essence of $\mathbb{M}$, can be developed using only formal languages and inference systems without appealing to a foundation of mathematics like set theory. Type theory does not object to model theory *per se*; but it would restrict attention to definable models. In fact, our models-as-signature-morphisms paradigm is an appealing way to define *models* from a type theoretical point of view. This is particularly elegant if we observe that the approach unifies the study of models of theories and that of implementations of specifications. In that case, of course, all models are expressible as signature morphisms, and adequacy holds.

The set theoretical school does not only define individual models but also the class of all models. Moreover, they see the sets of LF signatures and signature morphisms as defined within set theory. Then adequacy means the existence of a certain mapping between a class of models and a set of morphisms. But then a simple cardinality argument shows that adequacy usually does not hold. In order to maintain Theorem 3.30, we have to study weaker notions of adequacy. We leave this to further work.

Besides their simplicity, uniform representations have the advantage that the properties of soundness can be proved formally within the logical framework:

Theorem 5.10. A uniform logic $\mathbb{M}_P^L$ is strongly sound if L is a sound $\mathbb{M}$ signature.

Proof. Using Theorem 4.25, we only have to show that for every Σ^{syn} extending L^{syn} there is an $\mathbb{LF}$-morphism from Σ^{pf} to Σ^{mod} with a certain commutativity property. This follows immediately from the universal property of the pushout Σ^{pf} using the soundness of L. □

In order to give an example of a logic translation later (see Example 5.13), we will now sketch an example of another logic, which consists of a fragment of our encoding of first-order logic in Horozal and Rabe (2011).

Example 5.11 (first-order logic). To define first-order logic, $\mathbb{FOL}$, we give a uniform functor $\Phi^{\mathbb{FOL}}$ over the $\mathbb{M}$ signature FOL. The central declarations of FOL^{syn} are:

i	:	$\mathtt{type}$	terms
o	:	$\mathtt{type}$	formulas
$\supset$	:	$o \to o \to o$	implication
$\forall$	:	$(i \to o) \to o$	universal
ded	:	$o \to \mathtt{type}$	truth

FOL^{pf} adds natural deduction proof rules in a straightforward way. For simplicity, we assume that FOL^{mod} extends the same signature $\mathcal{F}_0 = HOL$ as ML^{mod}. This extension includes a declaration $univ : tp$ for the universe. $base_{FOL}$ and π_{FOL} are inclusions, and μ_{FOL} maps i to $tm\ univ$, o to $tm\ bool$ and all formulas to their semantics in the usual way.

We assume a $\mathbb{FOL}$ signature $\Sigma = (\Sigma_f, \Sigma_p, arit)$ where Σ_f and Σ_p are the sets of function and predicate symbols with arity function $arit : \Sigma_f \cup \Sigma_p \to \mathbb{N}$. Then $\Phi^{\mathbb{FOL}}(\Sigma)$ extends FOL^{syn} with declarations $f : i \to \ldots \to i \to i$ and $p : i \to \ldots \to i \to o$, respectively, for each function or predicate symbol.

5.3. *Uniform logic translations*

Logic translations between uniform logics are already possible using the general methods of Theorem 3.34. We refer to uniform translations in the special case where a fixed $\mathbb{M}$-signature morphism is used to translate the logical symbols.

Theorem 5.12 (uniform logic translations). For $i = 1, 2$, we assume functors

$$\Phi^i : \mathbf{Sig}^i \to L_i + \mathbb{LF},$$

together with:

— a functor $\Phi : \mathbf{Sig}^1 \to \mathbf{Sig}^2$;

— a $\mathbf{Sig}^{\mathbb{M}}_{\mathcal{F}_0}$-morphism $l = (l^{syn}, l^{pf}, l^{mod}) : L_1 \to L_2$;

— a natural transformation $m : \Phi^1 \to \Phi^2 \circ \Phi$ (seen as functors $\mathbf{Sig}^1 \to \mathbb{M} + \mathbb{LF}$) such that each m_Σ is of the form (l, α_Σ).

Then $(\varphi, \mathbb{M}_P \circ \overline{\cdot} \circ m)$ is a logic comorphism from $\mathbb{M}_P^{L_1} \circ \Phi^1$ to $\mathbb{M}_P^{L_2} \circ \Phi^2$.

Proof. Observe that $\overline{\cdot} \circ m$ is a natural transformation $\overline{\cdot} \circ \Phi^1 \to \overline{\cdot} \circ \Phi^2 \circ \Phi$ between functors $\mathbf{Sig}^1 \to \mathbf{Sig}^{\mathbb{M}}$, and also that $\mathbb{M}_P^{L_i} \circ \Phi^i = \mathbb{M}_P \circ \overline{\cdot} \circ \Phi^i$. The result then follows immediately from Theorem 3.34. □

Finally, we can give a uniform logic translation from modal logic to first-order logic.

Example 5.13 (translating ML to FOL). Using Examples 5.7 and 5.11, we can represent the well-known translation from modal logic to first-order logic that makes worlds explicit and relativises quantifiers:

(1) The functor $\Phi : \mathbf{Sig}^{\mathbb{ML}} \to \mathbf{Sig}^{\mathbb{FOL}}$ maps every modal logic signature Σ to the $\mathbb{FOL}$ signature that contains a binary predicate symbol acc (for the accessibility relation) and one unary predicate symbol p for every propositional variable $p \in \Sigma$.

(2) We cannot give an $\mathbb{M}$ signature morphism $ML \to FOL$ because the translation requires the fixed predicate symbol acc, which is present in every $\Phi(\Sigma)$ but not in FOL^{syn}, so we put $L_1 = ML$ and $L_2 = FOL' = \overline{(FOL, \Sigma_0)}$ where Σ_0 is the LF signature

$$FOL^{syn},\ acc : i \to i \to o.$$

Then $\Phi^{\mathbb{FOL}} \circ \Phi$ is indeed a functor

$$\mathbf{Sig}^{\mathbb{ML}} \to FOL' + \mathbb{LF}.$$

(3) We now define a $\mathbf{Sig}^{\mathbb{M}}$ morphism $l : ML \to FOL'$:
l^{syn} is given by

$$\begin{aligned}
&\mathtt{form} &&:= i \to o\\
&\supset &&:= [f : i \to o]\,[g : i \to o]\,[x : i]\,(f\ x) \supset (g\ x)\\
&\Box &&:= [f : i \to o]\,[x : i]\,\forall[y : i]\,((acc\ x\ y) \supset (f\ y))\\
&\mathtt{ded} &&:= [f : i \to o]\,ded\ \forall[x : i]\,f\ x
\end{aligned}$$

ML-sentences are mapped to FOL-formulas with a free variable (that is, terms of type $i \to o$). The intuition is that the free variable represents the world in which the ML-sentence is evaluated. $\supset$ and $\Box$ are translated in the expected way. Finally, the ML-truth judgment $\mathtt{ded}\ F$ is mapped to the FOL-judgment $ded\ \forall[x]\, l^{syn}(F)\ x$; note how $\forall$ is used to bind the above-mentioned free variable and to obtain a sentence.
l^{pf} extends l^{syn} with assignments that map every proof rule of modal logic to a proof in first-order logic. These are technical but straightforward. l^{mod} is the identity for all symbols of HOL and maps

$$\begin{aligned}
&worlds &&:= univ\\
&acc &&:= acc
\end{aligned}$$

It is easy to establish the commutativity requirements on $(l^{syn}, l^{pf}, l^{mod})$.

