Mechanically Verifying Logic Translations

Kristina Sojakova

Master Thesis

School of Engineering and Sciences, Jacobs University Bremen
Reviewers: Dr. Florian Rabe, Prof. Michael Kohlhase

24. August 2010

I hereby declare that this thesis has been written independently, except where sources and collaborations are acknowledged, and has not been submitted at another university for the conferral of a degree. Parts of this thesis are based on or closely related to previously published material or material that is prepared for publication at the time of this writing, namely [32] by Florian Rabe. Parts of this thesis are the result of collaboration with Florian Rabe.

Abstract

The work done in this thesis is part of the ongoing "Logic Atlas and Integrator" project, which aims to build a comprehensive system reconciling proof and model theory, where logics and logic translations are specified dynamically and are automatically verified for correctness. We build on the work done by Florian Rabe, which uses the Twelf module system to encode logics and logic translations specified in the model-theoretic framework of institutions. As a first case study we pick the well-known model theoretic translation of modal logic to first-order logic. We evaluate how well the current framework supports the automated verification of the translation and what changes are necessary to improve it. A significant contribution is the development of the theory of logical relations for Twelf, which were found to be necessary in order to verify the soundness of the modal logic translation.

Contents

1 Introduction and Related Work

Formal methods in computer science use mathematical logic as a foundation for the specification, development, and verification of hardware and software systems. For a number of logics there are a variety of (semi-)automated verification tools available - for first-order logic (FOL) we have model checkers such as Paradox [8] and theorem provers such as SPASS [42] or Vampire [36]; for higher-order logic (HOL) we have the semi-automated theorem prover Isabelle [27].

In practice, however, complex systems are rarely described using a single logic. A larger system often consists of several smaller parts, each specified in its own logic - the so-called heterogeneous specification. This raises the issue of scalability: in order to reason about the system as a whole, it is needed to provide translations among the various logics.

Systems which are designed for the specification of logics and deductive systems in general are referred to as logical frameworks. They provide a uniform representation of object logics and allow reasoning on a logic-independent level. Common tasks include verification of derivations, search for derivations, and proving meta-theorems about deductive systems.

Logical frameworks generally fall into one of two categories. Frameworks based on set theory tend to describe logics from a model-theoretic prespective, defining the notion of truth according to Tarski. Examples of model-theoretic frameworks include institutions [12] and general logics [23], where the former will be our framework of choice for defining logics. On the other hand, frameworks based on type theory tend to describe logics from a proof-theoretic perspective, defining the notion of truth according to Curry-Howard [10, 18]. Examples include the Edinburgh Logical Framework (LF, [14]) based on dependent-type theory [24], Coq [4] based on the calculus of constructions [9], and Isabelle [27] based on simple type theory [7].

Some logical frameworks have implementations which allow the specification of logic translations. In the Heterogeneous Tool Set (Hets, [26]) based on institutions, logics and logic translations are implemented in the underlying Haskell code by instantiating a designated class. The proofs of correctness are done on paper only. The functional programming language Delphin [31], which is based on the Twelf [28] implementation of LF, allows formalizing logic translations and type-checking them for correctness. However, model theory is not represented.

Logic translations are most useful if they permit borrowing [6], i.e. reasoning about theories in one logic by means of another logic, often a tool-supported one. Such a translation would for example translate a proof obligation to FOL and discharge it by calling the existing FOL provers. The soundness of borrowing can be established in two ways: proof-theoretically by translating the obtained proof back to the original logic, or model-theoretically by exhibiting a model-translation between the two logics.

Proof-theoretic translations have been used e.g. in [20] to translate parts of dependent type theory [24] to simple type theory [7], in [41] to translate Mizar [40] into FOL, and in Scunak [5] to translate parts of dependent type theory into FOL. Similar translations have been done for simple type theory, e.g. in Omega [2], Leo-II [3] and in the sledgehammer tactic of Isabelle [27]. In practice, however, the back-translation of the proof term to the original logic is quite complicated and in the cases of Mizar and Scunak it is not done at all.

The model-theoretic approach formalizes the translation in a mathematically rigorous way within the framework of institutions [12]. Its practical advantage is that no translation of proof terms is required - the mere existence of a proof in the target logic ensures the validity of the

statement in the source logic. On the other hand, for a system utilizing such model-theoretic translation it would be desirable to have a mechanically verified proof of the correctness of the translation.

The work done in this thesis is part of the ongoing Logic ATlas and INtegrator (LATIN, [21]) project, which intends to build a comprehensive network of logic formalizations and their translations, reconciling proof and model theory. A major goal is to build a system where logics and logic translations are specified dynamically and are automatically verified for correctness. Two of the systems forming the core of LATIN are Twelf and Hets, the former representing the proof-theoretic and the latter the model-theoretic perspective.

In this thesis we build on the work done in [32], which shows how to use the Twelf module system [33] to encode logics and logic translations. As a first case study we pick the well-known model theoretic translation of modal logic to FOL. We evaluate how well the current framework supports the automated verification of the translation and what changes are necessary to improve it.

A major contribution is the development of the theory of logical relations for LF, which were found to be necessary in order to verify the soundness of the translation of modal to first-order logic. For many translations the soundness can be established by proving the commutativity of a particular diagram. However, some translations such as the modal logic translation fail to satisfy the commutativity requirement and hence a different approach is needed.

Logical relations have been used as an important tool in the study of type theories; for examples we refer the reader to [29, 35], an overview is given in [30]. In formal languages a logical relation can be thought of as a relation between two interpretations. For LF we define logical relations by purely syntactic means as certain relations between morphisms. In the absence of commutativity, a logical relation can provide a criterion for a "weak commutativity" of morphisms, which can be sufficient to establish the soundness of the translation.

We present LF and the Twelf module system in Section 2. Section 3 describes the framework of institutions and institution comorphisms. In Section 4 we give the formal treatment of logical relations for LF. In Sections 5 and 6 we show how to encode logics and logic translations respectively, giving the modal logic translation as a running example. The latter section also shows how to prove the soundness of borrowing by means of logical relations. Section 7 concludes the paper.

Acknowledgements The author would like to thank Michael Kohlhase and Florian Rabe from Jacobs University Bremen as well as Till Mossakowski and Mihai Codescu from DFKI Bremen for their help and supervision. Further, the author would like to thank Rob Simmons and Ciera & Saul Jaspan from Carnegie Mellon University for their support.

2 The Edinburgh Logical Framework

The Edinburgh Logical Framework [14] is a formal metalanguage used for the formalization of deductive systems. It is related to the Martin-Löf Type Theory [24] and the corner of the lambda cube [1] that extends simple type theory with dependent function types and kinds. We

will work with the Twelf implementation of LF [28] using the Twelf module system [33], which makes LF particularly suitable for formalizing logics and logic translations.

LF syntax distinguishes three levels of expressions - *kinds* K, *type families* A, and *terms* M. Type families are kinded; those of the distinguished kind `type` are called *types* and are used to type terms. LF *signatures* Σ are lists of kinded type family symbols $a : K$ and typed constant symbols $c : A$. LF *contexts* Γ are lists of typed variables $x : A$. Finally, LF *morphisms* $\sigma : \Sigma_1 \to \Sigma_2$ are mappings between signatures associating with each symbol in Σ_1 an expression in Σ_2 of the same level. We thus have the following abstract syntax

$$\begin{array}{lllll} \textit{Kinds} & K & := & \texttt{type} \mid \Pi_{x:A} K \\ \textit{Type Families} & A & := & a \mid A\ M \mid \lambda_{x:A} A \mid \Pi_{x:A} A \\ \textit{Terms} & M & := & c \mid x \mid M\ M \mid \lambda_{x:A} M \\ \textit{Contexts} & \Gamma & := & \cdot \mid \Gamma, x : A \\ \textit{Signatures} & \Sigma & := & \cdot \mid \Sigma, a : K \mid \Sigma, x : A \\ \textit{Morphisms} & \sigma & := & \cdot \mid \sigma, a := A \mid \sigma, c := M \end{array}$$

where Π is the dependent function type/kind constructor, λ is the corresponding function abstraction, and juxtaposition indicates application. The type $\Pi_{x:A} B$ will be written as $A \to B$ is x if not free in B.

