Representing Model Theory in a Type-Theoretical Logical Framework[☆]

Fulya Horozal[a], Florian Rabe[a]

[a] *Jacobs University Bremen, Germany*

Abstract

In a broad sense, logic is the field of formal languages for knowledge and truth that have a formal semantics. It tends to be difficult to give a narrower definition because very different kinds of logics exist. One of the most fundamental contrasts is between the different methods of assigning semantics. Here two classes can be distinguished: model theoretical semantics based on a foundation of mathematics such as set theory, and proof theoretical semantics based on an inference system possibly formulated within a type theory.

Logical frameworks have been developed to cope with the variety of available logics unifying the underlying ontological notions and providing a meta-theory to reason abstractly about logics. While these have been very successful, they have so far focused on either model or proof theoretical semantics. We contribute to a unified framework by showing how a type/proof theoretical Edinburgh Logical Framework LF can be applied to the representation of model theoretical logics.

We give a comprehensive formal representation of first-order logic covering both its proof and its model theoretical semantics and its soundness in LF. For the model theory, we have to represent the mathematical foundation itself in LF, and we provide two solutions for that. Firstly, we give a meta-language that is strong enough to represent the model theory while being simple enough to be treated as a fragment of untyped set theory. Secondly, we represent Zermelo-Fraenkel set theory and show how it subsumes our meta-language. All representations are given in and mechanically verified by the Twelf implementation of LF. Moreover, we use the Twelf module system to treat all connectives and quantifiers independently. Thus, individual connectives are available for reuse when representing other logics, and we obtain the first version of a feature library from which logics can be pieced together.

Our results and methods are not restricted to first-order logic and scale to a wide variety of logical systems thus demonstrating the feasibility of comprehensively formalizing large scale representation theorems in a logical framework.

[☆]This work was partially supported by the Deutsche Forschungsgemeinschaft within grant KO 2428/9-1.

Contents

1. Introduction

Logic has been an important research topic in mathematics and computer science since the foundational crisis of mathematics. Research on logics has included the difficult and sometimes contentious question how to choose the ontological foundations of logic. Logical frameworks have proved an important research result to answer this question – they are abstract formalisms that permit the formal definition of specific logics.

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

While some model-theoretical frameworks attempt to integrate proof theory (e.g., [Mes89, MGDT05, Dia06]), the opposite integration is less developed. This is unfortunate because many of the results and techniques developed for proof theoretical logics could also benefit model-theoretical reasoning.

We are particularly interested in logic encodings in the Edinburgh Logical Framework (LF), which is related to Martin-Löf type theory and can be seen as the dependently-typed corner of the λ-cube ([Bar92]). LF represents syntax and proof theoretical semantics of a logic using higher order abstract syntax and the judgments-as-types paradigm ([ML96]). This has proved very successful for proof-theoretical logic representations ([HST94, AHMP98, Pfe00, NSM01]).

In [Rab08], we introduced a framework that attempts to preserve and exploit the respective advantages of model and proof theoretical representation. The central idea is to also represent the model theory of a logic in a type-theoretical logical framework by specifying models in a suitable meta-language.

In this paper we show how to implement such logic representations in LF. We pick LF because we have recently equipped the Twelf implementation of LF with a strong module system [PS99, RS09]. This module system is rigorously based on theory morphisms, which have proved very successful to reason about model-theoretical logic representations (e.g., [GB92, AHMS99, SW83]). Therefore, it is particularly appropriate for an encoding that combines proof- and model-theoretical aspects.

Our central results are (i) the full representation of first-order logic (FOL) comprising syntax, proof theory, and model theory, and (ii) a formal proof of the soundness of FOL based on this representation. In particular, the soundness proof is verified mechanically by the LF implementation Twelf. While this is interesting in itself, the main value of our work is not the encoding but the methodology we employ. We use FOL as an example logic mainly because it is most widely known and thus interferes least with the rather abstract subject matter. Other logics can be represented analogously.

Furthermore, we use the LF module system for a modular development of syntax, proof theory, model theory, and soundness, i.e., all connectives and quantifiers are treated separately in all four parts of the encoding. These modules can be reused flexibly to encode other logics. For example, we obtain encodings for any logic that arises by omitting some connectives or quantifiers from FOL. Less trivially, the encoding of each connective or quantifier can be reused for any logic using them. For example, this enabled one of our students to extend the work presented here to sorted FOL within two days.

Our approach is especially interesting when studying rarer or new logics, for which no smoothed-out semi-formal definitions are available yet. In particular, our framework can be used for the rapid prototyping of logics. Since it covers both proof and model theory, it permits an approach that we call *syntax-semantics-codesign*, to coin a phrase: Researchers can give a fully formal and mechanically verified definition of a formal language and its semantics at a level of convenience and elegance that competes with working it out on paper.

In Sect. 2, we describe some preliminaries and introduce some notation: FOL in Sect. 2.1, and LF in Sect. 2.2. In Sect. 3, we sketch the framework we will

use. The main sections of this paper are Sect. 4 and 5. In the former, we give the encoding of FOL in LF where we use a variant of higher-order logic as a simple and convenient meta-language to represent the models. In the latter, we extend the encoding to cover set theory itself as a foundation of mathematics, in which models are expressed. Thus, we can give a comprehensive representation of FOL and its set-theoretical model theory in LF. Both in Sect. 4 and 5, we describe the encoding of FOL in a way that makes the general methodology apparent and provides a template for the encoding of other logics.

A preliminary version of this paper has appeared as [HR09]. The present version has been fully revised and substantially extended. Most importantly, the encoding of set theory, which was only sketched in [HR09], has been worked out. Among the changes we made, two are especially notable. Firstly, we changed the meta-language employed to represent models from Martin-Löf type theory to higher-order logic. This was motivated by the desire to separate types and propositions rather than identify them. Secondly, in [HR09], we identified some features missing in the implementation of the LF module system. These have been added by now, which enabled us to completely refactor the LF encodings.

Our approach is very extensible, and we have treated or are currently working on corresponding representations of sorted, higher-order, and description logics. These are part of a logic atlas that is developed as a collaborative research effort within the LATIN project ([KMR09]). All Twelf sources are available from the project website.

2. Preliminaries

2.1. First-Order Logic

In this section, we will introduce first-order logic in order to give an overview of the definitions and notations we will use. The definitions here also serve as the reference definitions when proving the adequacy of our encodings.

Definition 1 (Signatures)**.** A FOL-*signature* is a triple (Σ_f, Σ_p, ar) where Σ_f and Σ_p are disjoint sets of function and predicate symbols, respectively, and $ar : \Sigma_f \cup \Sigma_p \to \mathbb{N}$ assigns arities to symbols. We will treat constants and boolean variables as the special case of arity 0.

Definition 2 (Expressions)**.** A FOL-*context* is a list of variables. For a signature Σ and a context Γ, the *terms* over Σ and Γ are formed from the variables in Γ and the application of function symbols $f \in \Sigma_f$ to terms according to $ar(f)$. The *formulas* over Σ and Γ are formed from the application of predicate symbols $p \in \Sigma_p$ to a number of terms according to $ar(p)$ as well as $\doteq$, *true*, *false*, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\forall$, and $\exists$ in the usual way. Formulas in the empty context are called Σ-*sentences*, and we write $Sen(\Sigma)$ for the set of sentences.

Definition 3 (Theories)**.** A FOL-*theory* is a pair (Σ, Θ) for a signature Σ and a set $\Theta \subseteq Sen(\Sigma)$ of *axioms*.

Definition 4 (Signature Morphisms)**.** Given two signatures $\Sigma = (\Sigma_f, \Sigma_p, ar)$ and $\Sigma' = (\Sigma'_f, \Sigma'_p, ar')$, a FOL-*signature morphism* $\sigma : \Sigma \to \Sigma'$ is an arity-preserving mapping from Σ_f to Σ'_f and from Σ_p to Σ'_p.

The *homomorphic extension* of σ – which we also denote by σ – is the mapping from terms and formulas over Σ to terms and formulas over Σ' that replaces every symbol $s \in \Sigma_f \cup \Sigma_p$ with $\sigma(s)$. The *sentence translation* $Sen(\sigma) : Sen(\Sigma) \to Sen(\Sigma')$ arises as the special case of applying σ to sentences.

Example 5 (Monoids and Groups). We will use the theories $Monoid = (MonSig, MonAx)$ and $Group = (GrpSig, GrpAx)$ of monoids and groups as running examples. $MonSig_f$ is the set $\{\circ, e\}$ where $\circ$ is binary (written infix) and e is nullary, and $MonSig_p$ is empty. $MonAx$ consists of the axioms for

- associativity: $\forall x\ \forall y\ \forall z\ x \circ (y \circ z) \doteq (x \circ y) \circ z$,
- left-neutrality: $\forall x\ e \circ x \doteq x$,
- right-neutrality: $\forall x\ x \circ e \doteq x$.

The theory *Group* extends *Monoid*, i.e., *GrpSig* adds a unary function symbol *inv* (written as superscript $^{-1}$) to *MonSig*, and *GrpAx* adds axioms for

- left-inverseness: $\forall x\ x^{-1} \circ x \doteq e$,
- right-inverseness: $\forall x\ x \circ x^{-1} \doteq e$

to *MonAx*. The inclusion mapping *MonGrp* is a signature morphism from *MonSig* to *GrpSig*. It is also a theory morphism from *Monoid* to *Group*.

There are various ways to define the *proof theory* of FOL. In this paper we choose the natural deduction calculus (ND) with introduction and elimination rules. We will use the phrase *proof theoretical semantics* when speaking about the induced provability relation; we will not consider proof normalization, which some authors mean when using that phrase.

Definition 6 (Proof Theoretical Theorems)**.** Given a theory (Σ, Θ), we say that $F \in Sen(\Sigma)$ is a theorem of (Σ, Θ) if the judgment $F_1, \ldots, F_n \vdash_\Sigma F$ is derivable for some $\{F_1, \ldots, F_n\} \subseteq \Theta$ using the calculus shown in Fig. 1. We write this as $\Theta \vdash_\Sigma F$.

Definition 7 (Proof Theoretical Theory Morphisms)**.** A signature morphism from Σ to Σ' is a *proof theoretical theory morphism* from (Σ, Θ) to (Σ', Θ'), written $\sigma : (\Sigma, \Theta) \xrightarrow{\mathsf{P}} (\Sigma', \Theta')$, if $Sen(\sigma)$ maps the axioms of (Σ, Θ) to theorems of (Σ', Θ'), i.e., for all $F \in \Theta$, $\Theta' \vdash_{\Sigma'} Sen(\sigma)(F)$ holds.

Lemma 8 (Proof Translation)**.** *Assume a theory morphism* $\sigma : (\Sigma, \Theta) \to (\Sigma', \Theta')$*. If* F *is a theorem of* (Σ, Θ)*, then* $Sen(\sigma)(F)$ *is a theorem of* (Σ', Θ')*. In other words, provability is preserved along theory morphisms.*

We develop the *model theory* of FOL as an institution ([GB92]).

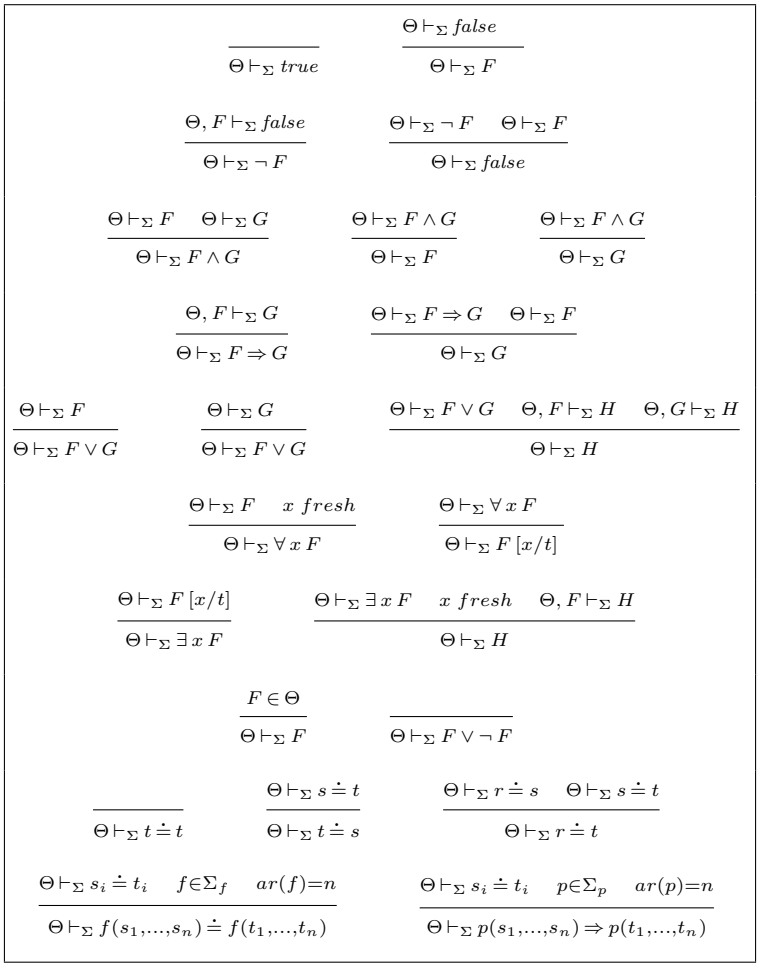


Figure 1: Proof Rules

Definition 9 (Models of a FOL-Signature)**.** A FOL-*model of a signature* Σ is a pair (U, I) where U is a non-empty set (called the *universe*) and I is an interpretation function of Σ-symbols such that

- $f^I \in U^{U^n}$ for $f \in \Sigma_f$ with $ar(f) = n$,
- $p^I \subseteq U^n$ for $p \in \Sigma_p$ with $ar(p) = n$.

We write $Mod(\Sigma)$ for the class of Σ-models.

Definition 10 (Model Theoretical Semantics)**.** Assume a signature Σ, a context Γ, and a Σ-model $M = (U, I)$. An *assignment* is a mapping from Γ to U. For an assignment α, the *interpretations* $[\![t]\!]^{M,\alpha} \in U$ of terms t and $[\![F]\!]^{M,\alpha} \in \{0, 1\}$ of formulas F over Σ and Γ are defined in the usual way by induction on the syntax. Given a sentence F, we write $M \models_\Sigma F$ if $[\![F]\!]^M = 1$.

Given a theory (Σ, Θ), we write the class of (Σ, Θ)-models as

$$Mod(\Sigma, \Theta) = \{M \in Mod(\Sigma) \mid M \models_\Sigma F \text{ for all } F \in \Theta\}$$

Definition 11 (Model Theoretical Theorems)**.** Given a theory (Σ, Θ), we say that $F \in Sen(\Sigma)$ is a *model theoretical theorem* of (Σ, Θ) if the following holds for all Σ-models M: If $M \models_\Sigma A$ for all $A \in \Theta$, then also $M \models_\Sigma F$.

Definition 12 (Model Reduction)**.** Given a signature morphism $\sigma : \Sigma \to \Sigma'$ and a Σ'-model $M' = (U, I')$, we obtain a Σ-model (U, I), called the *model reduct* of M' along σ, by putting $s^I = \sigma(s)^{I'}$ for all symbols of Σ. We write $Mod(\sigma) : Mod(\Sigma') \to Mod(\Sigma)$ for the induced model reduction.

Definition 13 (Model Theoretical Theory Morphisms)**.** Given two theories (Σ, Θ) and (Σ', Θ'), a *model theoretical theory morphism* from (Σ, Θ) to (Σ', Θ'), written $\sigma : (\Sigma, \Theta) \xrightarrow{\text{M}} (\Sigma', \Theta')$, is a signature morphism from Σ to Σ' such that $Mod(\sigma)$ reduces models of (Σ', Θ') to models of (Σ, Θ), i.e, for all $M' \in Mod(\Sigma', \Theta')$, we have $Mod(\sigma)(M') \in Mod(\Sigma, \Theta)$.

Lemma 14 (Satisfaction Condition)**.** *Assume a FOL-signature morphism $\sigma : \Sigma \to \Sigma'$, a Σ-sentence F, and a Σ'-model M'. Then $M' \models_{\Sigma'} Sen(\sigma)(F)$ iff $Mod(\sigma)(M') \models_\Sigma F$.*

Example 15 (Continued). The integers form a model $Int = (\mathbb{Z}, +, 0, -)$ for the theory of groups (where we use a tuple notation to give the universe and the interpretations of $\circ$, e, and inv, respectively). The model reduction $Mod(MonGrp)(Int) = (\mathbb{Z}, +, 0)$ along $MonGrp$ yields the integers seen as a model of the theory of monoids.