(4) Finally, we define the natural transformation m:
m_Σ must be an $\mathbb{M} + \mathbb{LF}$ morphism (l, α_Σ) from $\Phi^{\mathbb{FOL}}(\Phi(\Sigma))$ to $\Phi^{\mathbb{ML}}(\Sigma)$. For $\Sigma = \{p_1, \ldots, p_n\}$, the domain of α_Σ is

$$ML^{syn}, p_1 : \mathtt{form},\ \ldots,\ p_n : \mathtt{form},$$

and the codomain is

$$FOL^{syn},\ acc : i \to i \to o,\ p_1 : i \to o,\ \ldots,\ p_n : i \to o.$$

Thus, we put $\alpha_\Sigma = l^{syn},\ p_1 := p_1,\ \ldots,\ p_n := p_n$.

Remark 5.14. In the typical case where Φ^1 and Φ^2 are injective, we can modify Definition 5.12 to use a functor

$$\Phi' : L_1 + \mathbb{LF} \to L_2 + \mathbb{LF}$$

such that

$$\Phi^2 \circ \Phi = \Phi' \circ \Phi^1$$

represents the same signature translation as Φ but lives within the meta-logic $\mathbb{M}$.

Then, in the case covered in Example 5.13, we have Φ' and the natural transformation m are given simply by pushout along l. This yields a significantly simpler notion of a uniform logic translation induced by l alone. However, not all signature translations can be obtained in this way. A counter-example is the translation from sorted to unsorted FOL that translates single declarations in Σ to multiple declarations in $\Phi(\Sigma)$.

In general, Φ' and m can be formalised using the declaration patterns from Remark 5.8, but we leave the details to future work.

Remark 5.15 (non-simple morphisms). Note that l in Example 5.13 is not simple, which finally justifies the particular choice in Definition 4.4 explained in Remark 4.5. The induced sentence translation $\mathbf{Sen}^{\mathbb{M}}(l)$ can be factored into two steps:

(1) a compositional translation mapping F of type `form` to $l^{syn}(F)$ of type $i \to o$;
(2) a non-compositional step that is only applied once at top level maps $l^{syn}(F)$ to $\forall[x]\, l^{syn}(F)\, x$ of type o.

This is typical for non-trivial logic translations: if the semantics of $\mathbb{I}_1$ is encoded using the syntax of $\mathbb{I}_2$, then $\mathbb{I}_1$-sentences are translated to some $\mathbb{I}_2$-expressions, which must be lifted to $\mathbb{I}_2$-sentences using a final top-level step. Another example is the translation of many-valued propositional logic to first-order logic, where formulas are translated compositionally to terms and a final top-level step applies a unary predicate for designated truth values.

More generally, we can distinguish two kinds of sentence translation functions:

— *within-logic* translations $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma)$ translate along a signature morphism between two signatures of the same logic;
— *across-logic* translations $\alpha_\Sigma : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}'(\Phi(\Sigma))$ translate along a logic translation between two signatures of different logics.

The corresponding observation also applies to model and proof translations.

Within-logic translations are typically straightforward homomorphic mappings that can be represented as simple $\mathbf{Sig}^{\mathbb{M}}$-morphisms, but across-logic translations may involve complex encoding steps that may not be easily expressible using homomorphic mappings. When using a Grothendieck institution as in Diaconescu (2002), within-logic and across-logic translations are unified, and the latter are those that employ a non-trivial institution comorphism.

However, if we use a meta-logic like $\mathbb{M}$, both within- and across-logic translations must be induced by $\mathbf{Sig}^{\mathbb{M}}$-morphisms. Therefore, we need our more general definition of $\mathbf{Sig}^{\mathbb{M}}$-morphisms that includes non-simple ones. This problem is a consequence of using a meta-logic in which a limited formal language is used to express translations. The problem does not arise when using institutions (as we do in Section 3) because both within-logic and across-logic translations are represented as mappings and functors, which is the most general notion of a translation operation.

6. Discussion

Our basic motivation was to provide a logical framework in which both model-theoretical and proof-theoretical logics, as well as their combinations, can be formulated and studied. But a logical framework – or any framework for that matter – can only serve as an auxiliary device: it helps get work done, but does not do the work by itself. In fact, a good logical framework should be general and weak in the sense of being foundationally uncommitted (see also de Bruijn (1991)).

Therefore, to evaluate an individual framework, we must ask how it supports the solution of which concrete problems. More concretely, we must ask:

(i) what logics can be represented in it;
(ii) what applications become possible for represented logics;
(iii) whether the same could be achieved with other, more elegant frameworks.

We will discuss these questions in Section 6.1, 6.2 and 6.3, respectively.

6.1. *Coverage and limitations*

6.1.1. *Logics.* Our definitions are chosen carefully to subsume the frameworks of both institutions and LF so that existing logic representations can be reused or, when they only cover either proof or model theory, complemented. The available institution-based representations of model-theoretical logics (see, for example, Goguen and Burstall (1992), Mosses (2004), Borzyszkowski (2000) and Mossakowski *et al.* (2007)) become logic representations in our sense once their syntax and proof theory are made explicit in LF. Similarly, the available LF-based proof-theoretical logic representations (see, for example, Harper *et al.* (1993), Harper *et al.* (1994), Avron *et al.* (1992), Pfenning (2001), Pfenning (2000) and Avron *et al.* (1998)) become uniform logic definitions in our sense after encoding their model theory. In many cases, existing institution and LF-based representations of the same logic can be paired to complement each other.

Our definition of logic covers a wide range of logics including propositional, first-order and higher-order logics; logics based on untyped, simply typed, dependently typed and polymorphic type theories; modal, temporal and description logics; and logics with classical or intuitionistic semantics. Multiple truth values such as in three-valued logics are supported if some truth values are *designated* to obtain a two-valued satisfaction relation in the models (as we did in Goguen *et al.* (2007)), or if the syntax includes constants for the truth values.