We shall not discuss the LF typing and validity judgements here; for illustration purposes we give one rule for deriving the validity of signature morphisms and refer the reader to [14] and [33] for the rest. We have the rule

$$\frac{\vdash \sigma : \Sigma_1 \to \Sigma_2 \quad \cdot \vdash_{\Sigma_1} c : A \quad \cdot \vdash_{\Sigma_2} M : \bar{\sigma}(A)}{\sigma, c := M : \Sigma_1, c : A \to \Sigma_2}$$

indicating that $\sigma, c := M$ is a valid morphism from $\Sigma_1, c : A$ to Σ_2 if σ is a valid morphism from Σ_1 to Σ_2, c is a constant of type A over Σ_1, and M is a closed term of type $\bar{\sigma}$ over Σ_2, where $\bar{\sigma}$ denotes the homomorphic extension of σ to all closed expressions over Σ_1. From now on $\bar{\sigma}$ will be written simply as σ.

One consequence of the typing judgements is that signature morphisms preserve types and kinds, i.e. if $\sigma : \Sigma_1 \to \Sigma_2$ is a valid signature morphism and $\Gamma \vdash_{\Sigma_1} M : A$, then $\sigma(\Gamma) \vdash_{\Sigma_2} \sigma(M) : \sigma(A)$, where $\sigma(\Gamma)$ is defined in the obvious way. Similarly for kinds. Furthermore, signature morphisms preserve $\alpha\beta\eta$-equality, as was shown in [15]. Since Twelf considers $\alpha\beta\eta$-equal expressions to be identical, we make the convention that each LF/Twelf expression will be treated as a representative of its $\alpha\beta\eta$-equivalence class.

Signatures and morphisms - called *views* in Twelf - are the two top-level notions of the module system. They correspond to the LF signatures and morphisms, with the exception that a Twelf signature may contain instantiations of declared symbols of the form $c : E = E'$ in addition to conventional symbol declarations. Instantiations are introduced for efficiency and usability reasons and can be eliminated by replacing each occurrence of c by E'.

Furthermore, Twelf signatures and views may be constructed in a modular way. The module system provides two structuring mechanisms, called *inclusions* and *structures* respectively, which import symbols into their enclosing signature and give rise to a signature morphism

representing the inheritance relationship. We use the following abstract syntax:

$$
\begin{array}{lcll}
\textit{Toplevel} & G & := & \cdot \mid G,\ \%\texttt{sig}\ S\ =\ \{\Sigma\} \mid G,\ \%\texttt{view}\ v : S \to T = \{\sigma\} \\
\textit{Signatures} & \Sigma & := & \cdot \mid \Sigma,\ \%\texttt{include}\ S \mid \Sigma,\ \%\texttt{struct}\ s : S = \{\sigma\} \\
 & & & \Sigma,\ c : E \mid \Sigma,\ c : E = E \\
\textit{Morphisms} & \sigma & := & \cdot \mid \sigma,\ \%\texttt{include}\ \mu \mid \sigma,\ c := E \\
\textit{Expressions} & E & := & \texttt{type} \mid c \mid x \mid E\ E \mid [x : E]\, E \mid \{x : E\}\, E \mid E \to E \\
\textit{Composites} & \mu & := & \cdot \mid v\ \mu
\end{array}
$$

Here S, T stand for signature names, v stands for a view name, and s for a structure name. Constant symbols, both kinded and typed, are denoted by c. All LF terms, type families, and kinds are merged into the syntactic category E of expressions; brackets and braces are the Twelf syntax for the λ and Π constructors respectively. For simplicity, the constructor $\to$ is given as a separate alternative.

A structure $\%\texttt{struct}\ s : S = \{\sigma\}$ declared in a signature T imports all symbols declared in S into T, prefixed by the structure name s. The induced morphism $T.s$ maps each symbol c declared in S to $s.c$. Imported symbols are often instantiated: if S declares $c : E$ and σ contains the assignment $c := E'$, the imported symbol $s.c$ will be instantiated in T by E'.

Inclusions $\%\texttt{include}\ S$ declared in T are similar to structures but the induced signature morphism from S to T is always an inclusion. Consequently, inclusions are unnamed and may not carry instantiations. Multiple inclusions of the same signature are considered identical and the included symbols are available in T without qualification.

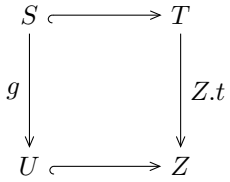
Finally, composites μ represent compositions of morphisms. They are useful in the modular development of morphisms - the declaration $\%\texttt{include}\ \mu$ includes the list of assignments given by μ into the enclosing σ.

The Twelf module system is conservative in the sense that every modular signature and view can be elaborated (flattened) into a valid non-modular signature resp. view. For our specifications we will mostly use inclusions as a structuring mechanism; however, an example below shows how we can use structures to conveniently compute pushouts for LF.

The LF signatures and morphisms form a category, which has pushouts along inclusions. Given an inclusion $S \hookrightarrow T$ and an LF morphism $g : S \to U$ implemented in Twelf, we can represent the pushout along $S \hookrightarrow T$ and g by the following Twelf signature:

```
%sig Z = {
  %include U.
  %struct t : T = {%include g}.
}.
```

The pushout thus contains all symbols from U, plus the symbols from $T \backslash S$ prefixed by t. The induced morphism from T to Z maps every symbol c of S to $g(c)$ and every other symbol d of T to $t.d$. The induced morphism from U to Z is an inclusion. We hence have the following diagram



3 Institutions and Comorphisms

We now present some definitions necessary for our work. We assume that the reader is familiar with the basic concepts of category theory and logic. For introduction to category theory see [22].

3.1 Institutions

Using categories and functors we can define an *institution*, which is a model theory-oriented formalization of a logical system first introduced by Goguen and Burstall in [12]. Institutions abstract from notions such as formulas, models, and satisfaction and structure the variety of different logics. Furthermore, they allow us to formulate logic-independent theorems for the abstract model theory of logic [11].

Definition 1 (Institution)**.** An institution is a 4-tuple $(Sig, Sen, Mod, \models)$
where

- Sig is a category
- $Sen : Sig \to \mathbf{Set}$ is a functor
- $Mod : Sig \to \mathbf{Cat}^{op}$ is a functor
- $\models$ is a family of relations $\models_\Sigma$ for $\Sigma \in |Sig|$, $\models_\Sigma \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$

such that for each morphism $\sigma : \Sigma \to \Sigma'$, sentence $F \in Sen(\Sigma)$, and model $M \in |Mod(\Sigma')|$ we have

$$Mod(\sigma)(M) \models_\Sigma F \quad \text{iff} \quad M \models_{\Sigma'} Sen(\sigma)(F)$$

The category Sig is called the *category of signatures*. The morphisms in Sig are called *signature morphisms* and represent notation changes and elargement of context. The functor Sen assigns to each signature Σ a set of *sentences* over Σ and to each morphism $\sigma : \Sigma \to \Sigma'$ the induced *sentence translation* along σ, also denoted by σ. Similarly, the functor Mod assigns to each signature Σ a category of *models* of Σ and to each morphism $\sigma : \Sigma \to \Sigma'$ the induced *model reduction* along σ, denoted by $|_\sigma$. For a signature Σ, the relation $\models_\Sigma$ is called a *satisfaction relation*. The satisfaction condition then represents the fact that truth is invariant under the change of notation and enlargement of context.

We can now use the framework of institutions to define notions on a logic-independent level.