We have given both proof theoretical and model theoretical definitions of *theorem* and *theory morphism*. In general, these must be distinguished to avoid a bias towards proof or model theory. However, they coindice if a logic is sound and complete:

Theorem 16 (Soundness and Completeness)**.** *Assume a FOL-theory (Σ, Θ) and a Σ-sentence F. Then $\Theta \vdash_\Sigma F$ iff $\Theta \models_\Sigma F$. Therefore, for a FOL-signature morphism $\sigma : \Sigma \to \Sigma'$, we have $\sigma : (\Sigma, \Theta) \xrightarrow{\text{P}} (\Sigma', \Theta')$ iff $\sigma : (\Sigma, \Theta) \xrightarrow{\text{M}} (\Sigma', \Theta')$.*

2.2. LF and Twelf

LF ([HHP93]) is a dependent type theory that extends simple type theory with dependent function types. We will work with the Twelf implementation of LF ([PS99]). The main use of LF and Twelf is as a *logical framework* in which deductive systems are represented.

We will develop the syntax and semantics of LF along an example representation of simple type theory (STT). Typically, *kinded type families* are declared to represent the syntactic classes of the system. For STT, we declare

tp : `type`
tm : $tp \to$ `type`

Here `type` is the LF-kind of types, and tp is an LF-type whose LF-terms represent the STT-types. And $tp \to \texttt{type}$ is the kind of type families that are indexed by terms of LF-type tp; then $tm\ A$ is the LF-type whose terms represent the STT-terms of type A.

Typed constants are declared to represent the expressions of the represented system. For STT, we add

$$\begin{array}{lcll} \Longrightarrow & : & tp \to tp \to tp & \%\texttt{infix right } 0\ \Longrightarrow \\ @ & : & tm\ (A \Longrightarrow B) \to tm\ A \to tm\ B & \%\texttt{infix left } 1000\ @ \\ \lambda & : & (tm\ A \to tm\ B) \to tm\ (A \Longrightarrow B) & \end{array}$$

Here $\Longrightarrow$ is a low-binding right-associative infix symbol that takes two tp-arguments and returns a tp. It represents STT-function type formation. In the following, we will always omit the fixity and associativity declarations if they are clear from the context. In particular, besides $\Longrightarrow$ and @, binary symbols such as connectives and equality are always assumed to be declared as infix.

@ is a strong-binding left-associative infix symbol that takes two arguments – an STT-term of type $A \Longrightarrow B$ and an STT-term of type A – and returns an STT-term of type B. It represents STT-function elimination, i.e., application. In the declaration of @, A and B are free variables. These variables are implicitly Π-bound at the outside. The full type of @ is $\Pi_{A:tp}\Pi_{B:tp} tm\ (A \Longrightarrow B) \to tm\ A \to tm\ B$, i.e., @ really takes 4 arguments. This uses the main feature of dependent type theory: The first two arguments A and B may occur in the types of the later arguments and in the return type. Twelf treats A and B as *implicit arguments* and infers their values from the other arguments. Thus, we can write $f\ @\ a$ instead of $@\ A\ B\ f\ a$.

Finally, λ represents STT-function introduction, i.e., abstraction. λ is declared using *higher-order abstract syntax*. LF-functions of type $S \to T$ are in bijection to the terms of type T with a free variable of type S; thus, higher-order arguments can be used to represent binders. The above λ takes a term of STT-type B with a free variable of type A represented as an LF term of type $tm\ A \to tm\ B$. It returns an STT term of type $A \Longrightarrow B$.

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 $[x : A]\ t\ x$. (Therefore, λ is available for user-declared symbols.) In particular, in the above example, the STT-term $\lambda_{x:A} t$ is represented as the LF-term $\lambda[x : A]\ t$. Finally, we write $A \to B$ instead of $\{x : A\}\ B$ if x does not occur in B, and we will also omit the types of bound variables if they can be inferred.

LF employs the Curry-Howard correspondence to represent proofs-as-term ([CF58, How80]) and extends it to the *judgments-as-types* methodology ([ML96]). For example, we can turn the above STT into a logic by adding a type *prop* of propositions and a truth judgment $True$ on it:

$$
\begin{array}{lcl}
\mathit{prop} & : & \texttt{type} \\
\mathit{True} & : & \mathit{prop} \to \texttt{type} \\
\Rightarrow & : & \mathit{prop} \to \mathit{prop} \to \mathit{prop} \\
\Rightarrow E & : & \mathit{True}\ (F \Rightarrow G) \ \to \ \mathit{True}\ F \ \to \ \mathit{True}\ G
\end{array}
$$

Here the type $\mathit{True}\ F$ represents the judgment that F is true. A judgment J is proved if there is a term of type J. Consequently, all axioms and inference rules such as the implication elimination rule $\Rightarrow E$ are represented as constants, and proofs of F are represented as terms of type $\mathit{True}\ F$.

Finally, an LF *signature* is a list of kinded type family declarations $a : K$ and typed constant declarations $c : A$. Both may carry definitions, i.e., $c : A = t$ introduces c as an abbreviation for t. This yields the following grammar for the fragment of LF we will use:

$$
\begin{array}{llll}
\text{Signatures} & \Sigma & ::= & \cdot \mid \Sigma,\ c : A \mid \Sigma,\ c : A = t \mid \Sigma,\ a : K \mid \Sigma,\ a : K = A \\
\text{Morphisms} & \sigma & ::= & \cdot \mid \sigma,\ c := t \mid \sigma,\ a := A \\
\text{Kinds:} & K & ::= & \texttt{type} \mid A \to K \\
\text{Type families:} & A, B & ::= & a \mid [x : A]\, B \mid B\ t \mid \{x : A\}\, B \\
\text{Terms:} & s, t & ::= & c \mid x \mid [x : A]\, t \mid s\ t
\end{array}
$$

Here we have already included LF *signature morphisms*. Given two signatures Σ and Σ', a signature morphism $\sigma : \Sigma \to \Sigma'$ is a typing-preserving map of Σ-symbols to 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 Σ-expressions, and we will write σ instead of $\overline{\sigma}$ from now on.

Signature morphisms preserve typing, i.e., if $\vdash_\Sigma E : F$, then $\vdash_{\Sigma'} \sigma(E) : \sigma(F)$. 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.

We will write $\sigma\ \sigma'$ for the diagram-order composition of signature morphisms, i.e., $(\sigma\ \sigma')(E) = \sigma'(\sigma(E))$.

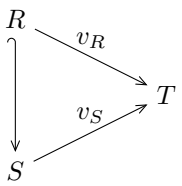
The module system for LF and Twelf ([RS09]) is based on the notion of signature morphisms ([HST94]). The toplevel declarations of modular LF declare named signatures and named signature morphisms, called *views*, e.g.,

$$\%\texttt{sig}\ S\ =\ \{\Sigma\}.\ \%\texttt{sig}\ T\ =\ \{\Sigma'\}.\ \%\texttt{view}\ v\ : S\ \to T\ = \{\sigma\}$$

Since signature morphisms must map axioms to proofs, a view has the flavor of a theorem establishing a translation from S to T or a representation of S in T or a refinement of S into T.

Besides views, the module system provides *inclusion* morphisms that hold by definition: $\%\texttt{include}\ S$ declared in T copies all declarations of S into T (thus changing T). This represents an inheritance or import relationship from S to T. The inclusion relation is transitive, and multiple inclusions of the same signature are identified. Twelf uses qualified names to access included symbols, but we will simply assume that included symbols c of S are accessible as c within T.

Views can be given modularly, too. If S includes R, then a view v_S from S to T must map all constants of S, i.e., also those of R. Often a view v_R from R to T is already present. In that case v_S can include v_R via $\%\texttt{include}\ v_R$ and only give maps for the symbols of S. If v_S is defined like that, the triangle on the right always commutes.



Thus, we arrive at the following grammar for the fragment of modular LF we will use. Here we use E for a kind, type family, or term as defined above:

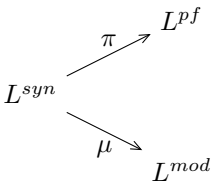
Start	G	::=	$\cdot \mid G,\ D_T \mid G,\ D_v$
Signatures	D_T	::=	$\%\texttt{sig}\ T\ =\ \{\Sigma\}$
Views	D_v	::=	$\%\texttt{view}\ v\ : S\ \to T\ =\{\sigma\}$
Sign. body	Σ	::=	$\cdot \mid \Sigma,\ c : E \mid \Sigma,\ c : E = E \mid \Sigma,\ \%\texttt{include}\ S$
View body	σ	::=	$\cdot \mid \sigma,\ c := E \mid \sigma,\ \%\texttt{include}\ v$

3. A Logical Framework Combining Proof and Model Theory

LF was designed as a language for the representation of formal systems. Similarly, the LF module system was designed as a language for the representation of translations between formal systems. This makes it a very appropriate framework for the comprehensive representation of a logic where translations – between different signatures of a logic as well as between syntax and semantics – are prevalent.

In the following, we will give an overview of the logical framework we gave in [Rab08, Rab10]. We will not actually give the framework itself, which requires a further level of abstraction beyond the scope of this paper. Instead, we define what the encoding of an individual logic looks like.

We assume a logic L defined along the lines of Sect. 2.1. An encoding of L in our framework consists of three LF signatures L^{syn} for the syntax, L^{pf} for the proof theory, and L^{mod} for the model theory, as well as two LF signature morphisms π and μ that translate the syntax into proof and model theory.



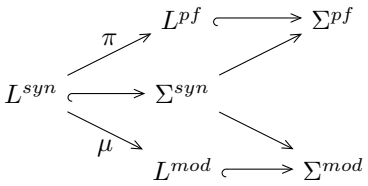
L^{syn} contains LF declarations for all symbols occurring in L-formulas. Type declarations are used for syntactic classes, e.g., sorts, terms, formulas, and judgments (typically including a truth judgment), and constant declarations are used for individual connectives, quantifiers, sorts, functions, predicates, axioms, etc. L^{pf} is typically an extension of L^{syn}, i.e., π is an inclusion morphisms. L^{pf} includes constant declarations for the axioms and inference rules of L; it may also contain type declarations for auxiliary judgments.

L^{syn} typically declares at least a type o of formulas and a type family $ded : o \to \texttt{type}$ for the truth judgment. In the simplest case, L^{pf} only adds inference rules for ded to L^{syn}.

L^{mod} contains declarations that describe models. For model-theoretically motivated logics such as first-order logic, L^{syn} and L^{mod} have a similar structure – after all for many logics L, the syntax was introduced as a way to describe the models in the first place. But for proof-theoretically motivated logics like some modal logics or intuitionistic logics, the model theory was developed a posteriori. For these, the L^{syn} and L^{mod} may vary considerable, e.g., L^{mod} might contain declarations to describe Kripke frames.

L^{mod} typically includes some meta-language that is used to reason about the truth in models, e.g., set theory. In the simplest case μ translates o to a type of truth values and *ded* to a predicate that holds for the designated truth values. Then L^{mod} axiomatizes the translation of formulas to truth values.

A specific signature Σ of L is represented as an extension of L. This corresponds to the uniform logic encodings in LF given in [HST94]. For example, signatures of propositional logic are sets of propositional variables, and the set $\Sigma = \{p_1, \ldots, p_n\}$ is encoded as the LF-signature $\Sigma^{syn} = L^{syn}$, $p_1 : o$, $\ldots$, $p_n : o$. By merging Σ^{syn} with L^{pf} and L^{mod}, we obtain Σ^{pf} and Σ^{mod}, respectively. This leads to the diagram below.



Technically, Σ^{pf} and Σ^{mod} are obtained as pushouts, a concept from category theory. We refer to [Lan98] for the basics of category theory and only remark that the category of LF-signatures has pushouts along inclusions ([HST94]). Intuitively, Σ^{mod} is obtained as follows: Take L^{mod} and add to it the translation along μ of all those declarations in Σ^{syn} that are not in L^{syn}. In other words, Σ^{mod} is the union of L^{mod} and Σ^{syn} sharing L^{syn}. Σ^{pf} is obtained accordingly.

Then Σ-sentences F are represented as β-η-normal LF-terms of type o over the signature Σ^{syn}. We will write $\ulcorner F \urcorner$ for the LF-term representing the sentence F. An encoding is adequate for the syntax if this representation is a bijection.

Similarly, Σ-proofs of F using assumptions $F_1, \ldots, F_n$ are represented as β-η-normal LF-terms over Σ^{pf} of type $\pi\,(ded\ F_1 \to \ldots \to ded\ F_n \to ded\ F)$. Again we write $\ulcorner P \urcorner$ for the encoding of the proof P and say that the encoding is adequate if it is a bijection.

We will elaborate on the representation of Σ-models and the truth in models throughout the text.

It is noteworthy how the framework takes a balanced position between proof and model theoretical perspectives on logic. In particular, the type *ded* F is used to represent truth both proof- and model-theoretically. Proof-theoretically, terms of type *ded* F represent proofs of F, model-theoretically *ded* F is a predicate on the truth value of F.

A particular feature of the framework is that soundness can be represented very naturally: A soundness proof of L is represented as a view from L^{pf} to L^{mod} that makes the resulting triangle commute. We will get back to that in Sect. 4.6.

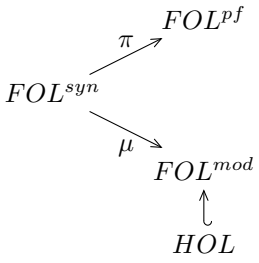
This framework is closely related to the logical frameworks of institutions ([GB92]) and LF ([HHP93]). From the perspective of institutions, it can be seen as utilizing LF-signatures to obtain concrete, strongly typed syntax to define signatures and sentences. Similarly, LF-signature morphisms are used to describe models in a way similar to parchments ([GB86]). A difference is the inclusion of proof theory and the separation into signatures Σ^{syn}, Σ^{pf}, and Σ^{mod}. Furthermore, the way LF is used to define logics and to do so modularly goes back to ideas from [HST94] and [Tar96].

From the perspective of LF, it adds signature morphisms as a means to reason about translations between signatures and logics in addition to the reasoning about logics and signatures possible with the existing logic representations. In this work, we give a concrete semantic domain, which permits to represent models in the framework as well.

4. Representing First-Order Logic

As described in Sect. 3, the encoding of FOL in LF consists of signatures FOL^{syn} for the syntax, FOL^{pf} for the proof theory, and FOL^{mod} for the model theory, together with two views $\pi : FOL^{syn} \to FOL^{pf}$ and $\mu : FOL^{syn} \to FOL^{mod}$. FOL signatures and theories will be encoded as extensions of FOL^{syn}.

We will describe FOL^{syn} in Sect. 4.1, FOL^{pf} and π in Sect. 4.2, and FOL^{mod} and μ in Sect. 4.4. FOL^{mod} will include a meta-language in which the models are specified. In textbook style descriptions, this meta-language is usually natural language implicitly based on some set-theoretical foundation of mathematics. We have to formalize this meta-language and thus pick an intuitionistic logic on top of simple type theory, which we refer to as HOL. We define it in Sect. 4.3. Then we discuss the adequacy of our encoding in Sect. 4.5. Finally, we prove the soundness of FOL by giving a view from FOL^{pf} to FOL^{mod} in Sect. 4.6.



When encoding signatures and theories in LF, we have the problem that definitions of FOL signatures usually permit arbitrary objects as symbol names. But LF and Twelf expressions have to be words over a countable alphabet. Therefore, we employ two restrictions that are somewhat severe theoretically but natural for applications in computer science. From now on, all FOL theories have a finite number of function and predicate symbols and axioms. It is straightforward to define encodings for infinite theories, but type-theoretical frameworks usually avoid reasoning about infinite signatures. Furthermore, all

function and predicate symbols are chosen from a fixed countable set, and without loss of generality, we assume this set to be the set of legal Twelf identifiers. Thus, we can use the same names in FOL signatures and their encodings.