The syntax of a logic can typically be expressed in our meta-logic $\mathbb{M}$. A limitation concerns logics where the notion of well-formed formulas is undecidable (such as in PVS (Owre *et al.* 1992)). This requires us to merge Σ^{syn} and Σ^{pf} to axiomatise well formedness and take a subtype of $base(\mathtt{form})$ as the type of sentences. This is possible with the more general framework we used in Rabe (2008), but we have omitted this here for simplicity.

Regarding proof theories, $\mathbb{M}$ inherits from LF the support of Hilbert, natural deduction, sequent and tableaux calculi. However, resolution-based proof theories cannot be represented as elegantly. Also, our notions of proof categories is biased against substructural logics – see Remark 3.8. This is a bias shared with both institutions and LF, the latter has even been extended to a linear variant for that reason (Cervesato and Pfenning 2002).

The representation of model theories in our meta-logic depends on two things:

(1) The foundation must be expressible in LF, which is typically the case.
(2) It must be adequate to represent models as morphisms; we discussed this in Remark 5.9.

Our logical framework allows us to express signatures, sentences, models and proofs of logics as LF objects. In addition, we have identified two extensions of LF that we need to reach a fully comprehensive framework:

(1) Logical relations should be developed and used to represent model morphisms (see Remark 4.21).
(2) A language of declaration patterns is needed to express the category of signatures as a whole, as opposed to individual signatures (see Remark 5.8).

6.1.2. *Logic translations.* For logic translations, the situation is more complicated. First, logic translations are less well classified than logics, which complicates a systematic study of which classes of logic translations are covered. Second, the representation of logic translations is significantly harder than for logics. In general, features present in the syntax translations that are covered express the model theory of one logic using the syntax of another, or code sentences of one logic as terms of another logic.

A sentence translation is covered if it is compositional up to a final top-level step as in Example 5.13.

A proof theory translation is covered if it translates inference rules to derivable rules. This excludes the non-compositional elimination of admissible rules such as cut elimination (where sentences and models are translated to themselves and proofs to cut-free proofs). Representing such translations requires a stronger notion of LF signature morphism that supports computation, for example, along the lines of Twelf's logic programs, Delphin (Poswolsky and Schürmann 2008) or Beluga (Pientka and Dunfield 2010). But the typing invariants of such signature morphisms would be a lot more complicated.

Model theory translations are typically covered if the model theories themselves are covered.

In Remark 5.15, we have already discussed the need for non-simple $\mathbf{Sig}^{\mathbb{M}}$-morphisms. We expect this to be a general phenomenon: while within-logic translations will always be represented by simple morphisms, the representation of more complex across-logic translations will require more complex notions of $\mathbf{Sig}^{\mathbb{M}}$-morphisms. Consequently, we expect future work to further generalise the definition of $\mathbf{Sig}^{\mathbb{M}}$-morphisms.

For example, some translations can only be encoded if the commutativity condition of Definition 4.4 is relaxed, especially for the lower rectangle dealing with the model translation. For example, the semantics of description logic interprets concepts as subsets of the universe. But the translation of description logic into first-order logic translates concepts to unary predicates, which the semantics of first-order logic interprets as functions from the universe to the booleans. Thus, the model translation commutes only up to the isomorphism between subsets and their characteristic functions. Recently, a weaker condition than commutativity was shown to be sufficient to obtain logic comorphisms in Sojakova (2010), namely, commutativity up to certain logical relations (see also Remark 4.21).

Finally there are some limitations of logic comorphisms, which are inherited from institution comorphisms:

— Logic comorphisms are limited to total translations. This excludes partial translations such as a translation from higher-order to first-order logic that is undefined for expressions containing λ. Such translations are important in practice when we want to borrow (semi-)automated theorem provers (for example, the use of first-order provers in Isabelle), but difficult to describe in a logical framework. To overcome this problem, we recently designed an extension of $\mathbb{LF}$ that supports partial signature morphisms (Dumbrava and Rabe 2010).

— Logic comorphisms separate the signature and the sentence translation. But there are applications of borrowing where a different signature translation is used for different conjectures over the same signature. For example, the Leo-II prover (Benzmüller *et al.* 2008) uses a translation from higher-order logic to sorted first-order logic by creating a new first-order sort for every function type that is mentioned in the higher-order conjecture. Such translations remain future work.

6.1.3. *Case studies.* In Rabe and Kohlhase (2011), we designed a generic module system based on theories and theory morphisms. In Rabe and Schürmann (2009), we extended the Twelf implementation of LF with the corresponding instance of this module system. We have used this implementation to conduct several large case studies where logics and logic translations are defined and machine checked efficiently.

A large number of case studies have been conducted in the LATIN project (Codescu *et al.* 2011). These include various incarnations of first-order logics, higher-order logics and modal and description logics, as well as a number of set and type theoretical foundations. These are written as modular LF signatures and morphisms, and the overall size of the atlas exceeds 1000 modules.

The most extensive case studies were presented in Horozal and Rabe (2011) and Iancu and Rabe (2011). In Horozal and Rabe (2011), we gave a comprehensive representation of first-order logic, which includes a formalisation of ZFC set theory to formalise the model theory of FOL and a formalised soundness proof of FOL in the sense of Theorem 4.25.

The biggest investment of time when representing logics in $\mathbb{M}$ is the formalisation of the foundation of mathematics. Therefore, we formalised three important foundations in Iancu and Rabe (2011): Zermelo Fraenkel set theory (Zermelo 1908; Fraenkel 1922); the logical framework Isabelle and the higher-order logic Isabelle/HOL (Paulson 1994; Nipkow *et al.* 2002); and the Mizar system for Tarski–Grothendieck set theory (Trybulec and Blair 1985). These include translations between the foundations, which even permits the translation of models across foundations.

6.2. *Applications*

Our framework was designed in response to a number of specific problems we encountered in practice. In all cases, the lack of a comprehensive framework like ours had previously precluded general solutions.

6.2.1. *An atlas of logics.*

An important application of logical frameworks is to structure and relate the multitude of logics in use. This goal was a central motivation of institutions and has also been pursued in LF (Pfenning *et al.* 2003). And in both model and proof-theoretical frameworks, there are substantial collections of presentations of model and proof-theoretical logics, respectively. But a collection comprising both model and proof theory has so far been lacking.

Our framework provides the theoretical base of the LATIN project (see Codescu *et al.* (2011); see also Section 6.1), which systematically builds such a collection of logic presentations. The LATIN atlas contains formalisations of logics and logic translations presented as a diagram of $\mathbb{M}$ signatures. It is made available as a web-based portal of documented logic definitions.