Definition 2 (Elementary Equivalence of Models)**.** Let $(Sig, Sen, Mod, \models)$ be an institution and Σ be a signature. We say that two models $M_1, M_2 \in |Mod(\Sigma)|$ are elementary equivalent if for any $F \in Sen(\Sigma)$ we have that

$$M_1 \models_\Sigma F \quad \text{iff} \quad M_2 \models_\Sigma F$$

Definition 3 (Theories and Semantic Entailment)**.** Let $(Sig, Sen, Mod, \models)$ be an institution. For a fixed Σ, let $\Gamma \subseteq Sen(\Sigma)$ and $F \in Sen(\Sigma)$. Then the pair (Σ, Γ) is called a theory of I. Furthermore, we say that Γ entails F, denoted $\Gamma \models_\Sigma F$, if for any model $M \in |Mod(\Sigma)|$ we have that

$$\text{if } M \models_\Sigma G \text{ for all } G \in \Gamma \text{ then } M \models_\Sigma F$$

Since the concept of institutions is a model-theoretic one, we do not have a formal notion of proofs or proof calculus. There exist frameworks which add proof theory to institutions, namely general logics in [23] and the framework described in [32]. However, as our focus here is model theory, we will work with institutions and the words logic and institution will be used synonymously.

Many well-known logics can be represented as institutions quite naturally. Our running examples will be FOL and modal logic.

Example 4 (FOL). The FOL institution is given as follows:

- Signatures of FOL are the usual (finite) first-order signatures and signature morphisms are type-preserving symbol maps.
- Sentences of a signature Σ are closed first-order formulas over Σ and the sentence translation along a morphism $\sigma : \Sigma \to \Sigma'$ is the homomorphic extension of σ to $Sen(\Sigma)$.
- Models of a signature Σ are pairs (U, I), where U is a set representing the semantic universe of individuals and I is an interpretation function from Σ to U. The function I interprets each n-ary function symbol as an n-ary function on U and each n-ary predicate symbol as an n-ary relation on U. Constants are treated as nullary functions. Model morphisms are just the identities and the model reduction functor along a morphism $\sigma : \Sigma \to \Sigma'$ maps each model (U, I) of Σ' to the model $(U, \sigma; I)$ of Σ.
- Satisfaction relation on Σ is defined by the usual Tarskian satisfaction of first-order formulas.

Example 5 (Modal Logic). We will use the following institution for modal logic:

- Signatures of modal logic are (finite) sets of propositional variables and signature morphisms are maps between sets.
- Sentences of a signature Σ are given inductively by defining all variables in Σ to be sentences over Σ, and defining all expressions of the form $\top, \bot, \neg S, S \wedge T, S \vee T, S \Rightarrow T, \Box S, \diamond S$ to be sentences over Σ whenever S and T are sentences over Σ. The sentence translation along a morphism $\sigma : \Sigma \to \Sigma'$ is the homomorphic extension of σ to $Sen(\Sigma)$.

- Models of a signature Σ are given by Kripke semantics - that is, a Σ-model is a triple (W, R, F), where W is a set representing the semantic universe of possible worlds, R is a binary relation on W representing accessibility, and F is an interpretation function mapping each symbol from Σ to a unary predicate on W. The signature morphisms are just the identities and the model reduction along a morphism $\sigma : \Sigma \to \Sigma'$ maps each model (W, R, F) of Σ' to the model $(W, R, \sigma; F)$ of Σ.

- Given a signature Σ, a Σ-sentence S, and a Σ-model (W, R, F), the satisfaction relation on Σ first inductively interprets S as a unary predicate on W and then applies universal quantification. Let I denote the intermediate interpretation function. If $S = \diamond T$, for instance, then S^I is defined to hold in a world w iff T^I holds in some world w' for which $w \; R \; w'$. Finally, S is defined to hold in (W, R, F) iff S^I holds in all worlds $w \in W$.

3.2 Institution Comorphisms

Analogously to the representation of logics, logic translations can be formalized as certain mappings between institutions, as described in [13]. In [39], Tarlecki specifically suggests using LF as a framework suitable for formalizing translations between institutions. Below we present the main definitions, the most important for our purposes being that of an *institution comorphism*.

Definition 6 (Institution Comorphism)**.** Let $I = (Sig^I, Sen^I, Mod^I, \models^I)$ and $J = (Sig^J, Sen^J, Mod^J, \models^J)$ be two institutions. An institution comorphism from I to J is a triple (Φ, α, β) where

- $\Phi : Sig^I \to Sig^J$ is a functor
- $\alpha : Sen^I \to \Phi; Sen^J$ is a natural transformation
- $\beta : Mod^I \leftarrow \Phi; Mod^J$ is a natural transformation

such that for each $\Sigma \in |Sig^I|$, $F \in Sen^I(\Sigma)$, and $M \in |Mod^J(\Phi(\Sigma))|$ we have

$$\beta_\Sigma(M) \models^I_\Sigma F \quad \text{iff} \quad M \models^J_{\Phi(\Sigma)} \alpha_\Sigma(F)$$

The satisfaction condition expresses the fact that truth is invariant under translation. We also have the dual notion of an *institution morphism*, where α and β are contra- and covariant, respectively.

Definition 7 (Institution Morphism)**.** Let $I = (Sig^I, Sen^I, Mod^I, \models^I)$ and $J = (Sig^J, Sen^J, Mod^J, \models^J)$ be two institutions. An institution morphism from I to J is a triple (Φ, α, β) where

- $\Phi : Sig^I \to Sig^J$ is a functor
- $\alpha : Sen^I \leftarrow \Phi; Sen^J$ is a natural transformation
- $\beta : Mod^I \to \Phi; Mod^J$ is a natural transformation

such that for each $\Sigma \in |Sig^I|$, $F \in Sen^J(\Phi(\Sigma))$, and $M \in |Mod^I(\Sigma)|$ we have

$$M \models^I_\Sigma \alpha_\Sigma(F) \quad \text{iff} \quad \beta_\Sigma(M) \models^J_{\Phi(\Sigma)} F$$

The following definition gives a very important property of institution comorphisms.

Definition 8 (Model Expansion)**.** Let (Φ, α, β) be an institution comorphism from I to J. We say that the comorphism has the model expansion property if each functor β_Σ for $\Sigma \in Sig^I$ is surjective on objects.

For the purposes of borrowing, it is enough to have a slightly weaker form of model expansion, as the following well-known theorem illustrates.

Lemma 9 (Borrowing)**.** *Let (Φ, α, β) be an institution comorphism from I to J having model expansion up to elementary equivalence. Then for any theory (Σ, Γ) of I and a sentence F over Σ, we have that*

$$\Gamma \models^I_\Sigma F \quad \textit{iff} \quad \alpha_\Sigma(\Gamma) \models^J_{\Phi(\Sigma)} \alpha_\Sigma(F)$$

where α_Σ on sets of sentences is defined in the obvious way.

In other words, we can use the institution J to reason about theories in I. For more on borrowing, see [6].

Example 10 (Translation of Modal Logic to FOL)*.* The translation of modal logic to FOL is as follows:

- Given a modal logic signature Σ, we obtain the corresponding FOL signature $\Phi(\Sigma)$ by declaring each symbol from Σ as a unary predicate symbol in $\Phi(\Sigma)$, and adding an additional binary predicate symbol *acc* representing the accessibility relation. The translation of a signature morphism $\sigma : \Sigma \to \Sigma'$ is σ itself, extended with identity on *acc*.
- For a fixed Σ, the translation of a Σ-sentence S is given by first inductively translating S to a first-order unary predicate and then applying universal quantification. Let α'_Σ denote this intermediate translation. If $S = \diamond S'$, for instance, then we define $\alpha'_\Sigma(S) = \exists y.\alpha'_\Sigma(S')(y) \wedge acc\ x\ y$. The resulting sentence translation is given by $\alpha_\Sigma(S) = \forall \alpha'_\Sigma(S)$. We note that α'_Σ is the syntactic equivalent of the interpretation function I used to determine the satisfaction relation. Similarly, α_Σ is the syntactic equivalent of interpreting a sentence as a statement about the semantic universe of worlds.
- For a fixed Σ, each first-order model (U, I) of $\Phi(\Sigma)$ is translated to the Σ-model (U, acc^I, I'), where I' is the restriction of I to symbols different from *acc*.

4 Logical Relations

As was mentioned in Section 1, a logical relation can be viewed as a relation between interpretations: For every type t, a logical relation ρ from a model M to a model N provides a relation $\rho_t \subseteq t^M \times t^N$. Logical relations can also be seen as a generalization of model morphisms: Instead of a map $\rho_t : t^M \to t^N$, a relation is used in higher-order languages because there is no natural inductive definition of a map $\rho_{s \to t}$ in terms of maps ρ_s and ρ_t. However, such logical relations are not in general closed under composition.