4.1. Syntax

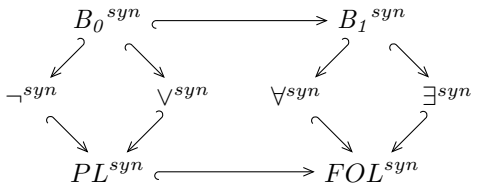


Figure 2: Modular Encoding of FOL Syntax

We encode the signature FOL^{syn} modularly, where each logical connective and quantifier is declared in a separate LF signature. The modular representation of FOL^{syn} is illustrated by the diagram in Fig. 2. Each node corresponds to a LF signature and each edge to an inclusion morphism. $B_0{}^{syn}$ and $B_1{}^{syn}$ are base signatures for propositional and first-order respectively, each connective or quantifier is encoded as an extension of the base signature, and then PL^{syn} for propositional logic and FOL^{syn} for first-order logic are encoded by including the needed fragments.

We only give some of the fragments as examples. The full encoding has one signature each for *true*, *false*, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\forall$, $\exists$ and $\doteq$, and all of those are included into FOL^{syn}.

The LF signatures are given in Fig. 3. In the signature $B_0{}^{syn}$ we introduce the type o for formulas and a type family $ded : o \to \texttt{type}$ for the truth judgment on formulas. In the signatures $\neg^{syn}$ and $\vee^{syn}$ we introduce $\neg$ and $\vee$ respectively. Both $\neg^{syn}$ and $\vee^{syn}$ inherit the symbols o and ded by including $B_0{}^{syn}$. We merge them to form the signature PL^{syn}. The signature $B_1{}^{syn}$ extends $B_0{}^{syn}$ with a type i for terms. We introduce the universal and existential quantifiers in the signatures $\forall^{syn}$ and $\exists^{syn}$, respectively. Finally, we define FOL^{syn} by including the signatures PL^{syn}, $\forall^{syn}$ and $\exists^{syn}$.

We can now encode FOL-signatures as LF-signatures that extend FOL^{syn}. The distinction between signatures and theories is not important from the perspective of LF as the encoding of axioms is very similar to the encoding of function and predicate symbols. Furthermore, we can always consider signatures as the special case of theories without axioms. Therefore, we will unify them and use Σ for both signatures and theories.

Definition 17 (Encoding Syntax)**.** Let Σ be a FOL-signature or theory. We define the LF-encoding Σ^{syn} of Σ as the LF-signature that includes FOL^{syn} and adds the following symbol declarations:

- $f : \underbrace{i \to \ldots \to i}_{n} \to i$ for function symbols f of Σ with $ar(f) = n$,

$\%\texttt{sig}\ {B_0}^{syn} = \{$
$\quad o \quad : \quad \texttt{type}$
$\quad ded \quad : \quad o \to \texttt{type}$
$\}$
$\%\texttt{sig}\ \neg^{syn} = \{$
$\quad \%\texttt{include}\ {B_0}^{syn}$
$\quad \neg \quad : \quad o \to o$
$\}$
$\%\texttt{sig}\ \vee^{syn} = \{$
$\quad \%\texttt{include}\ {B_0}^{syn}$
$\quad \vee \quad : \quad o \to o \to o$
$\}$
$\%\texttt{sig}\ PL^{syn} = \{$
$\quad \%\texttt{include}\ \neg^{syn}$
$\quad \%\texttt{include}\ \vee^{syn}$
$\}$

$\%\texttt{sig}\ {B_1}^{syn} = \{$
$\quad \%\texttt{include}\ {B_0}^{syn}$
$\quad i \quad : \quad \texttt{type}$
$\}$
$\%\texttt{sig}\ \forall^{syn} = \{$
$\quad \%\texttt{include}\ {B_1}^{syn}$
$\quad \forall \quad : \quad (i \to o) \to o$
$\}$
$\%\texttt{sig}\ \exists^{syn} = \{$
$\quad \%\texttt{include}\ {B_1}^{syn}$
$\quad \exists \quad : \quad (i \to o) \to o$
$\}$
$\%\texttt{sig}\ FOL^{syn} = \{$
$\quad \%\texttt{include}\ PL^{syn}$
$\quad \%\texttt{include}\ \forall^{syn}$
$\quad \%\texttt{include}\ \exists^{syn}$
$\}$

Figure 3: LF Signatures for FOL Syntax

- $p : \underbrace{i \to \ldots \to i}_{n} \to o$ for predicate symbols p of Σ with $ar(p) = n$,
- $a : ded\ \ulcorner F \urcorner$ for axioms F of Σ and some fresh name a.

Here $\ulcorner F \urcorner$ is the encoding of $F \in Sen(\Sigma)$. Every Σ-term t or formula F in context $x_1, \ldots, x_n$ is encoded as an LF term $\ulcorner t \urcorner : i$ or $\ulcorner F \urcorner : o$, respectively, in context $x_1 : i, \ldots, x_n : i$. $\ulcorner t \urcorner$ and $\ulcorner F \urcorner$ are defined by an obvious induction, and we only give the case of quantifiers as an example:

$$\ulcorner \forall x\ F \urcorner = \forall [x : i] \ulcorner F \urcorner.$$

Definition 18 (Encoding Signature Morphisms)**.** Let $\sigma : \Sigma \to \Sigma'$ be a FOL signature morphism. Its LF-encoding is the LF signature morphism $\sigma^{syn} : {\Sigma}^{syn} \to {\Sigma'}^{syn}$ that maps all symbols of FOL^{syn} to themselves and every function or predicate symbol s of Σ to $\sigma(s)$.

Example 19. $Monoid^{syn}$ is the encoding of the theory *Monoid* from Ex. 5. For example, the binary function symbol $\circ$ in *MonSig* is encoded as the symbol $\circ : i \to i \to i$ that takes two arguments of LF-type i and returns an LF-term of type i. $Group^{syn}$ is defined accordingly.

$\%\texttt{sig}\ \mathit{Monoid}^{syn} = \{$
  $\%\texttt{include}\ \mathit{FOL}^{syn}$
  $\circ \quad : \quad i \to i \to i \qquad \%\texttt{infix}\ \circ$
  $e \quad : \quad i$
  $\mathit{assoc} \quad : \quad \mathit{ded}\ \forall\,[x]\,\forall\,[y]\,\forall\,[z]\,x \circ (y \circ z) \doteq (x \circ y) \circ z$
  $\mathit{neut}_l \quad : \quad \mathit{ded}\ \forall\,[x]\,(e \circ x) \doteq x$
  $\mathit{neut}_r \quad : \quad \mathit{ded}\ \forall\,[x]\,(x \circ e) \doteq x$
$\}$

4.2. Proof Theory

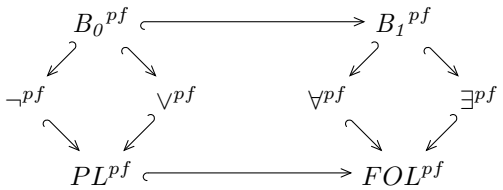


Figure 4: Modular Encoding of FOL Proof Theory

The encoding of the FOL proof theory has the same modular structure as the encoding of the FOL syntax. This is illustrated in Fig. 4 where each node has one additional – not displayed – inclusion from its counterpart in Fig. 2.

The signatures ${B_0}^{pf}$ and ${B_1}^{pf}$ typically contain the structural rules of the chosen calculus. In our case, they are equal to the signature ${B_0}^{syn}$ and ${B_1}^{syn}$ because encodings of natural deduction calculi in LF automatically inherit weakening, exchange, and contraction rules from LF. We distinguish these signatures anyway because of the conceptual clarity, and because the analogues of ${B_0}^{pf}$ or ${B_1}^{pf}$ in the encodings of other logics do contain additional declarations, e.g., when using sequent or tableaux calculi.

$\%\texttt{sig}\ {B_0}^{pf} = \{$
  $\%\texttt{include}\ {B_0}^{syn}$
$\}$
$\%\texttt{sig}\ {B_1}^{pf} = \{$
  $\%\texttt{include}\ {B_0}^{pf}$
  $\%\texttt{include}\ {B_1}^{syn}$
$\}$

The signatures $\neg^{pf}$, $\forall^{pf}$, etc. encode the introduction and elimination rules for the individual connectives. We refer to [HHP93] for details about the encoding of proof rules and only give disjunction as an example.

$$
\begin{array}{l}
\%\texttt{sig}\ \vee^{pf}\ =\ \{ \\
\quad \%\texttt{include}\ B_0{}^{pf} \\
\quad \%\texttt{include}\ \vee^{syn} \\
\quad orI_l \quad : \quad ded\ \ F \ \rightarrow\ ded\ \ F \vee G \\
\quad orI_r \quad : \quad ded\ \ G \ \rightarrow\ ded\ \ F \vee G \\
\quad orE \quad : \quad ded\ \ F \vee G \ \rightarrow\ (ded\ \ F \rightarrow ded\ \ H) \ \rightarrow\ (ded\ \ G \rightarrow ded\ \ H) \\
\qquad\qquad \rightarrow\ ded\ \ H \\
\}
\end{array}
$$

Finally, the signatures PL^{pf} and FOL^{pf} collect the fragments in almost the same way as for the syntax encodings. The only difference in PL^{pf} is that the law of excluded middle is added:

$$
\begin{array}{l}
\%\texttt{sig}\ PL^{pf}\ =\ \{ \\
\quad \%\texttt{include}\ \neg^{pf} \\
\quad \%\texttt{include}\ \vee^{pf} \\
\quad tnd \quad : \quad ded\ F \vee \neg F \\
\}
\end{array}
$$

Similarly, there is one further proof rule added to FOL^{pf}: the axiom $ded\ \ \exists[x]\, true$. This axiom may be surprising because it is redundant in usual axiomatizations of FOL. This redundancy is due to the $\forall$ elimination rule, which can instantiate $\forall[x]\, true$ with any term including fresh variables. This amounts to assuming non-emptiness of the universe. In LF, only terms that are well-formed in the current context are eligible so that we obtain a system that is complete for a variant of FOL where the universe may be empty. Therefore, we explicitly add an axiom to make it non-empty.

Definition 20. For every FOL-signature or theory Σ, the LF signature Σ^{pf} is the canonical pushout of the diagram below.

$$
\begin{array}{ccc}
 & & FOL^{pf} \hookrightarrow \Sigma^{pf} \\
 & \pi \nearrow & \quad\ \pi_\Sigma \nearrow \\
FOL^{syn} & \hookrightarrow & \Sigma^{syn}
\end{array}
$$

Here by canonical we mean that Σ^{pf} includes FOL^{pf} and adds a declaration $c : \pi_\Sigma(A)$ for every declaration $c : A$ that Σ^{syn} adds to FOL^{syn}. π_Σ maps FOL^{syn}-symbols according to π and other symbols to themselves.

Note that in the special case of FOL, both π and π_Σ are inclusions.

We have the Curry-Howard representation of proofs as terms:

Definition 21 (Encoding Proofs)**.** For a FOL-theory Σ, every derivation p of $F_1, \ldots, F_n \ \vdash_\Sigma \ F$ is encoded as as an LF term $\ulcorner p \urcorner : ded\ \ulcorner F_1 \urcorner \rightarrow \ldots \rightarrow ded\ \ulcorner F_n \urcorner \rightarrow ded\ \ulcorner F \urcorner$ over Σ^{pf} by a straightforward induction.

4.3. A Meta-Language for the Representation of Model Theory

Foundations. As all formal representations, the representation of the model theory requires a suitable meta-language. For the syntax and proof theory, LF is a very appropriate meta-language – this is not surprising because it is what LF was designed to be. LF only offers minimal syntactic means: judgments as types with implication (via $\to$) and universal quantification (via Π), and the former is just a special case of the latter. Still, it has been quite successful at covering a large class of logics because the syntax and proof theory of a logic are often (and in fact should usually be – see [ML96]) defined in terms of a grammar, judgments about its expressions, and an inference system for these judgments. LF can be seen as the result of applying Occam's razor to these requirements.

The situation is different for the representation of model theory. Models are described in the language of mathematics, and it is difficult to formalize this language without making a foundational commitment. For example, we could chose a variant of set theory (as in [PC93, TB85]), higher-order logic ([Chu40], as in [Gor88, NPW02, Har96]), Martin-Löf type theory ([ML74], as in [Nor05]), or the calculus of constructions ([CH88], as in [BC04, ACTZ06]), all of which provide implementations with strong computational support. LF, on the other hand, is too weak to be such a foundation because the type theory is minimalistic and the use of higher-order abstract syntax is incompatible with the natural way of adding computational power.

However, because of this weakness, LF can serve as a minimal, neutral framework in which to formalize the foundation itself. Moreover, since the choice of foundation changes the notion of model (and thus possibly the truth of a statement about models), an encoding can only be adequate relative to a fixed foundation. For example, consider first-order logic with models taken in a set theory with large cardinals. Therefore, it is even desirable to make the foundation part of the encoding. This also has the benefit that it becomes possible to build foundations modularly and to compare and translate between them. For example, we have formalized Mizar and Isabelle/HOL along with translations into ZFC set theory [KMR09].

Approximating Foundations. In this paper, we choose ZFC set theory as the foundation because it is the standard foundation of mathematics. Therefore, we encode ZFC in LF and use it as the meta-language to define models. However, ZFC behaves badly computationally because it is engineered towards elegance and simplicity rather than decidability or efficiency. Therefore, we also use a second meta-language – a variant of higher-order logic (HOL) – which can be worked with efficiently.

The intuition is that ZFC gives the official definition, and HOL is a sound but incomplete approximation of ZFC. Using the module system, we can state the relation between HOL and ZFC precisely by giving an LF signature morphism φ from HOL to ZFC. Via φ, we can regard ZFC as a refinement of HOL or HOL as a fragment of ZFC. Of course, HOL is not complete (relative to standard models in ZFC) and thus does not necessarily yield adequate encodings of model

theory. However, for certain results, working with HOL is sufficient, and then it is preferable.

Moreover, we can construct a chain of foundations of increasing strength, e.g., HOL $\to$ ZF $\to$ ZFC, and always work in the weakest possible foundation. This is in keeping with mathematical practice not to commit to a specific foundation unless necessary, and leads to an approach we call *little foundations* (inspired by [FGT92]). For example, our encodings in [KMR09], avoid the use of excluded middle and the axiom of choice whenever possible.

In this section, we will only consider HOL, which we introduce below and use as the meta-language to represent models in Sect. 4.4. In Sect. 5, we encode ZFC set theory in LF, refine our HOL-based semantics into a ZFC-based one, and revisit the encoding of model theory.

HOL as a Meta-Language. For the representation of first-order logic, we need the booleans *bool*, an arbitrary set *univ* (the universe), and functions between the universe and the booleans. Among the latter are functions from $univ^n$ to *univ*, from $univ^n$ to *bool*, and from *bool* and $bool^2$ to *bool* interpreting the function symbols, predicate symbols, and the propositional connectives, respectively. Furthermore, the quantifiers must be interpreted as second-order functions from $bool^{univ}$ to *bool*. Finally, these functions should be typed, and that leads us to the choice of HOL as the meta-language.

$$
\begin{array}{lcl}
\%\texttt{sig}\ HOL\ =\ \{ & & \\
\quad set & : & \texttt{type} \\
\quad \Longrightarrow & : & set \to set \to set \\
\quad elem & : & set \to \texttt{type} \\
\quad \lambda & : & (elem\ A \to elem\ B) \to elem\ (A \Longrightarrow B) \\
\quad @ & : & elem\ (A \Longrightarrow B) \to elem\ A \to elem\ B \\
\quad prop & : & \texttt{type} \\
\quad True & : & prop \to \texttt{type} \\
\quad true & : & prop \\
\quad false & : & prop \\
\quad \wedge & : & prop \to prop \to prop \\
\quad \Rightarrow & : & prop \to prop \to prop \\
\quad \neg & : & prop \to prop \\
\quad \vee & : & prop \to prop \to prop \\
\quad \Leftrightarrow & : & prop \to prop \to prop \\
\quad \doteq & : & elem\ A \to elem\ A \to prop \\
\quad \forall & : & (elem\ A \to prop) \to prop \\
\quad \exists & : & (elem\ A \to prop) \to prop \\
\} & &
\end{array}
$$

Figure 5: Encoding of HOL

To define HOL, we encode it as the LF signature given in Fig. 5. Actually, we

only give a partial signature here and omit all the proof rules. The full version of the encoding of HOL can be found at [KMR09]. Note that this signature extends the running example of Sect. 2.2 except that we write *set* and *elem* instead of *tp* and *tm* to emphasize the relation to set theory. *set* is the type of sets, and $\Longrightarrow$ gives the set of functions between two sets. $elem\ A$ is the type of elements of set A – this lets us reason about the elements of a set without using the $\in$ relation. HOL must be a sound but not necessarily a complete fragment of set theory: Thus, the relation $a : elem\ A$ must imply $a \in A$, but the inverse does not have to hold. Then λ and @ encode function formation and application. This yields the standard encoding of simple type theory in LF.