6.2.2. *Logic-aware machines.*

A central motivation of logical frameworks has been the goal to generically mechanise arbitrary logics: the implementation of one logical framework induces implementations of individual logics defined in it. This requires an abstract definition of logic and a machine-understandable language in which individual logics can be presented. Our framework provides both by presenting individual uniform logics as $\mathbb{M}$-signatures, that is, LF spans. Therefore, we can now use implementations of LF (such as Twelf (Pfenning and Schürmann 1999)) to process logic presentations mechanically.

We have given an example for this work flow in Codescu *et al.* (2012). The Hets system (Mossakowski *et al.* 2007) implements the framework of institutions and acts as a mediator between parsers, static analysers and theorem provers for various object logics. Previously, Hets could only work with a fixed number of institutions and comorphisms implemented individually in the underlying programming language, but our framework connects Hets to declarative logic presentations in LF, so Hets is now able to work with arbitrary uniform logics defined in $\mathbb{M}$.

6.2.3. *Rapidly prototyping logics.*

The definition of a new logic is typically the result of a long quest, and the evaluation of candidate logics often requires large case studies, which in turn usually require sophisticated machine support. Logical frameworks can support this process by permitting the swift implementation of candidate logics.

Using our framework, users can present candidate logics concisely and quickly, and Twelf immediately induces an implementation. In particular, Twelf provides type reconstruction and module systems out of the box. Both features require a major investment to realise for individual logics, but are indispensable in practice. Moreover, themodule system can be used to build a library of logic components that serve as building blocks for large logics.

We employ this methodology in the LATIN atlas: each logic feature (such as a connective or a type formation operator) is presented separately along with its proof and model-theoretical semantics, and logics are composed using colimits. Therefore, the presentation of each logic only requires formalising those features not already present in LATIN. Moreover, LATIN includes a library of modular formalisations of a variety of type and set theories so that logics can be built easily on top of and interpreted in a wide range of languages.

6.2.4. *Verified heterogeneous reasoning.* At the moment, mechanically verified formalisations of mathematics are restricted to individual proof-theoretical logics, whose implementations find and verify theorems: examples include Isabelle/HOL (Nipkow *et al.* 2002) and Mizar (Trybulec and Blair 1985). In this context, heterogeneous reasoning, that is, the reuse of theorems along logic translations, is problematic because the translation engine would lie outside the verifying system. Applications are usually limited to translating and dynamically reverifying a suite of theorems (see, for example, Krauss and Schropp (2010) and Keller and Werner (2010)).

On the other hand, model-theoretical adequacy (in the sense of Definition 3.29) has been used very successfully in the institution community to borrow proof systems along a statically verified logic translation (Cerioli and Meseguer 1997). But while there are systems (such as Hets (Mossakowski *et al.* 2007)) that permit the presentation and application of such translations, there is no mechanised support for verifying them.

Our framework provides such support. For example, Sojakova (2010) uses it to prove the model-theoretical adequacy of the translation from modal logic to first-order logic. Proving the well formedness and adequacy of the translation reduces to type checking in LF, which is verified mechanically by Twelf.

6.2.5. *Logical knowledge management.* In Rabe and Kohlhase (2011), we have developed the MMT interface language for logic-related applications. The latter include both deduction systems, such as theorem provers, proof assistants and type checkers, and management systems, such as databases, browsers, IDEs and search engines. MMT combines a modular representation format with a scalable knowledge management infrastructure (Kohlhase *et al.* 2010) to provide an exchange format for logics, signatures and theories, as well as their morphisms, for foundations, and for expressions, proofs and models.

For example, these objects can be presented using Twelf, exported as MMT and then imported by any other application. In particular, authors can use Twelf's human-oriented concrete syntax, including the reconstruction of omitted terms and types, and the importing application can use fully reconstructed, and thus easily machine-processable in MMT format. This work flow is used in Codescu *et al.* (2012) to present logics in Twelf and use them in Hets.

6.3. *Related work*

6.3.1. *Frameworks based on model theory.* There are several closely related frameworks that define logics and logic comorphisms (possibly with different names) as certain tuples of categorical objects.

Our definitions of logics and logic comorphisms follow and subsume those of *institutions* and institution comorphisms (Goguen and Burstall 1992). We obtain a forgetful functor from logics to institutions by dropping the proof theory from a logic and the proof translation from a logic comorphism. Similarly, if we drop the condition on proof translations in Definition 3.22, we recover institution comorphism modifications. These were introduced in Diaconescu (2002), albeit with a weaker condition; a similar concept was called a representation map in Tarlecki (1996). Our Definition 3.22 extends the definition of institution comorphism modification given in Mossakowski (2005) in the expected way.

Meseguer (1989) introduced *general logics* as tuples $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models, \vdash)$, where $\vdash$ is an *entailment system* between the sentences. Entailment systems formalise the provability relation without formalising proofs. Our use of $\vdash_\Sigma F$ as a truth judgment is different from entailment systems, but our provability relation $\Theta \vdash_\Sigma F$ is always an entailment system. In addition, Meseguer (1989) defines a proof calculus as a functor from theories to a fixed but arbitrary category **Str**. Our definition of proof categories can be viewed as the special case $\mathbf{Str} = \mathcal{PFCAT}$, and in this special case, proofs can be represented as morphisms.

Mossakowski *et al.* (2005) and Diaconescu (2006) introduced *proof-theoretic institutions* as, essentially, tuples $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models, \mathbf{Pf})$. Here the relation between sentences and objects of the proof category is predetermined because the objects of the proof categories are always the sets of sentences. Our definition of proof categories is more general and uses the truth judgment $\vdash$ to relate sentences to objects in the proof category. In particular, the objects of our proof categories can be the multi-sets of sentences, which solves a problem discovered by us in an early revision of Diaconescu (2006): there, the free generation of a proof system by a set of rules is restricted to logics with signature morphisms $\sigma : \Sigma \to \Sigma'$ for which $\mathbf{Sen}(\sigma)$ is injective; otherwise, σ would not induce a canonical functor $\mathbf{Pf}(\sigma)$ between proof categories. Later, Lawvere theories were used to overcome this problem in the revised version of Mossakowski *et al.* (2005): there, no size restriction on the products is used, and proof categories are not small.

Mossakowski *et al.* (2005) also generalised to $\mathbb{C}/\mathbb{D}$-*institutions* where $\mathbb{C}$ and $\mathbb{D}$ are the codomains of **Pf** and **Mod**, respectively, and **Sen** is obtained by composing **Pf** with a fixed functor $\mathbb{C} \to \mathcal{SET}$. Our logics are almost $\mathcal{PFCAT}/\mathcal{CAT}$ institutions; the difference is again that we give **Sen** and **Pf** separately and relate them through $\vdash$.