The characteristic feature of the definition of logical relations is that functions are related if and only if they map related arguments to related values. The central result is the Basic Lemma, which states that terms with free variables are interpreted as related values if related values are substituted for the free variables. The definition and lemma generalize easily to n-ary logical relations.

There have been several results regarding more general classes of relations [37, 25]. These still satisfy the basic lemma and enjoy better closure properties but have less intuitive definitions. The largest class of relations satisfying the basic lemma is studied systematically in [17] and [30] where they are called prelogical and lax logical relations, respectively. The authors give several characterizations of these relations, and the most intuitive one for our purposes is the following: A lax logical relation is analogous to a logical relation except it has to relate two functions only when they are expressible by the same term.

The above restriction to λ-definable functions indicates a syntactical flavor of lax logical relations, and our purpose here is to capture it by purely syntactical means. Consequently, we do not consider relations between models of type theory. Instead, we appeal to the category of signatures and signature morphisms and use the fact that signature morphisms from S to T behave like models: Both a model of S and a morphism from S to T are inductively defined interpretations of S. In fact, models can be seen as morphisms where the codomain T is implicitly the semantic universe. This also casts a new light on the structural logical relations employed in [38], where S is an LF-encoding of simple type theory and T a meta-logic interpreting S.

This approach is applicable to all type theories that are expressive enough to internalize the "forall ... if ... then ..." statement that occurs in the case of function types. This includes in particular dependent type theory as in LF [14] and higher-order logic with polymorphism as in Isabelle [27].

In this section we fix arbitrary LF signatures S and T and LF signature morphisms $\mu_i : S \to T$ for $i = 1, \ldots, n$. The intuition behind the definition of a (syntactical) logical relation ρ on $\mu_1, \ldots, \mu_n$ is as follows: For every closed type A in S, $\rho(A)$ should be a type family of kind $\mu_1(A) \to \ldots \to \mu_n(A) \to \texttt{type}$ in T, giving the relation on A. Furthermore, for every closed term $t : A$ in S, $\rho(t)$ should be a term of type $\rho(A)\ \mu_1(t)\ \ldots\ \mu_n(t)$, proving the Basic Lemma for t.

Notation 11. Below are some conventions and notations we will use.

- For an expression E, we denote by $\mu_i(E)$ the expression obtained by translating E along μ_i and indexing all free variables in the translated expression by $.^i$
- For a context $\Gamma = x_1 : A_1, \ldots, x_n : A_m$, the expression $\lambda_{x_1:A_1} \ldots \lambda_{x_n:A_m} A$ will be denoted by $\lambda_\Gamma A$. Similarly for Π.
- For a tuple $\vec{N} = (N_1, \ldots, N_m)$, the expression $E\ N_1\ \ldots\ N_m$ will be denoted by $E\ \vec{N}$.
- We denote by $\vec{x}$ the tuple $(x^1, \ldots, x^n)$ and by $\vec{\mu}(E)$ the tuple $(\mu^1(E), \ldots, \mu^n(E))$. If $\vec{E} = (E_1, \ldots, E_n)$, the context $x^1 : E_1, \ldots, x^n : E_n$ will be denoted by $\vec{x} : \vec{E}$ and the substitution $x^1/E_1, \ldots, x^n/E_n$ by $\vec{x}/\vec{E}$.

We are now ready to give the formal definition of logical relations in LF. A logical relation ρ will consists of 3 maps, all denoted by ρ, on contexts, kinds, and terms/type families of S

respectively. For kinds K, ρ takes a type family C as an additional argument and is written as $\rho^C(K)$.

Definition 12 (Logical Relations)**.** Assume the morphisms μ_i are given and let ρ be a function associating with each constant in S an expression in T. We extend ρ inductively as follows.

- On contexts: If Γ is a context in S, then $\rho(\Gamma)$ is defined by
  - $\rho(\cdot) = \cdot$
  - $\rho(\Gamma, x : A) = \rho(\Gamma),\ \vec{x} : \vec{\mu}(A),\ \kappa_x : \rho(A)\ \vec{x}$
- On kinds: If C is a Γ-type family of kind K in S, then $\rho^C(K)$ is defined by
  - $\rho^A(\texttt{type}) = \mu_1(A) \to \ldots \to \mu_n(A) \to \texttt{type}$
  - $\rho^C(\Pi_{x:A} K) = \Pi_{\vec{x}:\vec{\mu}(A)} \Pi_{\kappa_x:\rho(A)\ \vec{x}}\ \rho^{(C\ x)}(K)$
- On terms and type families: If M is a Γ-term or type family in S, then $\rho(M)$ is defined by
  - $\rho(x) = \kappa_x$
  - $\rho(M\ N) = \rho(M)\ \vec{\mu}(N)\ \rho(N)$
  - $\rho(\lambda_{x:A} M) = \lambda_{\vec{x}:\vec{\mu}(A)} \lambda_{\kappa_x:\rho(A)\ \vec{x}}\ \rho(M)$
  - $\rho(\Pi_{x:A} B) = \lambda_{\vec{f}:\vec{\mu}(\Pi_{x:A} B)} \Pi_{\vec{x}:\vec{\mu}(A)} \Pi_{\kappa_x:\rho(A)\ \vec{x}}\ \rho(B)\ (f^1\ x^1)\ \ldots\ (f^n\ x^n)$
    where f is not free in B.

Then ρ is called a *relation*. Furthermore, ρ is a *logical relation* if the following conditions holds for each symbol in S:

- If $a : K \in S$ for a kind K then $\cdot \vdash_T \rho(a) : \rho^a(K)$.
- If $c : A \in S$ for a type A then $\cdot \vdash_T \rho(c) : \rho(A)\ \vec{\mu}(c)$.

Notation 13. A logical relation ρ will be denoted by $\rho : \mu_1 \Rightarrow \ldots \Rightarrow \mu_n : S \to T$.

Lemma 14 (Restrictions)**.** *Let $\rho : \mu_1 \Rightarrow \ldots \Rightarrow \mu_n$ be a logical relation. Let S' bes a subsignature of S and $\mu'_1, \ldots, \mu'_n$ be the restrictions of $\mu_1, \ldots, \mu_n$ to S'. Furthermore, let ρ' be the restriction of ρ to (the contexts and expressions of) S'. Then $\rho' : \mu'_1 \Rightarrow \ldots \Rightarrow \mu'_n$ is a logical relation.*

Proof. Obvious. □

Lemma 15 (Substitution)**.** *Let ρ be a relation. Let $(\Gamma, x : A, \Delta)$ be an S-context and M be such that $\Gamma \vdash_S M : A$. Then for any $(\Gamma, x : A, \Delta)$-term or type family E in S we have*

$$\rho(E[x/M]) = \rho(E)[\vec{x}/\vec{\mu}(M), \kappa_x/\rho(M)]$$

and for any $(\Gamma, x : A, \Delta)$-type family C of kind K we have

$$\rho^{(C[x/M])}(K[x/M]) = \rho^C(K)[\vec{x}/\vec{\mu}(M), \kappa_x/\rho(M)]$$

Proof. It is easy to show the analogous claim for μ^i: *For any S-expression E' we have*

$$\mu_i(E'[x/M]) = \mu_i(E')[\vec{x}/\vec{\mu}(M), \kappa_x/\rho(M)]$$

Using this result it is straightfoward to prove the lemma by induction on the structure of E resp. K. □

Theorem 16 (Basic Lemma)**.** *Let ρ be a logical relation. Then we have the following invariants:*

- *If* $\Gamma \vdash_S A : K : \texttt{kind}$ *then*

$$\rho(\Gamma) \vdash_T \rho(A) : \rho^A(K)$$

- *If* $\Gamma \vdash_S M : A : \texttt{type}$ *then*

$$\rho(\Gamma) \vdash_T \rho(M) : \rho(A)\ \vec{\mu}(M)$$

Proof. Let E be a Γ-term or type family in S. We first prove a weaker form of the theorem, relying on the assumption that $\rho(\Gamma)$ is a valid T-context. We proceed by induction on the structure of E:

- For type families:
  - $E \equiv a$. Let K be such that $\Gamma \vdash_S a : K$. Then we must have $a : K \in S$. By definition

$$\cdot \vdash_T \rho(a) : \rho^a(K)$$

    Since $\rho(\Gamma)$ is by assumption well-formed, we have

$$\rho(\Gamma) \vdash_T \rho(a) : \rho^a(K)$$

    as desired.
  - $E \equiv C\ M$. Assume $\Gamma \vdash_S C : \Pi_{x:A}K$ and $\Gamma \vdash_S M : A$. Then $\Gamma \vdash_S C\ M : K[x/M]$. We have

$$\rho(C\ M) = \rho(C)\ \vec{\mu}(M)\ \rho(M)$$

    Using the inductive hypothesis for C and M we have

$$\rho(\Gamma) \vdash_T \rho(C) : \rho^C(\Pi_{x:A}K)$$
$$\rho(\Gamma) \vdash_T \rho(M) : \rho(A)\ \vec{\mu}(M)$$

    Now

$$\rho^C(\Pi_{x:A}K) = \Pi_{\vec{x}:\vec{\mu}(A)}\Pi_{\kappa_x:\rho(A)\ \vec{x}}\ \rho^{(C\ x)}(K)$$

    Hence

$$\rho(\Gamma) \vdash_T \rho(C\ M) : \rho^{(C\ x)}(K)[\vec{x}/\vec{\mu}(M), \kappa_x/\rho(M)]$$

Now by Lemma 15

$$\rho^{(C\ x)}(K)[\vec{x}/\vec{\mu}(M), \kappa_x/\rho(M)] = \rho^{(C\ M)}(K[x/M])$$

which yields

$$\rho(\Gamma) \vdash_T \rho(C\ M) : \rho^{(C\ M)}(K[x/M])$$

as desired.

– $E \equiv \lambda_{x:A} C$. Assume $\Gamma \vdash_S \lambda_{x:A} C : \Pi_{x:A} K$. Then $\Gamma, x : A \vdash_S C : K$. We have

$$\rho(\lambda_{x:A} C) = \lambda_{\vec{x}:\vec{\mu}(A)} \lambda_{\kappa_x:\rho(A)\ \vec{x}}\ \rho(C)$$

Furthermore

$$\rho^{(\lambda_{x:A} C)}(\Pi_{x:A} K) = \Pi_{\vec{x}:\vec{\mu}(A)} \Pi_{\kappa_x:\rho(A)\ \vec{x}}\ \rho^C(K)$$

Now

$$\rho(\Gamma, x : A) = \rho(\Gamma),\ \vec{x} : \vec{\mu}(A),\ \kappa_x : \rho(A)\ \vec{x}$$

By assumption $\rho(\Gamma)$ is well-formed. Furthermore, it is easy to see that $\mu_i(A)$ is a valid $\rho(\Gamma)$-type. Using the inductive hypothesis for A then shows that $\rho(A)$ is a valid $\rho(\Gamma)$-type as well. Thus we conclude that $\rho(\Gamma, x : A)$ is a valid context and we can use the inductive hypothesis for C to obtain

$$\rho(\Gamma, x : A) \vdash_T \rho(C) : \rho^C(K)$$

This implies

$$\rho(\Gamma) \vdash_T \rho(\lambda_{x:A} C) : \rho^{(\lambda_{x:A} C)}(\Pi_{x:A} K)$$

as desired.

– $E \equiv \Pi_{x:A} B$. Then $\Gamma \vdash_S A : \texttt{type}$ and $\Gamma, x : A \vdash_S B : \texttt{type}$. We have

$$\rho(\Pi_{x:A} B) = \lambda_{\vec{f}:\vec{\mu}(\Pi_{x:A} B)} \Pi_{\vec{x}:\mu(A)} \Pi_{\kappa_x:\rho(A)\ \vec{x}} \rho(B)\ (f^1\ x^1)\ \ldots\ (f^n\ x^n)$$

It is easy to see that $\mu_i(\Pi_{x:A} B)$ and $\mu_i(A)$ are valid $\rho(\Gamma)$-types in T. By induction hypothesis for A we have

$$\rho(\Gamma) \vdash_T \rho(A) : \mu_1(A) \to \ldots \to \mu_n(A) \to \texttt{type}$$

what implies that $\rho(A)\ \vec{x}$ is also a valid $\rho(\Gamma)$-type in T. Now denote by Δ the valid context

$$\rho(\Gamma),\ \vec{f} : \vec{\mu}(\Pi_{x:A} B),\ \vec{x} : \vec{\mu}(A),\ \kappa_x : \rho(A)\ \vec{x}$$

Then

$$\Delta \vdash_T f^i\ x^i : \mu_i(B)$$

As in the previous case, $\rho(\Gamma, x : A)$ is a valid $\rho(\Gamma)$-context and by induction hypothesis for B we have

$$\rho(\Gamma, x : A) \vdash_T \rho(B) : \mu_1(B) \to \ldots \to \mu_n(B) \to \texttt{type}$$

Since $\rho(\Gamma, x : A) \subseteq \Delta$, we have

$$\Delta \vdash_T \rho(B)\ (f^1\ y^1)\ \ldots\ (f^n\ y^n) : \texttt{type}$$

which shows that

$$\rho(\Gamma) \vdash_T \rho(\Pi_{x:A}B) : \mu_1(\Pi_{x:A}B) \to \ldots \to \mu_n(\Pi_{x:A}B) \to \texttt{type}$$

as desired.

- For terms:
  - $E \equiv c$. This case is analogous to the case $E \equiv a$ for type families.
  - $E \equiv x$. Then $\rho(x) = \kappa_x$. Let A be such that $\Gamma \vdash_S x : A$. Then Γ must have the form $\Gamma_1, x : A, \Gamma_2$ and $\rho(\Gamma)$ has the form

    $$\rho(\Gamma_1),\ \vec{x} : \vec{\mu}(A),\ \kappa_x : \rho(A)\ \vec{x},\ \Delta$$

    Since $\rho(\Gamma)$ is well-formed, we have

    $$\rho(\Gamma) \vdash_T \kappa_x : \rho(A)\ \vec{x}$$

    Thus as desired

    $$\rho(\Gamma) \vdash_T \rho(x) : \rho(A)\ \vec{\mu}(x)$$

  - $E \equiv M\ N$. Assume $\Gamma \vdash_S M : \Pi_{x:A}B$ and $\Gamma \vdash_S N : A$. Then $\Gamma \vdash_S M\ N : B[x/N]$. We have

    $$\rho(M\ N) = \rho(M)\ \vec{\mu}(N)\ \rho(N)$$

    Using the inductive hypothesis for M and N we have

    $$\begin{aligned} \rho(\Gamma) &\vdash_T \rho(M) : \rho(\Pi_{x:A}B)\ \vec{\mu}(M) \\ \rho(\Gamma) &\vdash_T \rho(N) : \rho(A)\ \vec{\mu}(N) \end{aligned}$$

    Now by β-reduction

    $$\begin{aligned} \rho(\Pi_{x:A}B)\ \vec{\mu}(M) = \Pi_{\vec{x}:\vec{\mu}(A)}\Pi_{\kappa_x:\rho(A)\ \vec{x}} \\ \rho(B)\ (\mu_1(M)\ y^1)\ \ldots\ (\mu_n(M)\ y^n) \end{aligned}$$

Hence

$$\rho(\Gamma) \vdash_T \rho(M\ N) : \rho(B)[\vec{x}/\vec{\mu}(N), \kappa_x/\rho(N)]$$
$$(\mu_1(M)\ \mu_1(N))\ (\mu_n(M)\ \mu_n(N))$$

Now by Lemma 15

$$\rho(B)[\vec{x}/\vec{\mu}(N), \kappa_x/\rho(N)] = \rho(B[x/N])$$

which yields

$$\rho(\Gamma) \vdash_T \rho(M\ N) : \rho(B[x/N])\ \vec{\mu}(M\ N)$$

as desired.