Finally, *prop* is the type of propositions. The propositional connectives are declared in the usual way. Equality and the quantifiers take an implicit argument A for the set which they operate on. Note that this means that we use equality only between elements of the same set and only use bound quantifiers. We omit the proof rules for HOL here and only state that we use rules for β and η conversion, and natural deduction introduction and elimination rules for the connectives and quantifiers. Equality is axiomatized as a congruence relation on each type. We do not assume the axiom of excluded middle – this turns out to suffice to axiomatize FOL models and makes us more flexible because we are not a priori committed to classical foundations of mathematics.

It is interesting to note that the actual encoding in Twelf is a little different because it already benefits from the LF module system: Our logic library represents the syntax and proof theory of the propositional connectives once and for all, and they are imported both into the object logic FOL and into the meta-language HOL.

The use of HOL as the meta-language means that the model theory of the object language (e.g., FOL) is represented as an extension of the signature HOL, i.e., a HOL-theory. Therefore, we define:

Definition 22 (HOL-theories)**.** A theory of HOL is an LF signature that includes the signature HOL and that only adds declarations of the following forms:

- base sets: $S : set$,
- constant symbols: $c : elem\ S$ for some set S,
- axioms: $a : True\ F$ for some proposition F.

While the model theory of the object language is represented as a HOL-theory, every individual model is represented as a HOL model. Therefore, we have to define HOL models as well. In Sect. 5, we will show how models can be encoded as syntactical entities of LF, but here we will do something simpler and define (standard) HOL models as platonic objects in an underlying set theoretical universe:

Definition 23 (HOL-Models)**.** Assume a fixed set-theoretical universe. A model M of an HOL-theory T is a mapping that assigns:

- to every base set $S : \mathit{set}$ a set S^M,
- to every constant symbol $c : \mathit{elem}\ S$ an element $c^M \in S^M$,

such that F^M is true for every axiom $a : \mathit{True}\ F$. Here S^M is obtained by recursively replacing $(S_1 \Longrightarrow S_2)^M$ with the set of functions from S_1^M to S_2^M. And F^M is obtained by replacing every base set S or constant c occurring in F with S^M or c^M, respectively, and evaluating the result as a proposition over the set theoretical universe.

Def. 23 is somewhat vague. For sake of definiteness, we can assume ZFC as the underlying set theory:

Example 24 (HOL-Models in ZFC). If our underlying set theory is ZFC, we can define F^M as follows:

1. replace all S and c with S^M and c^M, respectively,
2. relativize all quantifiers, i.e., $\forall\,[x : \mathit{elem}\ S]\,F$ becomes $\forall x \in S^M.\, F^M$,
3. replace $S \Longrightarrow S'$ with the set of functions from S^M to S'^M, and replace λ and @ with function formation and function application.

Then we are ready to represent the model theory of object logics in HOL in Sect. 4.4.

4.4. Model Theory

In this section we present our representation of FOL model theory in LF. Our encoding of FOL model theory consists two parts as illustrated in the diagram below: The specification of FOL models (given by the signature FOL^{mod}) and the interpretation of FOL syntax in terms of the semantics (given by the view μ from FOL^{syn} to FOL^{mod}).

$$\begin{array}{ccc} FOL^{syn} & \xrightarrow{\ \mu\ } & FOL^{mod} \\ & & \uparrow \\ & & HOL \end{array}$$

Both FOL^{mod} and μ follow the same modular structure as in FOL^{syn} and FOL^{pf}, however, for the sake of simplicity, we will present the flat version of our encoding in this paper.

Specification of FOL Models. In FOL^{mod} we first encode the model theoretical notion of truth and the universe of a model. We declare a set *bool* which represents the set $\{0, 1\}$ of truth values and axiomatize this by declaring 0 and 1, and axioms ax_{cons} and ax_{boole} to state that *bool* is indeed the desired 2-element set:

$$\begin{array}{lcl} \mathit{bool} & : & \mathit{set} \\ 0 & : & \mathit{elem}\ \mathit{bool} \\ 1 & : & \mathit{elem}\ \mathit{bool} \\ ax_{cons} & : & \mathit{True}\ \neg\,(0 \doteq 1) \\ ax_{boole} & : & \{F\}\ \ \mathit{True}\ (F \doteq 0\ \vee\ F \doteq 1) \end{array}$$

Recall that $\mathit{True} : \mathit{prop} \to \texttt{type}$ is the truth judgment of the meta-language HOL, that $\neg$ and $\vee$ are the negation and disjunction on HOL propositions, and

$\doteq$ is the typed-equality of HOL terms. In particular, these symbols are different from the symbols of the same name in the syntax of FOL.

We declare the symbol *univ* as a set for the universe of a FOL-model, and add an axiom making *univ* non-empty.

$$\begin{array}{lcl} \mathit{univ} & : & \mathit{set} \\ \mathit{nonemp} & : & \mathit{True}\ \exists\,[x : \mathit{elem}\,\mathit{univ}]\ \mathit{true} \end{array}$$

Next we encode the semantics of the logical symbols introduced in FOL^{syn}. For each logical symbol s^{syn} in FOL^{syn}, we declare a symbol s^{mod}, which represents the semantic operation used to interpret s^{syn} along with axioms specifying its values. This corresponds to the case-based definition of the semantics of a formula.

As examples we present the cases for $\vee$ and $\forall$. For the interpretation of $\vee$, we declare the symbol *or* as a HOL-function from bool^2 to *bool* and axiomatize it to be the binary supremum in the boolean 2-element lattice.

$$\begin{array}{lcl} \mathit{or} & : & \mathit{elem}\,(\mathit{bool} \Longrightarrow \mathit{bool} \Longrightarrow \mathit{bool}) \\ \mathit{or1} & : & \mathit{True}\,((F \doteq 1 \vee G \doteq 1)\ \Rightarrow\ (\mathit{or}\,@\,F\,@\,G) \doteq 1) \\ \mathit{or0} & : & \mathit{True}\,((F \doteq 0 \wedge G \doteq 0)\ \Rightarrow\ (\mathit{or}\,@\,F\,@\,G) \doteq 0) \end{array}$$

Similarly, for the interpretation of $\forall$, we specify the function *forall* that takes a *univ*-indexed family F of booleans and returns its infimum, i.e., it returns 1 iff all $F@x$ is 1 for all x.

$$\begin{array}{lcl} \mathit{forall} & : & \mathit{elem}\,((\mathit{univ} \Longrightarrow \mathit{bool}) \Longrightarrow \mathit{bool}) \\ \mathit{forall1} & : & \{F : \mathit{elem}\,(\mathit{univ} \Longrightarrow \mathit{bool})\} \\ & & \mathit{True}\,(\forall\,[x : \mathit{elem}\ \mathit{univ}]\,(F\,@\,x) \doteq 1\ \Rightarrow\ (\mathit{forall}\,@\,F) \doteq 1) \\ \mathit{forall0} & : & \{F : \mathit{elem}\,(\mathit{univ} \Longrightarrow \mathit{bool})\} \\ & & \mathit{True}\,(\exists\,[x : \mathit{elem}\ \mathit{univ}]\,(F\,@\,x) \doteq 0\ \Rightarrow\ (\mathit{forall}\,@\,F) \doteq 0) \end{array}$$

An overview of the operations and axioms declared for the remaining connectives and quantifiers is given in Fig. 6. In all cases except for equality, we have two axioms of the form $C_0 \Rightarrow F \doteq 0$ and $C_1 \Rightarrow F \doteq 1$, e.g., *and0* for when a conjunction is false and *and1* for when it is true. For equality, our results below require a slightly stronger condition, namely the axiom *eq1* of the form $C_1 \Leftrightarrow F \doteq 1$ (from which the corresponding $\neg C_1 \Leftrightarrow F \doteq 0$ can be derived).

Interpretation Function. The idea of the view μ is that it maps from the syntax to the semantics; it gives the cases of the interpretation function for the logical symbols. This takes the form of a view from FOL^{syn} to FOL^{mod}, which must give a FOL^{mod}-expression for all symbols declared in or included into FOL^{syn}.

Formulas are interpreted as boolean truth values, and we map the type o of formulas to the type *elem bool* of truth values. The truth value 1 is designated, i.e., represents truth. Therefore, we map *ded* to a type family that takes an argument F of type *elem bool* and returns the judgment that F is equal to 1. FOL-terms are interpreted as elements of the universe. Therefore, we map the type i of FOL-terms to the type *elem univ* of elements of *univ*.

true	:	$elem\ bool$
true1	:	$True\ true \doteq 1$
false	:	$elem\ bool$
false0	:	$True\ false \doteq 0$
not	:	$elem\,(bool \Longrightarrow bool)$
not0	:	$True\,(F \doteq 0\ \Rightarrow\ (not\,@\,F) \doteq 1)$
not1	:	$True\,(F \doteq 1\ \Rightarrow\ (not\,@\,F) \doteq 0)$
and	:	$elem\,(bool \Longrightarrow bool \Longrightarrow bool)$
and1	:	$True\,(F \doteq 1 \wedge G \doteq 1\ \Rightarrow\ (and\,@\,F\,@\,G) \doteq 1)$
and0	:	$True\,(F \doteq 0 \vee G \doteq 0\ \Rightarrow\ (and\,@\,F\,@\,G) \doteq 0)$
impl	:	$elem\,(bool \Longrightarrow bool \Longrightarrow bool)$
impl1	:	$True\,(F \doteq 0 \vee G \doteq 1\ \Rightarrow\ (impl\,@\,F\,@\,G) \doteq 1)$
impl0	:	$True\,(F \doteq 1 \wedge G \doteq 0\ \Rightarrow\ (impl\,@\,F\,@\,G) \doteq 0)$
exists	:	$elem\,((univ \Longrightarrow bool) \Longrightarrow bool)$
exists1	:	$\{F : elem\,(univ \Longrightarrow bool)\}$
		$True\,(\exists\,[x]\,(F\,@\,x) \doteq 1\ \Rightarrow\ (exists\,@\,F) \doteq 1)$
exists0	:	$\{F : elem\,(univ \Longrightarrow bool)\}$
		$True\,(\forall\,[x]\,(F\,@\,x) \doteq 0\ \Rightarrow\ (exists\,@\,F) \doteq 0)$
eq	:	$elem\,(univ \Longrightarrow univ \Longrightarrow bool)$
eq1	:	$True\,(F \doteq G\ \Leftrightarrow\ (eq\,@\,F\,@\,G) \doteq 1)$

Figure 6: Specification of FOL Models

$$
\begin{array}{lcl}
o & := & elem\ bool \\
ded & := & [F : elem\ bool]\ True\ (F \doteq 1) \\
i & := & elem\ univ
\end{array}
$$

The interpretation of the logical connectives is as expected. For example, the disjunction $F \vee G$ is interpreted by applying *or* to $\mu(F)$ and $\mu(G)$. And the universal quantification $\forall\ ([x : i]\,F)$ is interpreted as $\mu(\forall)\ \mu([x : i]\,F) = forall\,@\,\lambda\,([x : elem\ \ univ]\,\mu(F))$.

$$
\begin{array}{lcl}
\vee & := & [F : elem\ bool]\,[G : elem\ bool]\,or\,@\,F\,@\,G \\
\forall & := & [F : elem\,(univ \Longrightarrow bool)]\,forall\,@\,(\lambda\,F)
\end{array}
$$

Models of FOL-Theories. Finally we can define Σ^{mod} just like Σ^{pf}:

Definition 25. For every FOL-signature or theory Σ, the LF signature Σ^{mod} is the canonical pushout in the diagram below.

$$
\begin{array}{ccc}
FOL^{mod} & \hookrightarrow & \Sigma^{mod} \\
{\scriptstyle\mu}\nearrow & & \nearrow{\scriptstyle\mu_\Sigma} \\
FOL^{syn} & \hookrightarrow & \Sigma^{syn}
\end{array}
$$

Here by canonical we mean that Σ^{mod} includes FOL^{mod} and adds a declaration $c : \mu_\Sigma(A)$ for every declaration $c : A$ that Σ^{syn} adds to FOL^{syn}. μ_Σ maps FOL^{syn}-symbols according to μ and other symbols to themselves.

Example 26 (Continued). The signature $Group^{mod}$ looks as follows:

$$
\begin{array}{l}
\%\texttt{sig}\ Group^{mod} = \{ \\
\quad \%\texttt{include}\ FOL^{mod} \\
\quad \circ \quad : \quad elem\ univ \ \to \ elem\ univ \ \to elem\ univ \\
\quad e \quad : \quad elem\ univ \\
\quad inv \quad : \quad elem\ univ \ \to \ elem\ univ \\
\quad assoc \quad : \quad True\ \big(forall\,@\,(\lambda[x]\,forall\,@\,(\lambda[y]\,forall\,@\,(\lambda[z] \\
\qquad\qquad eq\,@\,(x \circ (y \circ z))\,@\,((x \circ y) \circ z))))\big) \doteq 1 \\
\quad \vdots \\
\}
\end{array}
$$

Notation 27. Note that the canonical pushout yields declarations $\circ : elem\ univ \to elem\ univ \to elem\ univ$ rather than $\circ : elem\,(univ \Longrightarrow univ \Longrightarrow univ)$. Thus, $Group^{mod}$ is technically not a HOL-theory in the sense of Def. 22. However, we can give signature morphisms back and forth between these. For example, if we have a signature with $\circ' : elem\,(univ \Longrightarrow univ \Longrightarrow univ)$, the morphisms map $\circ'$ to $\lambda[x]\,\lambda[y]\,x \circ y$ and $\circ$ to $[x]\,[y]\,\circ'\,@\,x\,@\,y$. In the following, we will assume that Σ^{mod} extends FOL^{mod} with declarations of the form $c : elem\ S_1 \Longrightarrow \ldots \Longrightarrow S_n$ and omit the connecting signature morphisms from the notation.

Definition 28 (Encoding Models)**.** Assume a fixed set theory in which FOL and HOL-models are defined, and pick two arbitrary sets $\mathcal{F}$ and $\mathcal{T}$ as the truth values. Then for every FOL-signature Σ and every Σ-model $M = (U, I)$, we define a HOL-model $\ulcorner M \urcorner$ of Σ^{mod} as follows:

$$0^{\ulcorner M \urcorner} = \mathcal{F} \qquad 1^{\ulcorner M \urcorner} = \mathcal{T} \qquad bool^{\ulcorner M \urcorner} = \{\mathcal{F}, \mathcal{T}\} \qquad univ^{\ulcorner M \urcorner} = U,$$

$not^{\ulcorner M \urcorner}$, $and^{\ulcorner M \urcorner}$, $or^{\ulcorner M \urcorner}$, $impl^{\ulcorner M \urcorner}$, $eq^{\ulcorner M \urcorner}$, $forall^{\ulcorner M \urcorner}$, and $exists^{\ulcorner M \urcorner}$ are defined in the obvious way, and for function symbols f and predicate symbols p we put

$$f^{\ulcorner M \urcorner} = f^I \qquad \text{and} \qquad p^{\ulcorner M \urcorner}(u_1, \ldots, u_n) = \mathcal{T} \text{ iff } (u_1, \ldots, u_n) \in p^I.$$

It is easy to check that this is indeed a HOL-model of Σ^{mod}, i.e., satisfies all the axioms of Σ^{mod}.

We could also encode model translations, but we can do this more elegantly in Sect. 5.

Example 29 (Continued). Consider the model Int from Ex. 15. It is encoded as a HOL-model of $Group^{mod}$ by putting $univ^{\ulcorner Int \urcorner} = \mathbb{Z}$, $\circ^{\ulcorner Int \urcorner} = +$, $e^{\ulcorner Int \urcorner} = 0$, and $inv^{\ulcorner Int \urcorner} = -$. The interpretations of all other symbols are uniquely determined.

4.5. Adequacy

In the previous sections, we have defined the LF signatures and morphisms $(\mathrm{FOL}^{syn}, \mathrm{FOL}^{pf}, \pi, \mathrm{FOL}^{mod}, \mu)$ intended to encode FOL. (π is simply an inclusion for FOL, but we will keep it in the notation to stress that FOL is only an example for a generic method.) Showing that such an encoding is adequate

means to show that the encoding has the same properties as the encoded logic. If an encoding is adequate, meta-logical results reached by reasoning about the logic encoding are guaranteed to hold for the encoded logic as well.