Fiadeiro and Sernadas (1988) introduced Π-*institutions* as tuples $(\mathbf{Sig}, \mathbf{Sen}, Cn)$ where Cn is a closure operator on sets of formulas defining consequence. Cn is treated proof-theoretically, and theory morphisms are studied proof-theoretically.

The idea of using a meta-logic like $\mathbb{M}$ is well known, and was studied for institutions in, for example, Tarlecki (1996). Our generalisation to logic encodings is not surprising, and our notion of model-theoretical adequacy is known as the model expansion property. The major novelty in our work is the use of LF as a concrete meta-logic.

Martí-Oliet and Meseguer (1996) proposed *rewriting logic* as another concrete meta-logic within an informal framework similar to general logics. Rewriting logic arises by combining sorted first-order logic with term rewriting, which makes it very different from LF. In rewriting logic, the syntax of an object logic is typically represented as a sorted first-order theory, and the proof theory as a set of rewriting rules; the model theory is not represented syntactically but through the model theory of rewriting logic. The main advantage of rewriting logic over LF is the use of rewriting to simplify expressions; the main advantage of LF is the use of dependent type theory and higher-order abstract syntax to represent proofs and variable binding as expressions.

The most advanced implementations of such frameworks are the Hets implementation of institutions (Mossakowski *et al.* 2007) and the Maude implementation of a simplified variant of rewriting logic (Clavel *et al.* 1996). Hets uses Haskell as an implementation language for the syntax of institutions and comorphisms. It maintains a graph of logics through which theories can be translated between logics and which serves as middleware between implementations of individual logics. Maude uses theories of rewriting logic to represent the syntax and proof theory of object logics as theories of the meta-logic. It implements term rewriting to provide computation within the logical framework.

Hets implements an abstract framework (institutions) whereas Maude implements a concrete meta-logic within a given framework (general logics). The latter corresponds to our approach: Twelf (Pfenning and Schürmann 1999) implements $\mathbb{LF}$ and thus $\mathbb{M}$ within the framework of our logics; in particular, logics are represented declaratively. The former is complementary to our approach, for example, we have used $\mathbb{M}$ as a convenient way to add logics to Hets in Codescu *et al.* (2012). Unlike Hets and Maude, our approach also permits the mechanised representation of model theory.

6.3.2. *Frameworks based on proof theory.* Our definition of $\mathbb{M}$ subsumes the logical framework LF. The use of LF signatures L^{syn} and L^{pf} to obtain proof-theoretical logic encodings are well understood (see, for example, Pfenning (2001)). The reasoning about the adequacy of these encodings corresponds to our comorphisms (Φ, α, γ). A difference is that in the proof-theoretical community, such encodings are considered adequate if α and γ are isomorphisms; we require only a weaker condition on γ here.

Isabelle (Paulson 1994) is an alternative logical framework, which uses higher-order logic with shallow polymorphism (Church 1940) instead of dependent type theory. The main advantages over LF are recursive and inductive computation and (semi-)automated reasoning support. Other type theories such as Martin-Löf type theory in Agda (Martin-Löf 1974; Norell 2005) or the calculus of constructions in Coq (Coquand and Huet 1988; Bertot and Castéran 2004) are also sometimes used as logical frameworks, and have similar advantages. The main advantages of LF are the use of higher-order abstract syntax, which facilitates adequacy proofs, and the support for signature morphisms in Twelf (Pfenning and Schürmann 1999).

Our definition of $\mathbb{M}$ and our results for it only depend on a few properties of LF. The main assumptions are that the signatures, and for each signature the contexts form categories, and that there are pushouts along inclusions. This is the case for almost all type theories. Thus, our definitions can be generalised easily to most type theories subsuming

LF such as Martin-Löf type theory or the calculus of constructions. Even though the signature *Base* uses dependent types, it is also easy to adapt the definitions to (simply typed) higher-order logic. In that case, the signature *Base* contains $\mathtt{ded} : \mathtt{form} \rightarrow \mathtt{prop}$ where `prop` is the type of propositions. This corresponds to logic encodings in Isabelle, where `ded` is typically called `Trueprop`.

While logic translations play a major role in model-theoretical frameworks, they have been studied less systematically for proof-theoretical frameworks. The most comprehensive approach was the Logosphere project (Pfenning *et al.* 2003), which employed LF as a framework for logic translations. These were implemented in two ways:

(1) using the operational semantics of Twelf based on logic programming;
(2) using the specifically developed functional programming language Delphin (Poswolsky and Schürmann 2008).

Our use of LF signature morphisms differs because our translations are represented in a declarative language.

Other logic translations are typically represented *ad hoc*, that is, as implementations outside a logical framework: see, for example, Obua and Skalberg (2006), McLaughlin (2006), Krauss and Schropp (2010) and Keller and Werner (2010). Instead of appealing to general results about the framework, the correctness of such a translation can be guaranteed by verifying the translated proofs. The disadvantage is that proofs must be produced, stored, translated and verified for each translated theorem.

6.3.3. *Representing models.* Our representation of models as morphisms goes back to Lawvere's representation of models as functors (Lawvere 1963) and the use of initial algebras as the semantics of theories (Goguen *et al.* 1978). This approach has been applied most systematically in the area of categorical models of type theories (see, for example, Pitts (2000)) and has been integrated into model-theoretical frameworks (see, for example, Goguen and Burstall (1986), Fiadeiro and Sernadas (1988) and Goguen *et al.* (2007)). The representation of models as signature morphisms into some fixed signature has also been used to show the amalgamation property of given institutions (see, for example, Diaconescu (2008)).

From a model-theoretical perspective, the novelty of our approach is to use morphisms in a category, namely $\mathbb{LF}$, whose objects and morphisms are given in terms of concrete syntax, and do not depend on any foundation of mathematics. That makes it possible to represent models as expressions in a mechanised language. Moreover, by separating Σ^{syn} and Σ^{mod}, and using morphisms out of Σ^{mod} as models (rather than morphisms out of Σ^{syn}), we can represent logics whose models have a different structure from the syntax. For example, Kripke models of modal logic use a set of worlds and an accessibility relation, which are not present in the syntax.

In type theories, model theory has occasionally been represented in a similar way. For example, in Benton *et al.* (2009) and Coquand and Dybjer (1997), the syntax of formal languages is represented as an inductive data type and models as inductive functions out of it. Isabelle (Paulson 1994) provides locales and interpretations, which behave very similarly to LF signatures and signature morphisms.

From a type theoretical perspective, the novelty of our approach is the systematic design of a logical framework on top of this intuition. Moreover, we abstract from the foundation by permitting the use of an arbitrary $\mathbb{LF}$ signature. Thus, we avoid the commitment to a specific foundation such as higher-order logic or Martin-Löf type theory that is often inherent in type theoretical representations of models.