– $E \equiv \lambda_{x:A} M$. Assume $\Gamma \vdash_S \lambda_{x:A} M : \Pi_{x:A} B$. Then $\Gamma, x : A \vdash_S M : B$. We have

$$\rho(\lambda_{x:A} M) = \lambda_{\vec{x}:\vec{\mu}(A)} \lambda_{\kappa_x:\rho(A)\ \vec{x}}\ \rho(M)$$

Employing β-reduction we have

$$\rho(\Pi_{x:A} B)\ \vec{\mu}(\lambda_{x:A} M) = \Pi_{\vec{x}:\vec{\mu}(A)} \Pi_{\kappa_x:\rho(A)\ \vec{x}}$$
$$\rho(B)\ (\mu_1(\lambda_{x:A} M)\ x^1)\ \ldots\ (\mu_n(\lambda_{x:A} M)\ x^n)$$

As in the case for type families, the context $\rho(\Gamma, x : A)$ is well-formed. Thus by inductive hypothesis for M we have

$$\rho(\Gamma, x : A) \vdash_T \rho(M) : \rho(B)\ \vec{\mu}(M)$$

Since $\mu_i(\lambda_{x:A} M)\ x^i = \mu_i(M)$, we have

$$\rho(\Gamma) \vdash_T \rho(\lambda_{x:A} M) : \rho(\Pi_{x:A} B)\ \vec{\mu}(\lambda_{x:A} M)$$

as desired.

Using the claim we have just proved it is easy to show by induction on the length of Γ that $\rho(\Gamma)$ is well-formed. This in combination with the claim proves the theorem. □

Lemma 17 (Left Composition with a Morphism)**.** *Let $\rho : \mu_1 \Rightarrow \ldots \Rightarrow \mu_n : S \to T$ be a logical relation and $r : T \to U$ be a morphism. Then $(r;\rho) : (r;\mu_1) \Rightarrow \ldots \Rightarrow (r;\mu_n) : S \to U$ defined by*

$$(r;\rho)(\Gamma) = \rho(r(\Gamma))$$
$$(r;\rho)(C) = \rho(r(C))$$
$$(r;\rho)^C(K) = \rho^{(r(C))}(r(K))$$

is a logical relation.

Proof. Follows from the type preservation theorem. □

Lemma 18 (Right Composition with a Morphism)**.** *Let $\rho : \mu_1 \Rightarrow \ldots \Rightarrow \mu_n : S \to T$ be a logical relation and $r : T \to U$ be a morphism. Then $(\rho; r) : (\mu_1; r) \Rightarrow \ldots \Rightarrow (\mu_n; r) : S \to U$ defined by*

$$\begin{aligned}(\rho; r)(\Gamma) &= r(\rho(\Gamma))\\ (\rho; r)(C) &= r(\rho(C))\\ (\rho; r)^C(K) &= r(\rho^C(K))\end{aligned}$$

is a logical relation.

Proof. Follows from the type preservation theorem. □

We now show how logical relations can be used to reason about the inhabitance of types, an application that will be utilized in Section 6 to prove properties about logic translations. We have the following notion and lemma for binary logical relations.

Definition 19 (Transition Pairs)**.** Let $\rho : \mu_1 \Rightarrow \mu_2 : S \to T$ be a logical relation and C be a type family in S of kind Π_Γ `type`, for $\Gamma = x_1 : A_1, \ldots, x_m : A_m$. A pair (Ψ, Ξ) of terms in T is called a transition pair for C under ρ if we have

$$\begin{aligned}\cdot \vdash_T \Psi &: \Pi_{\rho(\Gamma)}\ \mu_1(C)\ x_1^1\ \ldots\ x_m^1 \to \mu_2(C)\ x_1^2\ \ldots\ x_m^2\\ \cdot \vdash_T \Xi &: \Pi_{\rho(\Gamma)}\ \mu_2(C)\ x_1^2\ \ldots\ x_m^2 \to \mu_1(C)\ x_1^1\ \ldots\ x_m^1\end{aligned}$$

Lemma 20. *Let $\rho : \mu_1 \Rightarrow \mu_2 : S \to T$ be a logical relation and C be a type family in S of kind Π_Γ `type`, $\Gamma = x_1 : A_1, \ldots, x_m : A_m$. Assume there exists a transition pair (Ψ, Ξ) for C under ρ. If $(M_1, \ldots, M_m)$ is a spine of type Γ, then the type $\mu_1(C\ M_1\ \ldots\ M_m)$ is inhabited if and only if the type $\mu_2(C\ M_1\ \ldots\ M_m)$ is inhabited.*

Proof. Let $\Gamma = x_1 : A_1, \ldots, x_m : A_m$. By assumption we have

$$\cdot \vdash_S M_i : A_i[x_1/M_1] \ldots [x_{i-1}/M_{i-1}]$$

Thus by the Basic Lemma 16

$$\cdot \vdash_T \rho(M_i) : \rho(A_i[x_1/M_1] \ldots [x_{i-1}/M_{i-1}])\ \vec{\mu}(M_i)$$

By repeated application of Lemma 15 we have

$$\begin{aligned}\rho(A_i[x_1/M_1] \ldots [x_{i-1}/M_{i-1}]) = \rho(A_i)[&\vec{x}_1/\vec{\mu}(M_1),\\ &\kappa_{x_1}/\rho(M_1)] \ldots [\vec{x}_{i-1}/\vec{\mu}(M_{i-1}), \kappa_{x_1}/\rho(M_{i-1})]\end{aligned}$$

Thus

$$\begin{aligned}\cdot \vdash_T \Psi\ \vec{\mu}(M_1)\ \rho(M_1)\ \ldots\ \vec{\mu}(M_m)\ \rho(M_m) :&\\ \mu_1(C)\ \mu_1(M_1)\ \ldots\ \mu_1(M_m) \to \mu_2(C)\ \mu_2(M_1)\ \ldots\ \mu_2(M_m)&\end{aligned}$$

$$\begin{aligned}\cdot \vdash_T \Xi\ \vec{\mu}(M_1)\ \rho(M_1)\ \ldots\ \vec{\mu}(M_m)\ \rho(M_m) :&\\ \mu_2(C)\ \mu_2(M_1)\ \ldots\ \mu_2(M_m) \to \mu_1(C)\ \mu_1(M_1)\ \ldots\ \mu_1(M_m)&\end{aligned}$$

This shows that the types

$$\mu_1(C\ M_1\ \ldots\ M_m) \to \mu_2(C\ M_1\ \ldots\ M_m)$$
$$\mu_2(C\ M_1\ \ldots\ M_m) \to \mu_1(C\ M_1\ \ldots\ M_m)$$

are both inhabited. Hence the type $\mu_1(C\ M_1\ \ldots\ M_m)$ is inhabited if and only if the type $\mu_2(C\ M_1\ \ldots\ M_m)$ is inhabited. □

5 Encoding Logics

The full framework for encoding logics and their translations in Twelf was introduced in [32]. Here we leave out the representation of proof theory and focus on the model-theoretic aspects only.
Let *Base* be the Twelf signature

```
%sig Base = {
  o   : type.
  ded : o → type.
}.
```

Furthemore, let F_0, F be two fixed Twelf signatures and $P : F_0 \to F$ be a Twelf morphism. The signature F will be called a *foundation* and will usually be an encoding of a particular set theory; in our case F encodes the Zermelo-Fraenkel set theory with the axiom of choice (ZFC).

An encoding of a foundation, however, tends to be very large and hard to work with. For this reason, we often consider a fragment of the foundation, which is expressive enough to axiomatize the semantics of many logics while staying simple enough to work with. The signature F_0 represents the fragment and P provides a translation to the full foundation. In our framework the fragment will be an encoding of higher-order logic with functional extensionality.