To give a general formal definition what adequacy means, we need to do three things: (i) define in general what a logic is and when two logics are isomorphic, (ii) define in general what a logic encoding in LF is and how every such logic encoding induces a logic, and then (iii) for a specific logic encoding show that the induced logic is isomorphic to the encoded logic. Especially (ii) requires a large amount of work, which we carried out in [Rab08]. For simplicity, here, we will only consider the special case of adequacy of our encoding of FOL. For other logic encodings, the procedure is the same.

We begin by restating some known results for the adequacy of syntax and proof theory in our notation. See e.g. [HHP93, HST94], the encodings used there are not modular, but that is a minor difference that does not affect the proofs.

Theorem 30 (Adequacy for Syntax)**.** *For every FOL-signature Σ and context $\Gamma = x_1, \ldots, x_n$, the formulas are in a natural bijection with the $\beta\eta$-normal LF-terms of type o over Σ^{syn} in context $x_1 : i, \ldots, x_n : i$.*

Theorem 31 (Adequacy for Signature Morphisms)**.** *For every FOL-signature morphism $\sigma : \Sigma \to \Sigma'$ and every sentence $F \in Sen(\Sigma)$, we have $\ulcorner \sigma(F) \urcorner = \sigma^{syn}(\ulcorner F \urcorner)$.*

Theorem 32 (Adequacy for Proof Theory)**.** *For every FOL-theory Σ, the derivations of $F_1, \ldots, F_n \vdash_\Sigma F$ are in a natural bijection with the $\beta\eta$-normal LF-terms of type $\pi_\Sigma(ded\ \ulcorner F_1 \urcorner \to \ldots \to ded\ \ulcorner F_n \urcorner \to ded\ \ulcorner F \urcorner)$ over Σ^{pf}.*

In particular, F is a Σ-theorem iff $\pi_\Sigma(ded\ \ulcorner F \urcorner)$ is inhabited over Σ^{pf}.

Theorem 33 (Adequacy for Proof Theoretical Theory Morphisms)**.** *For a FOL-signature morphism $\sigma : \Sigma \to \Sigma'$, we have $\sigma : (\Sigma, \Theta) \xrightarrow{\mathrm{P}} (\Sigma', \Theta')$ iff $\sigma^{pf} : \Sigma^{pf} \to \Sigma'^{pf}$ can be extended to an LF signature morphism $(\Sigma, \Theta)^{pf} \to (\Sigma', \Theta')^{pf}$.*

Proof. This follows from Thm. 32 by extending σ^{pf} such that every $a : \pi_\Sigma(ded\ \ulcorner F \urcorner)$ occurring in $(\Sigma, \Theta)^{pf}$ is mapped to $\ulcorner p \urcorner$ for some proof p of $Sen(\sigma)(F)$. □

To give similar results for the model theory, we need a bijection between FOL-models of Σ and HOL-models of Σ^{mod}. First we prove that every single model of FOL^{mod} is adequate in the following sense:

Lemma 34. *Assume a HOL-model M of FOL^{mod}. Then $bool^M = \{0^M, 1^M\}$ is a two-element set, $univ^M$ is an arbitrary non-empty set, and $true^M$, $false^M$, and^M, or^M, $impl^M$, eq^M, $forall^M$, and $exists^M$ are the usual operations in the semantics of first-order logic with respect to the universe $univ^M$ and the truth values 0^M for falsity and 1^M for truth.*

Proof. Straightforward using the axioms in FOL^{mod}. □

Note that the interpretations of the booleans (and thus of the remaining operations on them) are not determined uniquely because the specific choice of truth values remains free. If we wanted to characterize them uniquely, e.g., $0^M = \varnothing$ and $1^M = \{\varnothing\}$, we would need to have more access to the underlying set theory than provided by HOL. However, once two arbitrary truth values and the universe are fixed, the interpretation of the connectives and quantifiers is determined. Then we can state the adequacy of the model theory as follows:

Theorem 35 (Adequacy for Model Theory)**.** *Assume a fixed set theory in which FOL- and HOL-models are defined, and pick two arbitrary sets $\mathcal{F}$ and $\mathcal{T}$. Let us call a FOL^{mod}-model* normal *if* 0 *and* 1 *are interpreted as $\mathcal{F}$ and $\mathcal{T}$.*

Then for every FOL-signature Σ, there is a natural bijection between FOL-models M of Σ and normal HOL models of Σ^{mod}. Furthermore, for every M and every Σ-sentence F, we have

$$M \models_\Sigma F \qquad \text{iff} \qquad (\mu_\Sigma(ded\ \ulcorner F \urcorner))^{\ulcorner M \urcorner}.$$

Recall that for FOL, we have $\mu_\Sigma(ded\ \ulcorner F \urcorner) = True\ (\mu_\Sigma(\ulcorner F \urcorner)\ \doteq\ 1)$.

Proof. One direction of the bijection is the encoding $M \mapsto \ulcorner M \urcorner$. For the inverse direction, assume a normal HOL-model M of Σ^{mod}. Because M is normal and because of Lem. 34, M has no freedom but to pick a non-empty set for $univ^M$ and operations for f^M and p^M. It is easy to see that each such choice yields a FOL-model of Σ, and that the two functions are bijections.

The second claim follows by a straightforward induction on F using the axioms of FOL^{mod}. □

It is tempting to assume that, parallel to Thm. 33, $\sigma : (\Sigma, \Theta) \xrightarrow{\mathtt{M}} (\Sigma', \Theta')$ can be encoded using LF-signature morphisms $(\Sigma, \Theta)^{mod} \to (\Sigma', \Theta')^{mod}$. But this is not the case: The existence of such morphisms is only sufficient but not necessary for σ to be a model-theoretical theory morphism. This is because HOL is not complete with respect to standard models. We will get back to this in Sect. 5.

Taking all these adequacy results together, we see that all results about FOL that can be stated in terms of encodings of syntax, proof theory, and model theory, carry over to FOL. As an example, let us consider soundness and completeness.

Theorem 36. *For a FOL-signature Σ and LF terms F_i, F, define*

(i) There is a term of type $\pi_\Sigma(ded\ \ F_1 \to \ldots \to ded\ \ F_n \to ded\ \ F)$ over Σ^{pf}.

(ii) For every HOL-model M of Σ^{mod}, if $(\mu_\Sigma(F_i))^M$ for all i, then also $(\mu_\Sigma(F))^M$.

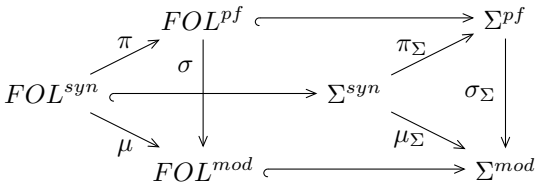
Then the logic FOL is sound iff (i) implies (ii), and complete iff (ii) implies (i).

Proof. Immediately using the adequacy of proof and model theory. □

Among all the meta-logical properties that can be studied after encoding a logic in LF, soundness is particularly interesting because we have the following result:

Theorem 37. *If there is a signature morphism σ from FOL^{pf} to FOL^{mod} such that $\pi\ \sigma = \mu$, then the logic FOL is sound.*

Proof. We proceed in two steps. First, we show that for every FOL-signature Σ there is a signature morphism σ_Σ from Σ^{pf} to Σ^{mod} such that the following diagram commutes:



σ_Σ is simply the universal morphism factoring σ and μ_Σ through the pushout Σ^{pf}.

Secondly, we show soundness using Thm. 36. So assume (i). Since signature morphisms are type-preserving mappings, there must be a term of type $\sigma_\Sigma(\pi_\Sigma(ded\ \ F_1 \to \ldots \to ded\ \ F_n \to ded\ \ F))$ over Σ^{mod}. Because $\pi_\Sigma\ \sigma_\Sigma = \mu_\Sigma$, this type is equal to $\mu_\Sigma(ded\ \ F_1) \to \ldots \to \mu_\Sigma(ded\ \ F_n) \to \mu_\Sigma(ded\ \ F)$. Now the implication introduction rule of HOL shows that (ii) holds. □

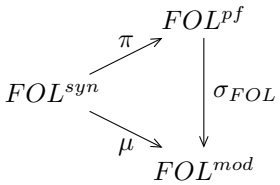
These results may be criticized as being implications between statements known to be true. But recall that the same methodology can be applied to a very wide variety of other logics, and we obtain the corresponding result for every such logic.

In general, Thm. 37 is only a sufficient criterion. It cannot be necessary for all logic encodings because HOL is only a (sound but incomplete) fragment of set theory, which may or may not be strong enough to carry out the soundness proof for the encoded logic. However, in our experience such a morphism typically exists for reasonable choices of the meta-language.

It is tempting to look for the analogue of Thm. 37 for *completeness*. But a morphism $\Sigma^{mod} \to \Sigma^{pf}$ can usually not be given. For example, FOL^{mod} is significantly more expressive than FOL^{pf} and therefore cannot be interpreted in FOL^{pf}. Even when such a morphism exists, it does not imply completeness. But it is promising to investigate other ways to encode completeness, which we leave to future work. For example, the central idea of many completeness proofs is to construct a canonical model whose objects are given by the syntax of the logic. Since LF provides an excellent way to talk about syntax, it is interesting to use LF to form a canonical model. If the syntax of LF is be reflected into the representation of the model theory, it can yield a general way of formalizing completeness proofs.

4.6. Soundness

Now we will apply Thm. 37 to encode the soundness proof of FOL in LF by giving a view σ_{FOL} from FOL^{pf} to FOL^{mod}. The structure of σ_{FOL} follows the modular structure of FOL^{pf}, i.e., the soundness is proved separately for every connective or quantifier. In particular, σ_{FOL} includes the view μ for all symbols of FOL^{syn}. Thus, the LF module system guarantees the commutativity of the diagram on the right.



The remaining symbols of FOL^{pf} are those encoding proof rules. Each of those must be mapped to a proof term in FOL^{mod}. This is straightforward, and we only give one example case and refer [KMR09] for the remaining cases.

The proof rule $orI_l : ded\ F \to ded\ F \vee G$ is included into FOL^{pf} from $\vee^{pf}$. It uses two implicit arguments $F : o$ and $G : o$ and must be mapped to a FOL^{mod}-term of type

$$\{F : elem\ bool\}\,\{G : elem\ bool\}\ True\ F \doteq 1 \ \to\ True\ (or\ @\ F\ @\ G) \doteq 1$$

Its map is given by

$$orI_l \ :=\ [F : elem\ bool]\,[G : elem\ bool]\,[p : True\ F \doteq 1] \ \Rightarrow_E\ or1\ (\vee_{Il}\ p)$$

Here $\Rightarrow_E$ is the modus ponens rule, and $\vee_{Il}$ is the left introduction rule of disjunction of the meta-language. $\Rightarrow_E$ and $\vee_{Il}$ are among the proof rules declared in HOL, which we omitted in Sect. 4.3.

5. Representing Set-Theoretical Model Theory

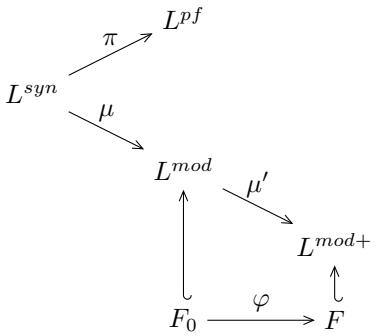
The representation of models given in Sect. 4.4 uses HOL as a meta-language. HOL is seen as a fragment of the foundation of mathematics, and to work with HOL rather than, e.g., a set theory, has the advantage of being simpler while not committing to a specific foundation. But it also has a drawback: FOL-models are represented as HOL-models and thus as platonic entities that live outside the logical framework LF.

It would be more appealing if FOL-models could be represented as LF entities themselves. This is indeed possible without changing the principal features of our approach: All we have to do is to refine the meta-language HOL so much that it becomes set theory. The refinements can be represented elegantly as LF signature morphisms – in this case from the encoding of HOL to an encoding of set theory.

More generally, we obtain the diagram below. Here $(L^{syn}, L^{pf}, \pi, L^{mod}, \mu)$ is a logic encoding as before. The foundation of mathematics is encoded as an LF signature F, and the model theory is defined in terms of a meta-language F_0, which is a fragment of F. A view $\varphi : F_0 \to F$ encodes the refinement of F_0 into F, or in other words, φ formalizes in what sense F_0 is a fragment of F.

Finally, we want to give a view μ' from L^{mod} to F, which translates the F_0-based encoding of model theory to F. μ' must have some free parameters, and this can be expressed in LF by adding these free parameters to the codomain of the view. L^{mod+} is the extension of F with these parameters.

Then, for any choice of these parameters, the composition $\mu\ \mu'$ translates the logical syntax into mathematics. In other words, we refine the logic encoding $(L^{syn}, L^{pf}, \pi, L^{mod}, \mu)$ based on F_0 to a logic encoding $(L^{syn}, L^{pf}, \pi, L^{mod+}, \mu\ \mu')$ based on F.



So far we have used first-order logic for L and higher-order logic for F_0. In the following, we will give F, μ', and φ. F will be an encoding of Zermelo-Fraenkel set theory (ZFC, [Zer08, Fra22]) encoded as an LF signature ZFC developed in Sect. 5.1. φ will give the standard semantics of hihger-order logic encoded as a view from HOL to ZFC developed in Sect. 5.2. Note that these steps are independent of the chosen logic L because higher-order logic is sufficient for the model theory of many logics (including sorted, simply-typed, modal, description, and intuitionistic logics). We will give L^{mod+} and the view μ' from FOL^{mod} to ZFC in Sect. 5.3. L^{mod+} will arise by adding to ZFC a free parameter for the universe.

We use ZFC because it is the most widely used foundation of mathematics. Other set theories such as Von-Neumann-Bernays-Gödel ([vN25, Ber37, Göd40]) could be used equally well. Similarly, type theoretic foundations such as HOL ([Chu40]) or the Calculus of Constructions ([CH88]) would work in the same way. We will elaborate on that in Sect. 5.2.

The above diagram still leaves open how individual models can be represented in LF. We will look at that in Sect. 5.4 where we will form a signature Σ^{mod+}, which will be like Σ^{mod} but in terms of ZFC rather than HOL, and then represent Σ-models as LF signature morphisms from Σ^{mod+} to ZFC. Finally we look at the encoding of model theoretical theory morphisms, at which point all aspects of FOL are encoded in LF. But since the Σ-models form a proper class and the LF expressions are countable, this raises adequacy questions, which we discuss in Sect. 5.5.

5.1. Representing Set Theory

Now we will represent ZFC set theory in LF. This is a necessary condition for the comprehensive representation of model theory in a logical framework. But it requires a significant investment. Very advanced encodings of set theory have been established in Mizar ([TB85] using Tarski-Grothendieck set theory [Try89, Tar38]), and in Isabelle/ZF ([Pau94, PC93]) employing sophisticated machine support. In particular, these encodings use semi-automated reasoning support and high-level proof description languages such as Isar ([Nip02]) for Isabelle. Our encoding was designed from scratch using hand-written proof terms.

There are two reasons to forgo those sophisticated encodings in favor of LF. Firstly, LF is superior to Isabelle as a logical framework: The dependent type theory permits elegant encodings of logics, and the module system based on signature morphisms permits elegant encodings of translations. We appreciate these fundamental aspects even though Isabelle is vastly superior to LF in terms of automation and tool support. Mizar offers dependent types, but it is a standalone encoding of set theory not based on a logical framework so that it cannot encode other languages such as logics or alternative set theories.

Secondly, our encoding of set theory differs from the above two in two fundamental but non-trivial design aspects. In both cases, only the existence of dependent types makes our design choices possible. These two aspects are the choice of primitive symbols and the use of a type system, which we will detail in Sect. 5.1.2 and 5.1.3, respectively.

The whole encoding of set theory comprises over 1000 lines of Twelf declarations. Therefore, we only showcase the most important features and refer to [KMR09] for the full encoding. In the following we will first explain the logic we use for our set theory in Sect. 5.1.1, then use it to develop untyped set theory in Sect. 5.1.2, and then build a typed set theory on top of the untyped one in Sect. 5.1.3. Only the typed set theory will be strong enough to subsume the meta-language HOL we developed in Sect. 4.3. Finally, we define the 2-element Boolean lattice $\mathbb{B}$ and its operations in Sect. 5.1.4. The Booleans are not needed to subsume HOL but are needed as the set of truth values when defining the semantics of FOL. All declarations together form the LF signature ZFC.