7. Conclusions

7.1. *A comprehensive logical framework.*

We have given a logical framework that permits comprehensive logic representations: the syntax, model theory and proof theory of logics, as well as their translations, are represented within a formal logical framework. In particular, our approach combines the model- and proof-theoretical perspectives. Despite the ontological and philosophical differences of these two perspectives on logic, we are able to preserve the flavour and advantages of each of them.

Specifically, we have extended the model-theoretical framework of institutions (Goguen and Burstall 1992) with the notions of judgments and proof categories. Our definitions preserve the elegance and abstraction of institutions while permitting a natural integration of proof-theoretical logics, which can be seen as the continuation of work undertaken in Meseguer (1989) and Mossakowski *et al.* (2005).

Correspondingly, we have extended the proof-theoretical framework LF (Harper *et al.* 1993) with the notions of homomorphisms and model categories. We use a logic based on LF as a meta-logic in which object logics are represented, which is a response to a suggestion made in Tarlecki (1996). Using this meta-logic, all logical notions are defined as syntactic objects in the LF type theory, and can thus be verified mechanically. This includes a special LF signature representing the foundation of mathematics so that we can represent model theory even though our meta-logic is foundationally uncommitted.

7.2. *Evaluation.*

Using the criteria we gave at the beginning of Section 6, we can conclude as follows:

(i) As described in Section 6.1, our framework preserves the large collection of existing logic encodings in institutions and LF. Moreover, since these encodings usually covered either the model theory or the proof theory, in many cases we can now give comprehensive encodings for the first time. For logic translations, our work errs on the side of simplicity in the simplicity/expressivity trade-off, and extensions remain for future work.

(ii) In Section 6.2, we described a number of applications that are enabled by our framework, focusing on those that are already underway. The key feature here is that our framework provides both an abstract definition of logics and a simple declarative language in which such logics can be presented concisely and easily.

(iii) Finally, our framework has been designed systematically as the simplest possible evolution of the existing frameworks that combines the above two properties. A special strength is the symmetric treatment of model theory (**Mod**, $\models$) and proof theory (**Pf** , $\vdash$). Moreover, the models-as-morphisms paradigm yields an elegant formalist representation of platonist model theory.

7.3. *Future work.*

Future work will focus on three lines of research:

(1) The theoretical framework and tool support now in place will be leveraged in the practical applications outlined in Section 6.2. In particular, we have started work on a logic atlas in the LATIN project (Codescu *et al.* 2011).
(2) Some of the limitations discussed in Section 6.1 can be overcome by using different underlying categories instead of $\mathbb{LF}$. For example, we can use a type theory with non-compositional or partial functions. Our results can be extended easily to such type theories.
(3) Our framework can be used for the theoretical analysis of a logic's meta-theory. This includes the mechanisation of soundness and completeness proofs: soundness will use Theorem 5.10; completeness is currently open, but we consider the approach along the lines of Remark 4.26 very promising. A related application is the modular development of logics as envisioned in Harper *et al.* (1994) and now implemented in the LATIN project.

Acknowledgments

The author was given the central idea of combining institutions and LF by Michael Kohlhase. The definitions in Sections 3 and 4 owe much to discussions with Răzvan Diaconescu, Joseph Goguen, Till Mossakowski, Frank Pfenning and Andrzej Tarlecki. The evaluation of the research presented here relied on collaboration with Carsten Schürmann on the implementation of the Twelf module system, and with Fulya Horozal, Mihnea Iancu and Kristina Sojakova in carrying out large-scale case studies.

References

Aiguier, M. and Diaconescu, R. (2007) Stratified institutions and elementary homomorphisms. *Information Processing Letters* **103** (1) 5–13.

Andrews, P. (1986) *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, Academic Press.

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

Avron, A., Honsell, F., Miculan, M. and Paravano, C. (1998) Encoding modal logics in logical frameworks. *Studia Logica* **60** (1) 161–208.

Awodey, S. and Rabe, F. (2011) Kripke Semantics for Martin-Löf's Extensional Type Theory. *Logical Methods in Computer Science* **7** (3).

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Schneider, P. (eds.) (2003) *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press.

Barendregt, H. (1992) Lambda calculi with types. In: Abramsky, S., Gabbay, D. and Maibaum, T. (eds.) *Handbook of Logic in Computer Science*, volume 2, Oxford University Press.

Barwise, J. (1977) An Introduction to First-Order Logic. In: Barwise, J. (ed.) *Handbook of Mathematical Logic*, North-Holland 5–46.

Benton, N., Kennedy, A. and Varming, C. (2009) Some Domain Theory and Denotational Semantics in Coq. In: Berghofer, S., Nipkow, T., Urban, C. and Wenzel, M. (eds.) Theorem Proving in Higher Order Logics. *Springer-Verlag Lecture Notes in Computer Science* **5674** 115–130.

Benzmüller, C., Paulson, L., Theiss, F. and Fietzke, A. (2008) LEO-II - A Cooperative Automatic Theorem Prover for Classical Higher-Order Logic (System Description). In: Armando, A., Baumgartner, P. and Dowek, G. (eds.) Automated Reasoning – 5th International Joint Conference, IJCAR 2010. *Springer-Verlag Lecture Notes in Computer Science* **6173** 162–170.

Bertot, Y. and Castéran, P. (2004) *Coq'Art: The Calculus of Inductive Constructions*, Springer-Verlag.

Béziau, J. (ed.) (2005) *Logica Universalis*, Birkhäuser.

Borzyszkowski, T. (2000) Higher-Order Logic and Theorem Proving for Structured Specifications. In: Choppy, C., Bert, D. and Mosses, P. (eds.) Workshop on Algebraic Development Techniques. *Springer-Verlag Lecture Notes in Computer Science* **1827** 401–418.

Bourbaki, N. (1974) *Algebra I*, Elements of Mathematics, Springer-Verlag.

Brachman, R. and Schmolze, J. (1985) An Overview of the KL-ONE Knowledge Representation Scheme. *Cognitive Science* **9** (2).

Brouwer, L. (1907) *Over de grondslagen der wiskunde*, Ph.D. thesis, Universiteit van Amsterdam. (English title: On the Foundations of Mathematics.)

Cartmell, J. (1986) Generalized algebraic theories and contextual category. *Annals of Pure and Applied Logic* **32** 209–243.

Căzănescu, V. and Roşu, G. (1997) Weak inclusion systems. *Mathematical Structures in Computer Science* **7** (2) 195–206.

Cerioli, M. and Meseguer, J. (1997) May I Borrow Your Logic? (Transporting Logical Structures along Maps). *Theoretical Computer Science* **173** 311–347.

Cervesato, I. and Pfenning, F. (2002) A Linear Logical Framework. *Information and Computation* **179** (1) 19–75.