A logic in Twelf will be represented as a diagram in the LF category of signatures having the following form:

$$\begin{array}{ccccc} Base & \xrightarrow[l^{truth}]{} & L^{syn} & \xrightarrow[l^{mod}]{} & L^{mod} \\ & & & & \uparrow \\ & & & & F_0 \xrightarrow{P} F \end{array}$$

where we require that $l^{truth}(ded)$ is a symbol in L^{syn} rather than an arbitrary type. The reason for this restriction will become clear in Section 6.

The intuition is that $l^{truth}(o)$ represents the syntactic class of formulas and $l^{truth}(ded)$ represents the truth of formulas. While in many cases l^{truth} will be an inclusion, in some cases it is needed that l^{truth} map o to a different type, as for instance in the encoding of higher-order logic. The symbol *ded* will usually be included in L^{syn} and mapped to itself.

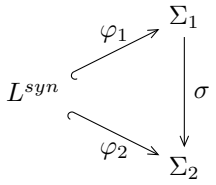
L^{syn} declares the syntax of the encoded logic, i.e. LF symbols that represent syntactic classes (e.g. terms or formulas), sorts, functions, predicates, logical connectives, etc. L^{mod} declares the model theory, i.e. LF symbols that axiomatize the properties of models.

The fragment F_0 serves as a metatheory for axiomatizing the model theory of logics and the foundation F serves as a metalanguage for expressing particular models. The morphism l^{mod} represents (part of) the interpretation function that translates syntax into a semantic realm.

We will now show how the above diagram induces an institution.

5.1 Signatures and Sentences

The signature category is defined to be the category of LF inclusion slices out of L^{syn}. That is, the signatures are LF morphisms of the form $\varphi : L^{syn} \hookrightarrow \Sigma$ for some Σ and the signature morphisms $\sigma : \varphi_1 \to \varphi_2$ are those LF morphisms which make the following triangle commute:



The identity morphisms and composition are inherited from LF. It is easy to see that the inclusion slices form a category. This definition, however, is not completely adequate because in many cases the logic being encoded is less expressive than LF itself. For this reason we often need to specify additional constraints in order to obtain the desired institution.

For instance, we might need to place restrictions on the form of declarations which can occur in the extended part of Σ. In case of modal logic, for example, we only want to allow declarations of propositional variables, i.e. having the form $p : o$. Declaring symbols of arbitrary types/kinds should not be permitted.

Similarly, we might need to consider only signature morphisms which have certain properties. For modal logic as well as FOL, signature morphisms only map symbols to symbols. LF morphisms, however, are allowed to map symbols to arbitrary expressions of the appropriate type/kind.

One possible solution to both problems is to introduce a set of patterns, which describe the form of permitted declarations/morphisms. A preliminary syntax for patterns has already been developed by Florian Rabe and Fulya Horozal but it has yet to be implemented for Twelf. For now we use the above definition and do not discuss the issue of patterns in detail.

Example 21 (Modal Logic Syntax). The syntax *ML* of modal logic (having the role of the the signature L^{syn}) is based on the syntax *MPL* for minimal propositional logic, which declares the connectives $\bot, \Rightarrow$ and defines all other propositional connectives in terms of these two. The modal logic syntax then adds the two additional connectives $\Box, \Diamond$. We illustrate below the encoding of $\Diamond$.

```
%sig Possibility = {
  %include Base.
  ◇ : o → o.
}.

%sig ML = {
```

```
  %include MPL.
  %include Necessity.
  %include Possibility.
}.
```

As can be seen from the above encodings, the morphism l^{truth} for modal logic is an inclusion.

Example 22 (FOL Syntax). The syntax *FOL* of first-order logic is based on the syntax *PL* for propositional logic and an extension *BaseFOL* of the *Base* signature, which declares the type i of individuals. The FOL syntax then adds the two quantifiers $\forall, \exists$. We illustrate below the encoding of $\forall$.

```
%sig Forall = {
  %include BaseFOL.
  ∀ : (i → o) → o.
}.

%sig FOL = {
  %include BaseFOL.
  %include PL.
  %include Forall.
  %include Exists.
}.
```

The morphism l^{truth} for FOL is likewise an inclusion.

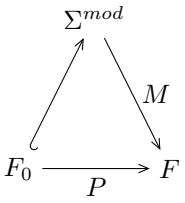
The sentences over a signature $\varphi : L^{syn} \hookrightarrow \Sigma$ are defined to be the LF terms of type $l^{truth}(o)$ over Σ. If $\sigma : \varphi_1 \to \varphi_2$ is a signature morphism, its restriction to $Sen(\varphi_1)$ is the desired sentence translation along σ. Since by definition any signature morphism σ is the identity on L^{syn}, it is straightforward to show that Sen is indeed a functor from the category of signatures to the category **Set** of sets.

5.2 Models and Satisfaction Relation

Given a signature $\varphi : L^{syn} \hookrightarrow \Sigma$, we construct an LF morphism φ^{mod} by a pushout along φ and l^{mod}, where Σ^{mod} denotes the pushout:

$$\begin{array}{ccc} L^{syn} & \xrightarrow{\varphi} & \Sigma \\ {\scriptstyle l^{mod}}\downarrow & & \downarrow{\scriptstyle \varphi^{mod}} \\ L^{mod} & \xrightarrow[\iota_\varphi]{} & \Sigma^{mod} \end{array}$$

The category of models of φ is defined to be the discrete category of those LF morphisms $M : \Sigma^{mod} \to F$ which make the following diagram commute



This guarantees that symbols of the methatheory have a fixed semantics. The composition $\varphi^{mod}; M$ then interprets the symbols of Σ in the semantic realm encoded by F. Since model morphisms are not essential for presenting the notion of institution, we omit them for simplicity.

The advantage of this definition is that models are now purely syntactical entities and hence are easier to express and reason about. On the other hand, it is clear that not every model in the original logic can be represented as an LF morphism in this way - a signature can have uncountably many models but there are only countably many LF morphisms between two fixed signatures. We will discuss this issue in more detail in Section 5.3.

Let $\sigma : \varphi_1 \to \varphi_2$ be a signature morphism. Then we have

$$\varphi_1 \ ; \ \sigma \ ; \ \varphi_2^{mod} = \varphi_2 \ ; \ \varphi_2^{mod}$$

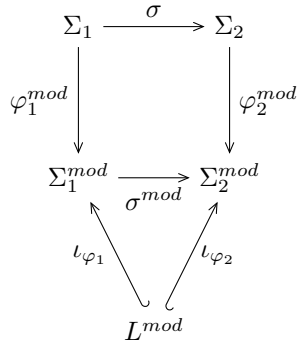
by definition of σ and

$$\varphi_2 \ ; \ \varphi_2^{mod} = l^{mod} \ ; \ \iota_{\varphi_2}$$

by definition of φ_2^{mod}. Thus

$$\varphi_1 \ ; \ \sigma \ ; \ \varphi_2^{mod} = l^{mod} \ ; \ \iota_{\varphi_2} \tag{1}$$

and by the universal property of the pushout Σ_1^{mod} there exists a unique morphism σ^{mod} such that the following LF diagram commutes



The model reduction functor along σ is then defined by mapping each model M of φ_2 to the model $\sigma^{mod}; M$ of φ_1. By uniqueness of the induced σ^{mod} morphism, it follows that if $\sigma : \varphi \to \varphi$ is an identity, then σ^{mod} is an LF identity. Similarly, if $\sigma_1 : \varphi_1 \to \varphi_2, \sigma_2 : \varphi_2 \to \varphi_3$ then $(\sigma_1; \sigma_2)^{mod} = \sigma_1^{mod}; \sigma_2^{mod}$. This proves that Mod is indeed a functor from the signature category to the dual category $\mathbf{Cat}^{op}$ of categories.

Example 23 (Foundation). The encodings of ZFC and HOL are described in [19]. Our foundation will be the signature *TypedZF*, which encodes the ZFC set theory and adds additional type constructors and operations on top of it, allowing us for instance to construct the type of elements of a given set or to quantify over the elements of a given set. We shall not go into further details here and refer the reader to [19].
The fragment is represented by the signature *HOL*:

```
%sig HOL = {
  set  : type.
  elem : set → type.

  ⟹ : set → set → type.          %infix ⟹.
  λ : (elem A → elem B) → elem (A ⟹ B).
  @ : elem (A ⟹ B) → (elem A → elem B).
  ...
}.
```

The type *set* represents the type of all sets and the type constructor *elem* returns the type of elements of a given set. The operator $\Longrightarrow$ constructs the set of functions between two given sets and $\lambda, @$ allow us to move between the LF function space and its set-theoretic equivalent. The operators $\Longrightarrow, \lambda, @$ thus act as the function constructor, abstraction, and application, respectively, for sets.