Notation 38. This section will require the reader to be very careful in separating levels as the LF encoding of ZFC contains three groups of connectives and quantifiers.

The first two groups are meta-level operations on propositions. They share the propositional connectives which are written normally, e.g., $\wedge$. The first group consists of the symbols used in the (untyped) first-order logic underlying set theory; here equality and quantifiers will be written as $\doteq^*$, $\forall^*$, and $\exists^*$. The second group consists of the symbols used in the typed set theory that we will develop on top of the untyped one; here equality and quantifiers will be written as $\doteq$, $\forall$, and $\exists$.

Finally the third group consists of the object level operations on Booleans. These will be written as $\wedge_*$, $\forall_*$, $\doteq_*$, etc.

The notations are chosen such that the symbols $\wedge$, $\forall$, $\doteq$, etc. are the intended interpretations of their counterparts in HOL, and the symbols $\wedge_*$, $\forall_*$, $\doteq_*$, etc. are the intended interpretations of the symbols declared in FOL^{mod}.

5.1.1. Logical Language

We base ZFC on first-order logic with equality. To reason about truth, we use an intuitionistic natural deduction calculus with introduction and elimination rules. The main LF declarations encoding this logic are the following ones.

$$\begin{array}{lcl} \mathit{set} & : & \texttt{type} \\ \mathit{prop} & : & \texttt{type} \\ \mathit{True} & : & \mathit{prop} \to \texttt{type} \end{array}$$

set is the single sort of sets, *prop* is the type of propositions, and $\mathit{True}\,F$ is the judgment for the truth of F.

We make two additions to the otherwise well-known syntax of first-order logic: sequential connectives and a description operator. Both arise naturally when encoding set theory as we will see below.

Sequential connectives mean that, e.g., in an implication $F \Rightarrow G$, G is only considered if F is true. This is very natural in mathematical practice – for example, mathematicians do not hesitate to write $x \neq 0 \Rightarrow x/x = 1$ when $/$ is only defined for non-zero dividers. This can be solved by using first-order logic with partial functions, but we hold that it is more elegant and closer to mathematics to use a sequential implication, i.e., the truth of F is assumed when considering G. Similarly, in a sequential conjunction $F \wedge G$, F is assumed true when considering G. We use sequential conjunction and implication; all other connectives are as usual.

Then the LF encoding contains the following declarations for propositions:

$$\begin{array}{lcl} \wedge & : & \{F : \mathit{prop}\}\,(\mathit{True}\,F \to \mathit{prop}) \to \mathit{prop} \\ \Rightarrow & : & \{F : \mathit{prop}\}\,(\mathit{True}\,F \to \mathit{prop}) \to \mathit{prop} \\ \neg & : & \mathit{prop} \to \mathit{prop}. \\ \vee & : & \mathit{prop} \to \mathit{prop} \to \mathit{prop} \\ \Leftrightarrow & : & \mathit{prop} \to \mathit{prop} \to \mathit{prop} \\ \doteq^* & : & \mathit{set} \to \mathit{set} \to \mathit{prop} \\ \forall^* & : & (\mathit{set} \to \mathit{prop}) \to \mathit{prop} \\ \exists^* & : & (\mathit{set} \to \mathit{prop}) \to \mathit{prop} \end{array}$$

Thus $\wedge$ and $\Rightarrow$ are applied to two arguments, a formula F and another formula which is stated in a context where F is true. This is written as, e.g., $F \wedge [p]\,G\,p$ where p is a proof of F that may be used by G. We will use $F \wedge G$ and $F \Rightarrow G$ as abbreviations when p does not occur in G; this yields the non-sequential variants of the connectives as special cases.

At this point it is not possible for G to actually make use of the truth of F because proofs cannot occur in formulas. This will change by the use of a description operator, and we will also use it when defining our typed set theory.

The proof rules for the sequential connectives are almost the same as for the usual ones. The only difference is that the proof of the first argument has to be supplied in a few places:

$$\begin{array}{lcl} \wedge I & : & \{p : True\ F\}\ \ True\ G\,p\ \ \rightarrow\ \ True\ F \wedge [p]\,G\,p \\ \wedge E_l & : & True\ F \wedge [p]\,G\,p\ \ \rightarrow\ \ True\ F \\ \wedge E_r & : & \{q : True\ F \wedge [p]\,G\,p\}\ \ True\ G\,(\wedge E_l\ q) \\ \Rightarrow I & : & (\{p : True\ F\}\ \ True\ G\,p)\ \ \rightarrow\ \ True\ F \Rightarrow [p]\,G\,p \\ \Rightarrow E & : & True\ F \Rightarrow [p]\,G\,p\ \ \rightarrow\ \ \{p : True\ F\}\ \ True\ G\,p \end{array}$$

Note that these rules contain the rules for the non-sequential connectives as special cases. We omit the well-known encoding of the introduction and elimination proof rules for the remaining connectives.

The *description operator* is a binder that takes a formula $F\,x$ with a free variable and returns the unique x satisfying $F\,x$. This is of course not well-formed for all F. Therefore, δ takes a dependent argument, which is a proof of $\exists^{*!}[x]\,F\,x$. Here $\exists^{*!}$ abbreviates the quantifier of unique existence. It can be encoded naturally using

$$\begin{array}{lcl} \exists^{*!} & : & (set \rightarrow prop) \rightarrow set\ =\ [F]\,\big(\exists^{*}[x]\,F\,x \wedge (\forall^{*}[y]\,F\,y \Rightarrow y \doteq^{*} x)\big). \\ \delta & : & \{F : set \rightarrow prop\}\ \ (True\ \exists^{*!}[x]\ F\,x)\ \ \rightarrow\ \ set \end{array}$$

Here dependent types permit us to require a proof of unique existence as an argument thus guaranteeing that only well-formed terms are formed. This is in contrast to the two description operators that are formalized in Isabelle/ZF or induced by the Mizar type system, respectively. Both are well-formed even for unsatisfiable formulas, in which case they return an arbitrary element. Thus, both Isabelle/ZF and Mizar assume not only the axiom of choice but also the existence of a global choice function, a commitment that we can avoid.

δ comes with an axiom scheme

$$ax_\delta \quad : \quad True\ F\ (\delta\ ([x]\,F\,x)\ P)$$

for an arbitrary proof P, which states that δ indeed yields the element with property F. Note that proof irrelevance is derivable from ax_δ, i.e., $(\delta\,F\,P)$ returns the same object no matter which proof P is used.

Note that both sequential connectives and the description operator crucially depend on the existence of dependent types in the logical framework.

5.1.2. Untyped Set Theory

Regarding the *primitive symbols*, our encoding attempts to stay as closely to mathematical practice as possible. We only use a single primitive non-logical symbol: the binary predicate $\in$: $set \rightarrow set \rightarrow prop$. This means that the only terms are the variables and those obtained from the description operator. Thus, all mathematical symbols besides $\in$ are introduced as abbreviations for sets whose (unique) existence has been proved.

Our encoding is in contrast to Mizar where primitive function symbols are used for singleton, unordered pair, and union ([Try89]) together with Tarski's

axiom of universes, and to Isabelle/ZF where primitive function symbols are used for empty set, powerset, union, infinite set, and replacement ([PC93]).

This permits us to follow the literature and encode all ZFC operations as existential axioms. For example, we can use

$$\forall^*[X]\,\exists^*[u]\,(\forall^*[y]\,(\exists[x]\,x \in X \wedge y \in x) \Rightarrow y \in u)$$

as the axiom of union. From this we can obtain a proof $P\ X$ of

$$\exists^{*!}[u]\,(\forall^*[y]\,(\exists[x]\,x \in X \wedge y \in x) \Leftrightarrow y \in u)$$

for an arbitrary set X. Then we can define the union operation as

$$\bigcup : set \to set = [X]\ \big(\delta\ \ ([u]\ \ \forall^*[y]\,(\exists[x]\,x \in X \wedge y \in x) \Leftrightarrow y \in u)\ \ (P\ X)\big)$$

Similarly, we proceed to define the empty set, unordered pairs, and powersets using the respective axiom, as well as encodings of the sets $\{x \in A | F(x)\}$ and $\{f(x) : x \in A\}$ using the axiom schemes of specification and replacement, respectively. Furthermore, we use δ and the axiom of infinity to obtain a specific infinite set, in our case the set of natural numbers. For all results described in this paper, we do not use δ or any of these seven axioms anywhere else, i.e., all other sets and operations on sets are defined in terms of these seven applications of δ.

Our results do not actually require the axioms of choice and excluded middle. Similarly, regularity and infinity are not used for the results presented in this section. However, these axioms may be needed to construct specific models such as models with an infinite domain.

We define ordered pairs (x, y) as $\{\{x\}, \{\{y\}, \varnothing\}\}$. Our definition is unusual and similar to Wiener's $\{\{\{x\}, \varnothing\}, \{\{y\}\}\}$ ([Wie67]). We do not use the common Kuratowski pairs $\{\{x\}, \{x, y\}\}$ because reasoning about them often requires the principle of excluded middle to distinguish the cases $x \doteq^* y$ and $x \not\doteq^* y$, which we try to avoid unless necessary. However, we immediately define the projections π_1 and π_2 and prove the conversions $\pi_i(x_1, x_2) \doteq^* x_i$ and $isPair\ u \Rightarrow (\pi_1(u), \pi_2(u)) \doteq^* u$, which are known from simple type theory. Afterwards all work is carried out in terms of these derived operations and properties so that the specific definition of pairing becomes less relevant. (The LF module system can check that only the derived operations are used.)

Based on ordered pairs, we can define relations and functions in the usual way. In particular, we define set theoretical notions of λ abstraction and application as operations $\lambda^*\,A\,([x : set]\,f\,x)$ encoding $\{(x, f(x)) : x \in A\}$ and $f\ @^*\ a$ encoding "the b such that $(a, b) \in f$". Again we immediately prove the conversions of β- and η-equality and use only those later on.

In a similar way, we define various other notions such as subset, singleton set, binary union, intersection, difference, disjoint union, etc., and derive natural deduction rules for them.

5.1.3. Typed Set Theory

We have already remarked that a major problem with formalizations of set theory is their complexity. Type theories favor algorithmic definitions and decidable notions, and these prove indispensable when formalizing major parts of mathematics in a computer.

It is not surprising that most of the biggest successes of formalized mathematics such as the formalization of the Four Color Theorem and the Kepler Conjecture ([Gon06, Hal03]) are achieved in type theory-based formalizations of mathematics – Coq ([BC04]) and HOL Light ([Har96]), respectively. Similarly, while Mizar is untyped a priori, it supports a very sophisticated type system as a derived notion that is internally represented using predicates over sets ([Wie07]). Isabelle/ZF offers much weaker support for typed reasoning, and this is one of the main reasons why both tool support and available content are farther developed in Isabelle/HOL ([NPW02]) using a typed foundation rather than in Isabelle/ZF.

In LF, again using the dependent typing, we can derive typed set theory in a rather simple way. The crucial idea is to use the dependent sum type $elem\ A := \Sigma_{x:set}(True\ \ x \in A)$ to represent the set A. Thus, elements x of A are represented as pairs (x, P) where P is a proof that x is indeed in A. If we also require proof irrelevance, i.e., $(x, P) = (x, P')$, then the type $elem\ A$ has exactly one term for every element of A. This is inspired by the Scunak language ([Bro06]), which uses this representation as a primitive notion and provides implementation support for it.

It is a minor inconvenience that LF (unlike Scunak) supports neither dependent sum types nor the ability to make all elements of a type definitionally equal (which would permit to state the proof irrelevance). Therefore, we have to add *elem* and its properties as primitives to our LF encoding as shown below

$$\begin{array}{lcl} elem & : & set \to \texttt{type} \\ el & : & \{x : set\}\ True\ \ x \in A \to elem\, A \\ which & : & elem\ \ A \to set \\ why & : & \{a : elem\ \ A\}\ True\ (which\ \ a) \in A \end{array}$$

along with an axiom for proof irrelevance and one for $(which\,(el\ x\,P)) \doteq^* x$. *el*, *which*, and *why* would simply be pairing and the two projections if LF had sum types. While this increases the needed primitives, these declarations only emulate features that could easily be added to the LF language: Dependent sum types are used in, e.g., [ML74] and [Nor05]; proof irrelevance is added in [LP09].

Using the types $elem\ A$, we can now lift all the basic untyped operations introduced above to the typed level. In particular, we define typed quantifiers $\forall$, $\exists$, typed equality $\doteq$, and typed function spaces $\Longrightarrow$ in the following.

Firstly, we define *typed quantifiers* such as $\forall : (elem\ A \to prop) \to prop$. In higher-order logic with internal propositions ([Chu40], compare $prop : set$ rather than $prop : \texttt{type}$ in Sect. 4.3), such typed quantification can be defined easily. In untyped set theory, this is intuitively possible using relativization,

e.g., $\forall F := \forall^*[x]\, x \in A \Rightarrow F\, x$ for $F : elem\ A \to prop$.

However, an attempt to formally define typed quantification like this meets a subtle difficulty: In $\forall F$, F only needs to be defined for elements of A whereas in $\forall^*[x]\, x \in A \Rightarrow F\, x$, F must be defined for all sets. Thus, $\forall$ is more general than $\forall^*$ in that it permits a weaker argument. Of course, in $\forall^*[x]\, x \in A \Rightarrow F\, x$, it is intended not to consider $F\, x$ if $x \notin A$. This is the motivation behind the introduction of sequential connectives in Sect. 5.1.1 above.

Using sequential connectives, we can define $\forall$ and $\exists$ as follows:

$$\begin{array}{lcl} \forall & : & (elem\ A \to prop) \to prop\ =\ [F]\ \big(\forall^*[x]\, x \in A \Rightarrow [p]\, (F\ (el\ x\, p))\big) \\ \exists & : & (elem\ A \to prop) \to prop\ =\ [F]\ \big(\exists^*[x]\, x \in A \wedge [p]\, (F\ (el\ x\, p))\big) \end{array}$$

Then we can derive introduction and elimination rules for $\forall$ and $\exists$, which look the same as those for the untyped ones.

Secondly, *typed equality* is easy to define:

$$\doteq \quad : \quad elem\ A \to elem\ A \to prop\ =\ [a]\, [b]\, (which\ a) \doteq^* (which\ b)$$

It is easy to see that all rules for $\doteq^*$ can be lifted to $\doteq$.

Finally, we can define *function types* that are defined in terms of untyped functions:

$$\begin{array}{lcl} \Longrightarrow & : & set \to set \to set\ = \ldots \\ \lambda & : & (elem\ A \to elem\ B) \to elem\, (A \Longrightarrow B)\ = \ldots \\ @ & : & elem\, (A \Longrightarrow B) \to elem\ A \to elem\ B\ = \ldots \\ beta & : & True\ \ ((@\, (\lambda\, [x]\, F\, x)\, A) \doteq^* F\, A)\ = \ldots \\ eta & : & True\ \ ((\lambda\, [x]\, (@\, F\, x)) \doteq^* F)\ = \ldots \end{array}$$

We omit the quite involved definitions and only mention that the typed quantifiers and thus the sequential connectives are needed in the definitions.

5.1.4. The Booleans

The Booleans $\mathbb{B}$ are easy to define as the set containing the two elements $0 = \varnothing$ and $1 = \{\varnothing\}$. However, it is interesting to note that there are two different ways to define this set: We can use the unordered pair $\{0, 1\}$ or the powerset $\mathcal{P}(1)$.

Clearly, $\{0, 1\}$ is a two-element set and $\{0, 1\} \subseteq \mathcal{P}(1)$, but it turns out that the two sets are only equal in the presence of the axiom of excluded middle. In fact, – maybe surprisingly – by using the set $\{x \in 1 \mid F\}$, it can be shown that $\{0, 1\} \doteq^* \mathcal{P}(1)$ is equivalent to $F \vee \neg F$ for all formulas F.

Therefore, an intuitionistic set theory would have to define $\mathbb{B} = \{0, 1\}$, and it presents no fundamental obstacles to do so. But it is more convenient to use $\mathbb{B} = \mathcal{P}(1)$ because then all operations on the Booleans can be obtained from the lattice operations in $\mathcal{P}(1)$. Therefore, we put $\mathbb{B} = \mathcal{P}(1)$.