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

Clavel, M., Eker, S., Lincoln, P. and Meseguer, J. (1996) Principles of Maude. In: Meseguer, J. (ed.) Proceedings of the First International Workshop on Rewriting Logic. *Electronic Notes in Theoretical Computer Science* **4** 65–89.

Codescu, M., Horozal, F., Kohlhase, M., Mossakowski, T. and Rabe, F. (2011) Project Abstract: Logic Atlas and Integrator (LATIN). In: Davenport, J., Farmer, W., Rabe, F. and Urban, J. (eds.) Intelligent Computer Mathematics. *Springer-Verlag Lecture Notes in Computer Science* **6824** 287–289.

Codescu, M., Horozal, F., Kohlhase, M., Mossakowski, T., Rabe, F. and Sojakova, K. (2012) Towards Logical Frameworks in the Heterogeneous Tool Set Hets. In: Mossakowski, T. and Kreowski, H. (eds.) Recent Trends in Algebraic Development Techniques 2010. *Springer-Verlag Lecture Notes in Computer Science* **7137** 139–159.

Coquand, T. and Dybjer, P. (1997) Intuitionistic model constructions and normalization proofs. *Mathematical Structures in Computer Science* **7** (1) 75–94.

Coquand, T. and Huet, G. (1988) The Calculus of Constructions. *Information and Computation* **76** (2/3) 95–120.

Curry, H. and Feys, R. (1958) *Combinatory Logic*, North-Holland.

de Bruijn, N. (1970) The Mathematical Language AUTOMATH. In: Laudet, M. (ed.) Proceedings of the Symposium on Automated Demonstration. *Springer-Verlag Lecture Notes in Mathematics* **25** 29–61.

de Bruijn, N. (1991) A Plea for Weaker Frameworks. In: Huet, G. and Plotkin, G. (eds.) *Logical Frameworks*, Cambridge University Press 40–67.

Diaconescu, R. (2002) Grothendieck institutions. *Applied Categorical Structures* **10** (4) 383–402.

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

Diaconescu, R. (2008) *Institution-independent Model Theory*, Birkhäuser.

Dumbrava, S. and Rabe, F. (2010) Structuring Theories with Partial Morphisms. In: Workshop on Algebraic Development Techniques.

Farmer, W. (2010) Chiron: A set theory with types, undefinedness, quotation, and evaluation. SQRL Report 38, McMaster University.

Farmer, W., Guttman, J. and Thayer, F. (1992) Little Theories. In: Kapur, D. (ed.) International Conference on Automated Deduction (CADE). *Springer-Verlag Lecture Notes in Computer Science* **607** 567–581.

Fiadeiro, J. and Sernadas, A. (1988) Structuring theories on consequence. In: Recent Trends in Data Type Specification. *Springer-Verlag Lecture Notes in Computer Science* **332** 44–72.

Fraenkel, A. (1922) Zu den Grundlagen der Cantor–Zermeloschen Mengenlehre. *Mathematische Annalen* **86** 230–237. (English title: On the Foundation of Cantor–Zermelo Set Theory.)

Gentzen, G. (1934) Untersuchungen über das logische Schließen I and II. *Mathematische Zeitschrift* **39** 176–210 and 405–431. (English title: Investigations into Logical Deduction.)

Girard, J. (1987) Linear Logic. *Theoretical Computer Science* **50** 1–102.

Gödel, K. (1930) Die Vollständigkeit der Axiome des Logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik* **37** 349–360. (English title: The Completeness of the Axioms of the Logical Calculus of Functions.)

Goguen, J. and Burstall, R. (1986) A study in the foundations of programming methodology: specifications, institutions, charters and parchments. In: Pitt, D., Abramsky, S., Poigné, A. and Rydeheard, D., (eds.) Workshop on Category Theory and Computer Programming. *Springer-Verlag Lecture Notes in Computer Science* **240** 313–333.

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

Goguen, J., Mossakowski, T., de Paiva, V., Rabe, F. and Schröder, L. (2007) An Institutional View on Categorical Logic. *International Journal of Software and Informatics* **1** (1) 129–152.

Goguen, J. and Rosu, G. (2002) Institution morphisms. *Formal Aspects of Computing* **13** 274–307.

Goguen, J., Thatcher, J. and Wagner, E. (1978) An initial algebra approach to the specification, correctness and implementation of abstract data types. In: Yeh, R. (ed.) *Current Trends in Programming Methodology*, volume 4, Prentice Hall 80–149.

Gordon, M. (1988) HOL: A Proof Generating System for Higher-Order Logic. In: Birtwistle, G. and Subrahmanyam, P. (eds.) *VLSI Specification, Verification and Synthesis*, Kluwer-Academic Publishers 73–128.

Haftmann, F. and Wenzel, M. (2006) Constructive Type Classes in Isabelle. In: Altenkirch, T. and McBride, C. (eds.) Types for Proofs and Programs: Selected papers, International Workshop, TYPES 2006. *Springer-Verlag Lecture Notes in Computer Science* **4502** 160–174.

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

Harper, R., Sannella, D. and Tarlecki, A. (1994) Structured presentations and logic representations. *Annals of Pure and Applied Logic* **67** 113–160.

Harrison, J. (1996) HOL Light: A Tutorial Introduction. In: Srivas, M. K. and Camilleri, A. J. (eds.) Proceedings of the First International Conference on Formal Methods in Computer-Aided Design. *Springer-Verlag Lecture Notes in Computer Science* **1166** 265–269.

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

Hilbert, D. (1926) Über das Unendliche. *Mathematische Annalen* **95** 161–90.

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

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

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

Keller, C. and Werner, B. (2010) Importing HOL Light into Coq. In: Kaufmann, M. and Paulson, L. (eds.) Interactive Theorem Proving. *Springer-Verlag Lecture Notes in Computer Science* **6172** 307–322.

Kohlhase, M., Rabe, F. and Zholudev, V. (2010) Towards MKM in the Large: Modular Representation and Scalable Software Architecture. In: Autexier, S., Calmet, J., Delahaye, D., Ion, P., Rideau, L., Rioboo, R. and Sexton, A. (eds.) Intelligent Computer Mathematics. *Springer-Verlag Lecture Notes in Computer Science* **6167** 370–384.

Krauss, A. and Schropp, A. (2010) A Mechanized Translation from Higher-Order Logic to Set Theory. In: Kaufmann, M. and Paulson, L. (eds.) Interactive Theorem Proving. *Springer-Verlag Lecture Notes in Computer Science* **6172** 323–338.

Lambek, J. and Scott, P. (1986) *Introduction to Higher-Order Categorical Logic*, Cambridge Studies in Advanced Mathematics **7**, Cambridge University Press.

Lawvere, F. (1963) *Functional Semantics of Algebraic Theories*, Ph.D. thesis, Columbia University.

Lawvere, W. (1969) Adjointness in Foundations. *Dialectica* **23** (3-4) 281–296.

Mac Lane, S. (1998) *Categories for the working mathematician*, Springer-Verlag.

Martí-Oliet, N. and Meseguer, J. (1996) Rewriting Logic as a Logical and Semantic Framework. In: Rewriting Logic and its Applications. *Electronic Notes in Theoretical Computer Science* **4** 352–358.

Martin-Löf, P. (1974) An Intuitionistic Theory of Types: Predicative Part. In: *Proceedings of the '73 Logic Colloquium*, North-Holland 73–118.

Martin-Löf, P. (1996) On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic* **1** (1) 3–10.

McLaughlin, S. (2006) An Interpretation of Isabelle/HOL in HOL Light. In: Shankar, N. and Furbach, U. (eds.) Proceedings of the 3rd International Joint Conference on Automated Reasoning. *Springer-Verlag Lecture Notes in Computer Science* **4130** 192–204.

Meseguer, J. (1989) General logics. In: Ebbinghaus, H.-D. *et al.* (eds.) *Proceedings, Logic Colloquium 1987*, North-Holland 275–329.

Mossakowski, T. (2005) *Heterogeneous Specification and the Heterogeneous Tool Set*, Habilitation thesis. (Available at `http://www.informatik.uni-bremen.de/ till/`.)

Mossakowski, T., Goguen, J., Diaconescu, R. and Tarlecki, A. (2005) What is a logic? In: Béziau, J. (ed.) *Logica Universalis*, Birkhäuser 113–133.

Mossakowski, T., Maeder, C. and Lüttich, K. (2007) The Heterogeneous Tool Set. In: O. Grumberg and M. Huth (ed.) TACAS 2007. *Springer-Verlag Lecture Notes in Computer Science* **4424** 519–522.

Mossakowski, T., Tarlecki, A. and Pawlowski, W. (1997) Combining and Representing Logical Systems. In: Moggi, E. and Rosolini, G. (eds.) *Category Theory and Computer Science* **1290** 177–196.

Mosses, P. D. (ed.) (2004) CASL Reference Manual. *Springer-Verlag Lecture Notes in Computer Science* **2960**.

Naumov, P., Stehr, M. and Meseguer, J. (2001) The HOL/NuPRL proof translator – a practical approach to formal interoperability. In: Boulton, R. and Jackson, P. (eds.) 14th International Conference on Theorem Proving in Higher Order Logics. *Springer-Verlag Lecture Notes in Computer Science* **2152** 329–345.

Nipkow, T., Paulson, L. and Wenzel, M. (2002) Isabelle/HOL – A Proof Assistant for Higher-Order Logic. *Springer-Verlag Lecture Notes in Computer Science* **2283**.

Norell, U. (2005) The Agda WiKi. (Available at `http://wiki.portal.chalmers.se/agda`.)

Obua, S. and Skalberg, S. (2006) Importing HOL into Isabelle/HOL. In: Shankar, N. and Furbach, U. (eds.) Proceedings of the 3rd International Joint Conference on Automated Reasoning. *Springer-Verlag Lecture Notes in Computer Science* **4130** 298–302.

Owre, S., Rushby, J. and Shankar, N. (1992) PVS: A Prototype Verification System. In: Kapur, D. (ed.) 11th International Conference on Automated Deduction (CADE). *Springer-Verlag Lecture Notes in Computer Science* **607** 748–752.

Paulson, L. (1994) Isabelle: A Generic Theorem Prover. *Springer-Verlag Lecture Notes in Computer Science* **828**.

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

Pfenning, F. (2001) Logical frameworks. In: *Handbook of automated reasoning*, Elsevier 1063–1147.

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

Pfenning, F., Schürmann, C., Kohlhase, M., Shankar, N. and Owre, S. (2003) The Logosphere Project. (Available at `http://www.logosphere.org/`.)

Pientka, B. and Dunfield, J. (2010) A Framework for Programming and Reasoning with Deductive Systems (System description). In: Armando, A., Baumgartner, P. and Dowek, G. (eds.) Automated Reasoning – 5th International Joint Conference, IJCAR 2010. *Springer-Verlag Lecture Notes in Computer Science* **6173** 15–21.

Pitts, A. (2000) Categorical Logic. In: Abramsky, S., Gabbay, D. and Maibaum, T. (eds.) *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, Oxford University Press 39–128.

Poswolsky, A. and Schürmann, C. (2008) System Description: Delphin – A Functional Programming Language for Deductive Systems. In: Abel, A. and Urban, C. (eds.) International Workshop on Logical Frameworks and Metalanguages: Theory and Practice. *Electronic Notes in Theoretical Computer Science* **228** 113–120.

Rabe, F. (2008) *Representing Logics and Logic Translations*, Ph.D. thesis, Jacobs University Bremen. (Available at `http://kwarc.info/frabe/Research/phdthesis.pdf`.)

Rabe, F. and Kohlhase, M. (2011) A Scalable Module System. (Available at `http://arxiv.org/abs/1105.0548`.)

Rabe, F. and Schürmann, C. (2009) A Practical Module System for LF. In: Cheney, J. and Felty, A. (eds.) *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, ACM International Conference Proceedings Series, LFMTP'09 40–48.

Robinson, A. (1950) On the application of symbolic logic to algebra. In: *Proceedings of the International Congress of Mathematicians*, American Mathematical Society 686–694.

Seely, R. (1984) Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society* **95** 33–48.

Smullyan, R. (1995) *First-Order Logic* (second corrected edition), Dover.

Sojakova, K. (2010) Mechanically Verifying Logic Translations, Master's thesis, Jacobs University Bremen.

Tarlecki, A. (1996) Moving between logical systems. In: Haveraaen, M., Owe, O. and Dahl, O.-J. (eds.) Recent Trends in Data Type Specifications – 11th Workshop on Specification of Abstract Data Types. *Springer-Verlag Lecture Notes in Computer Science* **1130** 478–502.

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

Tarski, A. and Vaught, R. (1956) Arithmetical extensions of relational systems. *Compositio Mathematica* **13** 81–102.

Trybulec, A. and Blair, H. (1985) Computer Assisted Reasoning with MIZAR. In: Joshi, A. (ed.) *Proceedings of the 9th International Joint Conference on Artificial Intelligence* 26–28.

Whitehead, A. and Russell, B. (1913) *Principia Mathematica*, Cambridge University Press.

Zermelo, E. (1908) Untersuchungen über die Grundlagen der Mengenlehre I. *Mathematische Annalen* **65** 261–281. (English title: Investigations in the foundations of set theory I.)