Then most of the operations on the Booleans are straightforward:

$$
\begin{array}{lcl}
\neg_* & : & elem\,\mathbb{B} \Longrightarrow \mathbb{B} \;=\; \lambda[x]\,1 \setminus x \\
\wedge_* & : & elem\,\mathbb{B} \Longrightarrow \mathbb{B} \Longrightarrow \mathbb{B} \;=\; \lambda[x]\,\lambda[y]\,x \cap y \\
\vee_* & : & elem\,\mathbb{B} \Longrightarrow \mathbb{B} \Longrightarrow \mathbb{B} \;=\; \lambda[x]\,\lambda[y]\,x \cup y \\
\Rightarrow_* & : & elem\,\mathbb{B} \Longrightarrow \mathbb{B} \Longrightarrow \mathbb{B} \;=\; \lambda[x]\,\lambda[y]\,\mathit{reflect}\ (x \subseteq y) \\
\forall_* & : & elem\ (A \Longrightarrow \mathbb{B}) \Longrightarrow \mathbb{B} \;=\; \lambda[f]\,\bigcap(\mathit{image}\ f) \\
\exists_* & : & elem\ (A \Longrightarrow \mathbb{B}) \Longrightarrow \mathbb{B} \;=\; \lambda[f]\,\bigcup(\mathit{image}\ f)
\end{array}
$$

where $\setminus$ returns the difference of two sets, *image* f is the image of the function f, *reflect* F encodes the set $\{x \in 1 | F\}$, and $\bigcap$ and $\bigcup$ return the union or intersection, respectively, of a set of sets. We omit their definitions.

In the usual way, we can prove the basic properties of the lattice operations, from which we can prove the intended properties of the Boolean operations.

Finally, we use the axiom of excluded middle once to prove that $\mathbb{B}$ is equal to $\{0, 1\}$.

5.2. Viewing Higher-Order Logic in Set Theory

Now that we have developed an encoding of set theory, we want to show that it subsumes the meta-language HOL used in Sect. 4 to represent model theory. Let ZFC be the LF signature containing our set theory. Then the subsumption can be expressed in LF formally as a view from HOL to ZFC.

This basic idea of the view is straightforward because all constants of HOL are mapped to ZFC-constants of the same name, i.e., we have a view

$$
\begin{array}{l}
\%\texttt{view}\ \varphi\ : HOL\ \to ZFC\ = \{ \\
\quad \begin{array}{lcl}
set & := & set \\
elem & := & elem \\
prop & := & prop \\
True & := & True \\
\vdots & &
\end{array} \\
\}
\end{array}
$$

The view must also map all proof rules of HOL to proofs of the corresponding ZFC theorems. For the propositional connectives, those are the same rules assumed for the first-order logic underlying ZFC. For the quantifiers and equality, they are derived rules for $\forall$, $\exists$, and $\doteq$. For β- and η-equality, they are the corresponding derived rules of ZFC.

Instead of ZFC set theory, we could use any other foundation into which we can give such a view φ. This includes other set theories but also typed foundations such as the usual higher-order logic with internal propositions. For example, the latter arises if we use the type theory from Sect. 2.2 but with a declaration $prop : tp$; in that case the view would map $prop$ to $tm\ prop$.

5.3. Viewing Model Theory in Set Theory

Next we should define a view μ' from FOL^{mod} to ZFC. However, it is not possible to fix the value of $\mu'(univ)$ because it may be different in every model. Thus, μ' must be parametric in the choice of $\mu'(univ)$ and consequently also in

that of $\mu'(nonemp)$. We solve that by introducing FOL^{mod+} as ZFC with two free parameters as below

$$
\begin{array}{l}
\%\texttt{sig}\ FOL^{mod+} = \{ \\
\quad \%\texttt{include}\ ZFC \\
\quad U \ : \ set \\
\quad P \ : \ True\ \exists^*[x]\, x \in U \\
\}
\end{array}
$$

and giving a view $\mu' : FOL^{mod} \to FOL^{mod+}$ instead.

This view is again modular to interpret every connective separately. Moreover, μ' includes (and thus reuses) φ so that the LF module system guarantees that the diagram on the right commutes. We will not present the modular structure here, but rather give examples of the most interesting cases.

$$
\begin{array}{ccc}
FOL^{mod} & \xrightarrow{\mu'} & FOL^{mod+} \\
\uparrow & & \uparrow \\
HOL & \xrightarrow{\varphi} & ZFC
\end{array}
$$

μ' interprets the booleans as the two-element set of truth values, and maps *univ* and *nonemp* to the free parameters U and P as below.

$$
\begin{array}{lcl}
bool & := & \mathbb{B} \\
0 & := & 0 \\
1 & := & 1 \\
\vdots & & \\
univ & := & U \\
nonemp & := & P
\end{array}
$$

All connectives and quantifiers are mapped to their counterparts on $\mathbb{B}$, e.g., $\wedge$ is mapped to $\wedge_*$ and $\forall$ to $\forall_*$. The proofs of the axioms are simple.

5.4. Representing Model Theory

Models. Assume a logic encoding $(L^{syn}, L^{pf}, \pi, L^{mod+}, \mu\ \mu')$ using a foundation F as before. The basic idea behind the encoding of L-models M of Σ is given by the diagram below. Here Σ^{mod+} arises in the same way as Σ^{mod}, i.e., by pushout of Σ^{syn} along $\mu\ \mu'$ over L^{syn}, or equivalently by pushout of Σ^{mod} along μ' over L^{mod}. Then the encoding $\ulcorner M \urcorner$ of M is a morphism from Σ^{mod+} to F such that following diagram commutes:

$$
\begin{array}{ccc}
L^{syn} & \hookrightarrow & \Sigma^{syn} \\
 & {\scriptstyle \mu\ \mu'} \searrow \quad \swarrow {\scriptstyle \mu_\Sigma\ \mu'_\Sigma} & \\
 & L^{mod+} \hookrightarrow \Sigma^{mod+} & \\
 & \uparrow \qquad \downarrow {\scriptstyle \ulcorner M \urcorner} & \\
 & F \xrightarrow{id_F} F &
\end{array}
$$

Notation 39. The notation $\ulcorner M \urcorner$ was already used in Sect. 4 for the encoding of M in HOL. We will reuse it in Sect. 5 for the encoding of a model M in LF.

This encoding captures and formalizes two of the most central intuitions about the syntax and semantics of formal languages. Firstly, the semantics of a formal language is a structure-preserving translation of the syntax into some semantic realm. For logics the semantic realm is usually mathematics, in our case encoded by a foundation F. The interpretation function is given by $\mu_\Sigma\ \mu'_\Sigma\ \ulcorner M \urcorner$. Secondly, the syntax consists of two parts: logical and non-logical symbols. In our case, the semantics of the logical symbols is given by a fixed morphism $\mu\ \mu'$, and the semantics of the non-logical symbols is given by the morphism $\ulcorner M \urcorner$.

$\ulcorner M \urcorner$ must map all symbols of Σ^{mod+}, which can be split into three groups. Firstly, symbols included from F have a fixed meaning in the foundation; the commutativity ensures that $\ulcorner M \urcorner$ is the identity on them. Secondly, symbols included from L^{mod+} encode fixed parts of the models that do not depend on the signature; in the case of FOL, this is the universe encoded using the symbols U and P. Thirdly, the symbols inherited from Σ^{syn} when constructing the pushout are the non-logical symbols.

We can formalize the above intuitions as follows:

Definition 40 (Encoding Models)**.** An *LF-based model* of a FOL-signature or theory Σ is a morphism $I : \Sigma^{mod+} \to ZFC$ that is a retraction of the inclusion $ZFC \hookrightarrow \Sigma^{mod+}$.

Note that this definition includes the case when Σ is a theory. In that case, LF-based models map the axioms in Σ^{mod+} (which stem from the pushout of Σ^{syn}) to proof terms in ZFC.

Definition 41 (Encoding Semantics)**.** Assume an LF-based model I of a FOL-signature Σ and a term or formula E over Σ. Then the LF-based semantics of E is given by $(\mu_\Sigma\ \mu'_\Sigma\ I)(\ulcorner E \urcorner)$, which we also write as $[\![E]\!]^I$.

Due to the type preservation of LF signature morphism, the LF-based semantics $[\![t]\!]^I$ of a term t is indeed an (encoding of an) element of the universe and the LF-based semantics $[\![F]\!]^I$ of a formula is an (encoding of a) truth value.

We have a very strict notion of identity between LF-based models inherited from LF signature morphisms: Two signature morphisms are equal if they agree for all arguments up to $\beta\eta$-equality. For the representation of models, we need a more relaxed notion based on whether equality can be proved in ZFC:

Definition 42 (Equality of Models)**.** Two LF-based model I_1 and I_2 of a FOL-signature or theory Σ are *provably equal* if $True\ I_1(U) \doteq^* I_2(U)$ and all types $True\ I_1(s) \doteq I_2(s)$ for all function or predicate symbols s of Σ are inhabited.

Note that if Σ is a theory, we do not require the equality of $I_1(a)$ and $I_2(a)$ for axioms a, i.e., the proofs of the axioms are irrelevant (as long as they exist). Similarly, we do not require any equality of $I_1(P)$ and $I_2(P)$.

Model Reduction. As before, a FOL-signature morphism $\sigma : \Sigma \to \Sigma'$ is encoded as an LF signature morphism $\sigma^{syn} : \Sigma^{syn} \to \Sigma'^{syn}$, and pushout along $\mu\,\mu'$ yields

an LF signature morphism $\sigma^{mod+} : \Sigma^{mod+} \to \Sigma'^{mod+}$. Then our representation of models as morphisms permits a very elegant representation of model reduction by composition with σ^{mod+}.

Theorem 43 (Encoding Model Reduction)**.** *Assume a FOL signature morphism $\sigma : \Sigma \to \Sigma'$. If I is an LF-based model of Σ', then $\sigma^{mod+}\, I$ is an LF-based model of Σ. Moreover, for a term or formula E over Σ, we have*

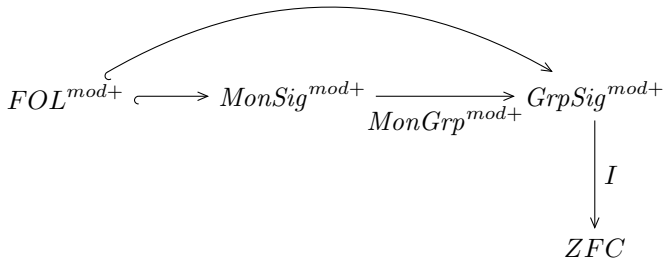
$$[\![E]\!]^{\sigma^{mod+}\, I} = [\![\sigma(E)]\!]^I.$$

Proof. Both claims follow immediately from the properties of signature morphisms and the construction of σ^{mod+} by pushout. □

Readers familiar with institutions will recognize the second claim as the satisfaction condition.

Example 44 (Continued). Consider the model Int from Ex. 15. It can be represented as an LF-based model $I : GrpSig^{mod+} \to ZFC$. $I(U)$ is the ZFC-term for the set of integers, which is straightforward to define using the natural numbers. $I(P)$ is some proof over ZFC that proves U is non-empty. Then we only have to define I for the symbols $\circ$, e, and inv, which are mapped to ZFC-expressions representing the operations $+$, 0, and inv on the integers.

Via pushout, the FOL-signature morphism $MonGrp : MonSig \to GrpSig$ from Ex. 5 gives rise to an LF signature morphism $MonGrp^{mod+} : MonSig^{mod+} \to GrpSig^{mod+}$. The encoding of the model reduct $Mod(MonGrp)(Int)$ is obtained as the composition $MonGrp^{mod+}\, I$ as in the following diagram:



$MonGrp^{mod}\, I$ agrees with I but is not defined for inv.

Alternatively, we can encode Int as a model of the theory $Group$. That model must map from $Group^{mod+}$ to ZFC, i.e., additionally maps the axioms of $Group$ to ZFC. These are mapped to proofs that the integers do indeed satisfy the axioms of a group.

5.5. Adequacy

The encoding of models as signature morphisms raises a difficult question based on a simple cardinality argument: There is a proper class of FOL-models of Σ, but only countably many signature morphisms to encode them. Therefore, we have to look carefully at the adequacy of our encoding.

First of all we have:

Lemma 45. *Our encoding of set theory is adequate in the following sense:*

- *Every closed term $t : set$ induces a set $\llcorner t \lrcorner$.*
- *Two closed terms s, t induce the same set if the type $True\ s \doteq^* t$ is inhabited.*
- *For every closed term $t : elem\ A$, the set $\llcorner which\ t \lrcorner$ is an element of $\llcorner A \lrcorner$.*

Proof. All claims rely on the assumption that every proof rule of the underlying first-order logic and every axiom in ZFC is sound with respect to the platonic universe of set theory. For researchers objecting to parts of the encoding (e.g., to excluded middle), the corresponding result holds after modifying ZFC.

For the first claim, we expand all definitions in t. This yields either a term of the form $which\ (el\ t'\ P)$, which is provably equal to the smaller term t' so that we can recurse, or a term of the form $\delta\ F\ Q$. Thus, for every term $t : set$, we can obtain a provable formula $\exists^{*!}[x]\, F\ x \wedge F\ t$. This shows the existence of a set that t represents. The second claim holds because the existence of a term inhabiting $True\ s \doteq^* t$ shows that $s \doteq^* t$ is a provable formula. The third claim follows easily from the first one. □

Then we have:

Theorem 46. *Assume a FOL-signature Σ. Every LF-based model I induces a FOL-model $\llcorner I \lrcorner$.*

Proof. The universe of $\llcorner I \lrcorner$ is the set $\llcorner I(U) \lrcorner$. For Σ-symbols s, the interpretation of s in $\llcorner I \lrcorner$ is the object $\llcorner which\ I(s) \lrcorner$. □

Whether or not all sets and models are induced by LF-terms and LF-based models is a philosophical question. If we adopt a formalist or even a constructivist point of view, then the LF-terms are (representatives of) all the sets and thus the LF-based models are all the models. If we adopt a platonic point of view, only some models can be encoded, but these include – intuitively – all models whose components can be written down or named. Furthermore, we can always create variations of our signature ZFC to accommodate other perspectives. For example, we can add a choice operator to represent models obtained by applying the axiom of choice.

Then we have the following adequacy results for those models that can be represented:

Definition 47. A FOL-model M is *definable* if $M = \llcorner I \lrcorner$ for some LF-based model I. In that case we also write $I = \ulcorner M \urcorner$.

Theorem 48 (Adequacy for Model Theory)**.** *For every definable FOL-model M of Σ, and every term or formula E over Σ:*

$$[\![E]\!]^M = \llcorner [\![E]\!]^{\ulcorner M \urcorner} \lrcorner$$

Proof. This is straightforward from the definitions. The only subtlety is to show that $\llcorner[\![E]\!]^{\ulcorner M \urcorner}\lrcorner$ does not depend on which LF-based model is chosen for $\ulcorner M \urcorner$. But that is the case because any two possible choices must be provably equal. □

Theorem 49 (Adequacy for Model Reduction)**.** *For every FOL-signature morphism $\sigma : \Sigma \to \Sigma'$ and every definable FOL-model M' of Σ':*

$$Mod(\sigma)(M') = \llcorner \sigma^{mod+} \ulcorner M' \urcorner \lrcorner$$

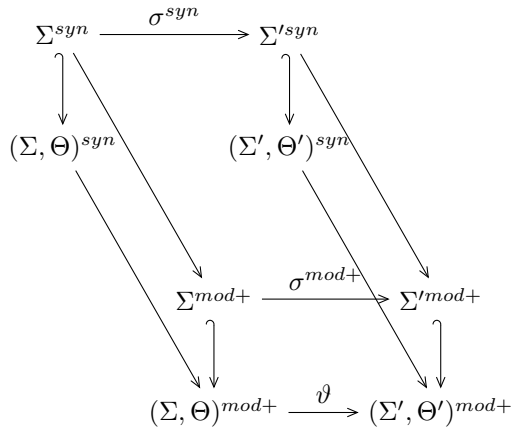
In particular, reducts of definable models are definable.

Proof. This is straightforward from the definitions. □

Finally, even though we may not be able to encode all models, we can adequately encode the property of being a model theoretical theory morphism:

Theorem 50 (Adequacy for Model Theoretical Theory Morphisms)**.** *For a FOL-signature morphism $\sigma : \Sigma \to \Sigma'$, we have $\sigma : (\Sigma, \Theta) \xrightarrow{\texttt{M}} (\Sigma', \Theta')$ iff $\sigma^{mod+} : \Sigma^{mod+} \to \Sigma'^{mod+}$ can be extended to an LF signature morphism $\vartheta : (\Sigma, \Theta)^{mod+} \to (\Sigma', \Theta')^{mod+}$.*

Proof. Consider the following commutative diagram, where as before $\sigma^{syn} : \Sigma^{syn} \to \Sigma'^{syn}$ is the encoding of σ and all $^{mod+}$-nodes arise as pushouts of the corresponding ^{syn}-node along $\mu\,\mu' : L^{syn} \to L^{mod+}$:



The claim is that σ is a model theoretical theory morphism iff such a ϑ exists.

The right-to-left direction is easy: ϑ contains proof terms that show that the reduct of a (Σ', Θ')-model satisfies all the axioms of Θ.

To show the converse direction, recall that we only consider finite theories, and observe that given Σ, Θ, Σ', Θ', and σ, we can write the following formula in the first-order language of ZFC.

$$\begin{aligned} f \;=\; \forall U, s_1, \ldots, s_n.(M'(U, s_1, \ldots, s_n) \wedge \textstyle\bigwedge_i A'_i(U, s_1, \ldots, s_n)) \\ \Rightarrow (M(U, \sigma_1, \ldots, \sigma_m) \wedge \textstyle\bigwedge_i A_i(U, \sigma_1, \ldots, \sigma_m)) \end{aligned}$$

Here m is the number of function and predicate symbols declared in Σ; Σ' declares function and predicate symbols named $s_1, \ldots, s_n$; $M(x, y_1, \ldots, y_m)$ expresses that $(x, y_1, \ldots, y_m)$ is a Σ-model with universe x; $A_i(x, y_1, \ldots, y_m)$ expresses that said Σ-model satisfies the i-th axiom in Θ; $M'(x, y_1, \ldots, y_n)$ and $A'_i(x, y_1, \ldots, y_n)$ are defined accordingly for Σ' and Θ'; and $\sigma_i \in \{s_1, \ldots, s_n\}$ is the result of applying σ to s_i.

Then f is a theorem over ZFC iff σ is a model-theoretical theory morphism. So assume a proof of f, from which we obtain a corresponding LF proof term p over the signature ZFC.

Moreover, over the signature Σ'^{mod+}, which extends ZFC, we have a proof term q proving $M'(U, s_1, \ldots, s_n) \wedge \bigwedge_i A'_i(U, s_1, \ldots, s_n)$. From p and q, we obtain a Σ'^{mod+}-proof term r_i proving $A_i(U, \sigma_1, \ldots, \sigma_n)$ for the i-th axiom in Θ. By putting $\vartheta(a_i) = r_i$, we obtain the needed LF signature morphism. □

The above proof rests on the philosophical assumption that a statement about ZFC – in this case, the statement $\sigma : (\Sigma, \Theta) \xrightarrow{\mathtt{M}} (\Sigma', \Theta')$ – can only be true if there is a proof of it in the first-order language of ZFC. Moreover, we assume that this first-order language is indeed the one that we encoded in ZFC. Researchers working with a different variant of set theory can apply Thm. 50 accordingly after modifying ZFC.

Finally, observe that the proof depends on the ability to switch between the internal representation of models – the tuples $(U, s_1, \ldots, s_n)$, which can be encoded as LF terms over ZFC – and the external representation of models as LF signature morphisms into ZFC.

6. Related Work

There are formalizations of the semantics of formal languages in various frameworks. For example, in [BKV09], a simple functional programming language is formalized in Coq ([BC04]). In [MNvOS99], a simple while-language is formalized in Isabelle/HOL ([NPW02]). Both use domain-theoretical models formalized based on posets where we use set theoretical models. In [CD97], simple type theory is formalized in ALF ([MN94]) using the normalization-by-evaluation method. They use a glued model in which interpretations are paired with normal forms. A similar result was obtained in [Coq02] using Kripke models.

Although the settings of these results are very different from each other's and from ours, all approaches are quite similar in that they define syntax and models and give an interpretation function satisfying a soundness property. A novelty of our approach is to base the models on an explicitly formalized foundation: We formalize set theory and use sets as the universes of the models. In the cited formalizations, on the other hand, the universes are types of the framework's type theory, which thus acts as an implicit foundation. Our approach makes the foundation flexible and avoids such an implicit commitment. For example, we could represent the cited formalizations in LF using an LF signature for Coq, Isabelle/HOL, or ALF, respectively, instead of the one for set theory.

Another difference is that the cited approaches all represent the interpretation function as a function of the framework's type theory. As this is impossible in LF, we use LF signature morphisms, which are less flexible but provide an elegant characterization of sound interpretation functions.

Dually, there are other formalizations of the semantics of formal languages in Twelf. In [App01], a formalization of HOL in Twelf is used to define the semantics of a machine language for proof-carrying code. This work does not focus on the separation of syntax, models, and interpretation. Instead, all notions are introduced as definitional extensions of HOL. From our perspective, they use HOL as the foundation and define the models by extending HOL whereas syntax and interpretation function are left implicit. In [LCH07], an SML-equivalent language is given, and state-transition systems are used to formalize the evaluation of expressions. These corresponds to our models, but they are not strictly separated from the syntax as in our case. Like our views, the interpretation function and soundness proof live on the meta-level: They are given by a number of logic programs formalized in the Twelf meta-theory. Despite their formidable sizes, both these Twelf developments are monolithic because they predate the module system.

Regarding our specific encodings, the encodings of first-order syntax and proof theory are straightforward and well-known (see, e.g., [HHP93]). Only the systematic use of modularity is novel. Our encoding of HOL is well-known, too, but note that there are two flavors of HOL, both based on simple type theory. The most common one based on [Chu40] treats propositions as terms of a special type *prop*. In LF, that would correspond to the declaration $prop : set$. This flavor is used in, e.g., [HST94, App01, Har96, NPW02]. The advantage is that the connectives and quantifiers can be introduced as HOL-terms. Our variant with $prop : \texttt{type}$ treats propositions as external to the type theory, i.e., propositions are not HOL-terms. This is more general and necessary to treat HOL as a fragment of first-order set theory where propositions and sets are strictly separated.

Our encoding of set theory is novel both in general and in the context of Twelf. The most advanced other formalizations of set theories are the ones in Mizar ([TB85]) and Isabelle/ZF ([PC93]). Scunak ([Bro06]) is a recent system developed specifically to exploit dependent type theory when encoding set theory. We discussed the differences between these and our encoding in Sect. 5.1. Other ways to encode set theory in dependently-typed frameworks use the framework's type theory as the foundation of mathematics, as in [Acz78]. Such encodings can be mechanized in systems like Agda and Coq, see e.g., [Gri09].

The main advantage of these other systems over LF/Twelf is that they provide a stronger notion of definitional equality and (semi-)automated proof support. For example, Isabelle, Agda, and Coq permit the declaration of recursive functions that are evaluated automatically by the framework. Scunak implements the proof irrelevance we axiomatize in Sect. 5.1.3. All of them are connected to automated or semi-automated proof tools or provide tactic languages. LF, on the other hand, is ontologically much simpler, even minimalistic. Con-

sequently, proof terms are fully explicit and more complicated to construct by hand. But LF (as well as Scunak) can benefit from the use of higher-order abstract syntax, which simplifies the reasoning about adequacy relative to traditional mathematics.

Finally, the idea of encoding models as morphisms goes back to Lawvere's work on functorial models ([Law63]) and the work on initial algebra semantics, e.g., in [GTW78]. While these have been developed in logical frameworks on paper before, e.g., in [MTP97] and [GMdP+07], our work marks the first time that they can be formalized and machine-checked in a logical framework.

7. Conclusion

We have given a comprehensive representation of first-order logic in a logical framework. Contrary to previous work, our representation covers both the proof and the model theoretical semantics given as provability and satisfaction, respectively. For example, the framework of institutions has been applied to the model theoretical semantics ([GB92]), and the framework LF to the proof theoretical semantics ([HHP93]), but a comprehensive representation has so far been lacking. This was due to the large ontological and philosophical differences between these two views on logic.

These differences are so big that we needed three major preliminary efforts to make this representation possible. In [Rab08, Rab10], we conceived the logical framework combining model and proof theory that we have built upon here. In [RS09], we gave the LF and Twelf module system that we used to implement the representation. In fact, our work is the largest case study in the Twelf module system to date. And finally, we needed a representation of a foundation of mathematics, which we have described in Sect. 5.1. These combine to a strong and flexible framework whose potential is exemplified by the representation of FOL we have given.

Our work will leverage future representations in two ways. Firstly, framework design and implementation are in place now, and this paper provides a detailed template how to represent logics. In fact, we have started this already ([KMR09]), and we expect further successes fast. Secondly, the Twelf module system permits the reuse of existing representation fragments. Our representation has separated all language features into independent and reusable components so that further logics can be represented by only adding individual language features such as sorted quantification or simple function types. Moreover, the meta-language for the model theory and its interpretation in ZFC set theory are composed modularly as well. Therefore, they cannot only be reused for many other logics but can also be refined flexibly if a more expressive meta-language or a different foundation of mathematics are needed. For example, we could easily extend the meta-language from HOL to a dependently-typed DHOL for a particular logic representation.

An important application of logical frameworks is to use the logic representations to reason about the represented logic. To that effect, we gave adequacy

results for syntax, proof theory, and model theory, and for the respective translations along morphisms. We gave a criterion to prove the soundness of a logic within the framework, and we used this to give a fully machine-verified soundness proof of first-order logic. A similar treatment of completeness remains future work.

Finally our work is part of a larger effort to obtain an atlas of logics and translations between them. Our work explains and exemplifies how logics, foundation, and models should be represented. A similar case study for logic translations is given recently in [Soj10]. Within the LATIN project ([KMR09]), our results will be integrated with the heterogeneous specification tool Hets ([MML07]) and the scalable Web infrastructure based on the markup language OMDoc ([Koh06]).

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

[Acz78] P. Aczel. The Type Theoretic Interpretation of Constructive Set Theory. In A. Macintyre, L. Pacholski, and J. Paris, editors, *Logic Colloquium '77*, pages 55–66. North-Holland, 1978.

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

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

[App01] A. Appel. Foundational Proof-Carrying Code. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 247–258. IEEE, 2001.

[Bar92] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.

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

[Ber37] P. Bernays, 1937. Seven papers between 1937 and 1954 in the Journal of Symbolic Logic.

[BKV09] N. Benton, A. Kennedy, and C. Varming. Some Domain Theory and Denotational Semantics in Coq. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2009.

[Bro06] C. Brown. Combining Type Theory and Untyped Set Theory. In U. Furbach and N. Shankar, editors, *International Joint Conference on Automated Reasoning*, pages 205–219. Springer, 2006.

[CD97] T. Coquand and P. Dybjer. Intuitionistic model constructions and normalization proofs. *Mathematical Structures in Computer Science*, 7(1):75–94, 1997.

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

[CH88] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, 1988.

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

[Coq02] C. Coquand. A Formalised Proof of the Soundness and Completeness of a Simply Typed Lambda-Calculus with Explicit Substitutions. *Higher-Order and Symbolic Computation*, 15(1):57–90, 2002.

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

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

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

[Fra22] A. Fraenkel. The notion of 'definite' and the independence of the axiom of choice. 1922.

[GB86] J. Goguen and R. Burstall. A study in the foundations of programming methodology: specifications, institutions, charters and parchments. In D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Workshop on Category Theory and Computer Programming*, pages 313–333. Springer, 1986.

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

[GMdP+07] J. Goguen, T. Mossakowski, V. de Paiva, F. Rabe, and L. Schröder. An Institutional View on Categorical Logic. *International Journal of Software and Informatics*, 1(1):129–152, 2007.

[Göd40] K. Gödel. The Consistency of Continuum Hypothesis. *Annals of Mathematics Studies*, 3:33–101, 1940.

[Gon06] G. Gonthier. A computer-checked proof of the four colour theorem, 2006. `http://research.microsoft.com/~gonthier/`.

[Gor88] M. Gordon. HOL: A Proof Generating System for Higher-Order Logic. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer-Academic Publishers, 1988.

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

[Gri09] J. Grimm. Implementation of Bourbaki's Elements of Mathematics in Coq: Part One, Theory of Sets. Technical Report HAL:inria-00408143, INRIA, 2009.

[GTW78] J. Goguen, J. Thatcher, and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology*, volume 4, pages 80–149. Prentice Hall, 1978.

[Hal03] T. Hales. The flyspeck project, 2003. See `http://code.google.com/p/flyspeck/`.

[Har96] J. Harrison. HOL Light: A Tutorial Introduction. In *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 265–269. Springer, 1996.

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

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

[HR09] F. Horozal and F. Rabe. Representing Model Theory in a Type-Theoretical Logical Framework. In *Fourth Workshop on Logical and Semantic Frameworks, with Applications*, volume 256 of *Electronic Notes in Theoretical Computer Science*, pages 49–65, 2009.

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

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

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

[Lan98] S. Mac Lane. *Categories for the working mathematician*. Springer, 1998.

[Law63] F. Lawvere. *Functional Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963.

[LCH07] D. Lee, K. Crary, and R. Harper. Towards a mechanized metatheory of Standard ML. In M. Hofmann and M. Felleisen, editors, *Symposium on Principles of Programming Languages*, pages 173–184. ACM, 2007.

[LP09] W. Lovas and F. Pfenning. Refinement Types as Proof Irrelevance. In P. Curien, editor, *Typed Lambda Calculi and Applications*, volume 5608 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2009.

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

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

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

[ML96] P. Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):3–10, 1996.

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

[MN94] L. Magnusson and B. Nordström. The ALF proof editor and its proof engine. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs*, volume 806 of *Lecture Notes in Computer Science*, pages 213–237. Springer, 1994.

[MNvOS99] O. Müller, T. Nipkow, D. von Oheimb, and O. Slotosch. HOLCF=HOL+LCF. *Journal of Functional Programming*, 9(2):191–223, 1999.

[MTP97] T. Mossakowski, A. Tarlecki, and W. Pawlowski. Combining and Representing Logical Systems. In E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science*, pages 177–196. Springer, 1997.

[Nip02] T. Nipkow. Structured Proofs in Isar/HOL. In H. Geuvers and F. Wiedijk, editors, *TYPES conference*, pages 259–278. Springer, 2002.

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

[NPW02] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[NSM01] P. Naumov, M. Stehr, and J. Meseguer. The HOL/NuPRL proof translator - a practical approach to formal interoperability. In *14th International Conference on Theorem Proving in Higher Order Logics*. Springer, 2001.

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

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

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

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

[Rab08] F. Rabe. *Representing Logics and Logic Translations*. PhD thesis, Jacobs University Bremen, 2008. Available at `http://kwarc.info/frabe/Research/phdthesis.pdf`.

[Rab10] F. Rabe. A Logical Framework Combining Model and Proof Theory. To be submitted, see `http://kwarc.info/frabe/Research/rabe_combining_09.pdf`, 2010.

[RS09] F. Rabe and C. Schürmann. A Practical Module System for LF. In J. Cheney and A. Felty, editors, *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, pages 40–48. ACM Press, 2009.

[Soj10] K. Sojakova. Toward Mechanically Verifying Logic Translations, 2010. Master's thesis, Jacobs University Bremen.

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

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

[Tar38] A. Tarski. Über Unerreichbare Kardinalzahlen. *Fundamenta Mathematicae*, 30:176–183, 1938.

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

[TB85] A. Trybulec and H. Blair. Computer Assisted Reasoning with MIZAR. In A. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 26–28, 1985.

[Try89] A. Trybulec. Tarski Grothendieck Set Theory. *Journal of Formalized Mathematics*, Axiomatics, 1989.

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

[vN25] J. von Neumann. Eine Axiomatisierung der Mengenlehre. *Journal für die reine und angewandte Mathematik*, 154:219–240, 1925.

[Wie67] N. Wiener. A Simplification of the Logic of Relations. In J. van Heijenoort, editor, *From Frege to Gödel*, pages 224–227. Harvard Univ. Press, 1967.

[Wie07] F. Wiedijk. Mizar's Soft Type System. In K. Schneider and J. Brandt, editors, *Theorem Proving in Higher Order Logics*, volume 4732 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2007.

[Zer08] E. Zermelo. Untersuchungen über die Grundlagen der Mengenlehre I. *Mathematische Annalen*, 65:261–281, 1908. English title: Investigations in the foundations of set theory I.