HOL further declares the type o of formulas and the constructor *ded* for truth judgements. It specifies the usual propositional connectives *true, false, not, and, or, imp* and allows to quantify over the elements of a given set using *forall* and *exists*. Finally, *HOL* adds proof rules and axioms for the declared symbols, which we shall omit for simplicity. The translation of *HOL* to *TypedZF* is provided by the morphism *HOL-ZF* described in [19].

Example 24 (Model Theory of FOL). The first-order semantics formalized in *HOL* interprets the type o of formulas as (the elements of) the set of booleans and the logical connectives as operations on booleans. Since this is the case for other logics as well, it is convenient to have the *HOL* encoding of booleans in a separate signature, which we will call *Universes*.

This signature declares the boolean constants and axioms and adds an encoding of the usual logical connectives as operations on booleans. Apart from the propositional connectives $\top, \bot, \neg, \wedge, \vee, \Rightarrow$ it also declares the boolean quantifiers $\forall$ and $\exists$, which quantify over the elements of a given set.

The semantics of FOL can now be straightforwardly encoded in the signature FOL^{mod} which has the role of L^{mod} for first-order logic:

```
%sig FOL^mod = {
  %include HOL.
  %include Universes.

  univ : set.
  non_empty_universe : ded exists [x : elem univ] true.
}.
```

where the *non_empty_universe* axiom guarantees the existence of at least one individual. The morphism *FOL-FOL*^{mod} interprets the FOL syntax in boolean semantics in the obvious way and has the role of l^{mod}. We illustrate below the interpretation of o, ded, i and *forall*:

```
%view Base-FOL^mod : Base → FOL^mod = {
  o   := elem bool.
  ded := [F : elem bool] ded F eq 1.
}.

%view BaseFOL-FOL^mod : BaseFOL → FOL^mod = {
  %include Base-FOL^mod.
  i := elem univ.
}.

%view Forall-FOL^mod : Forall → FOL^mod = {
  %include BaseFOL-FOL^mod.
  forall := ∀.
}.

%view FOL-FOL^mod : FOL → FOL^mod = {
  %include BaseFOL-FOL^mod.
  %include Forall-FOL^mod.
  ...
}.
```

Example 25 (Model Theory of Modal Logic). The signature ML^{mod} encodes the Kripke semantics analogously to the first-order semantics, adding the symbols for the accessibility relation:

```
%sig ML^mod = {
  %include HOL.
  %include Universes.

  world : set.
  acc'  : elem (world ⟹ world ⟹ bool).
  acc   : elem world → elem world → elem bool
        = [v][w] acc' @ v @ w.        %infix acc.
  exists_world : ded exists [x : elem world] true.
}.
```

The accessibility relation is represented by acc' as a set-theoretic binary function returning a boolean, and acc is the equivalent LF function introduced for convenience. The morphism $ML\text{-}ML^{mod}$ translates modal logic syntax to Kripke semantics in the usual way. We illustrate below the translation of o, ded and $\diamond$:

```
%view Base-ML^mod : Base → ML^mod = {
  o   := elem (world ⟹ bool).
  ded := [f : elem (world ⟹ bool)]
           ded (∀[w : elem world] f @ w) eq 1.
}.

%view Necessity-ML^mod : Necessity → ML^mod = {
  %include BaseML^mod.
  ◇ := [f : elem (world ⟹ bool)]
         λ[w : elem world] ∃[w' : elem world] (w acc w') ∧ (f @ w').
}.

%view ML-ML^mod : ML → ML^mod = {
  %include Base-ML^mod.
```

%include *Necessity-ML^{mod}* .
...
}.

The type o of formulas is interpreted as the type of unary set-theoretic functions taking a world as an argument and returning a boolean. The symbol *ded* is interpreted as an LF function taking a predicate[1] f as an argument and returning a type which is nonempty if and only if f returns 1 for all possible worlds (here the boolean version of universal quantification is used). Finally, $\diamond$ is interpreted as an LF function taking a predicate f as an argument and returning a predicate which is 1 for a world w if and only if there exists a world accessible from w for which f returns 1.

Given a signature φ, a φ-sentence S, and a φ-model M, we define the satisfaction relation on φ by putting $M \models_\varphi S$ if and only if the type $ded_l\ S$ is interpreted as inhabited, i.e. there exists a t such that

$$\cdot \vdash_F t : \varphi^{mod}; M\ (ded_l\ S)$$

where ded_l denotes $l^{truth}(ded)$.

This notion of truth, albeit undecidable, is a purely syntactical one and goes along the lines of the Curry-Howard isomorphism [10, 18]. With all parts of the institution represented syntactically, we can reason about the encoded logic by purely type-theoretic means, without having to refer to the underlying semantics of LF. The proof of the satisfaction condition in the next section is an example of this.

5.3 Satisfaction Condition and Adequacy

Let $\sigma : \varphi_1 \to \varphi_2$ be a signature morphism, S a φ_1-sentence, and M a φ_2-model. Since σ is by definition identity on L^{syn}, we have

$$\varphi_2^{mod}; M\ (ded_l\ \sigma(S)) = \sigma; \varphi_2^{mod}; M(ded_l\ S)$$

Now $\sigma; \varphi_2^{mod} = \varphi_1^{mod}; \sigma^{mod}$ by definition of σ^{mod}, what implies

$$\varphi_2^{mod}; M\ (ded_l\ \sigma(S)) = \varphi_1^{mod}; \sigma^{mod}; M\ (ded_l\ S)$$

This proves that $M|_\sigma \models_{\varphi_1} S$ iff $M \models_{\varphi_2} \sigma(S)$ as desired and we have the following theorem.

Theorem 26. *The logic induced by the LF diagram $(L^{syn}, L^{mod}, l^{truth}, l^{mod})$ as described in this section is an institution.*

A natural question to ask at this point is how the encoded logic relates to the original one, which we had in mind when implementing L^{syn}, L^{mod} and other components of the encoding. This problem is referred to as *adequacy* of the encoding. One definition of adequacy is given in [32]; however, it does not address several important issues.

Firstly, it requires the existence of a comorphism with model expansion from the original logic to the encoded one. For most logics this requirement is too strong - as we have already

[1] The word 'predicate' here refers to a term of type *elem* (*world* $\Longrightarrow$ *bool*).

remarked in Section 5.2, the encoded logic will in general have less models than the original one, the class of models of the latter being restricted to those defined constructively using the axioms of ZFC. For this reason the model translation will generally not be surjective on objects.

Furthermore, the definition in [32] requires a comorphism in one direction only, which is not sufficient to guarantee adequacy. It merely provides an embedding of the original logic in the encoded one, while permitting the latter to be more expressive. We thus give the following definition of adequacy, developed in collaboration with Till Mossakowski.

Definition 27 (Adequacy)**.** Let $I = (Sig^I, Sen^I, Mod^I, \models^I)$ be an institution. Let D be an LF diagram representing a logic and $J = (Sig^J, Sen^J, Mod^J, \models^J)$ be the institution induced by D. Then we say that D is an adequate encoding of I if there exists an institution morphism (Φ, α, β) from J to I such that

- Φ is a retraction, i.e. there exists a functor $\Phi' : Sig^I \to Sig^J$ such that $\Phi'; \Phi$ is the identity functor on Sig^I.
- $\alpha_\Sigma : Sen^I(\Phi(\Sigma)) \to Sen^J(\Sigma)$ is bijective for each $\Sigma \in Sig^J$.
- $\beta_\Sigma : Mod^J(\Sigma) \to Mod^I(\Phi(\Sigma))$ is surjective on objects up to elementary equivalence for each $\Sigma \in Sig^J$.

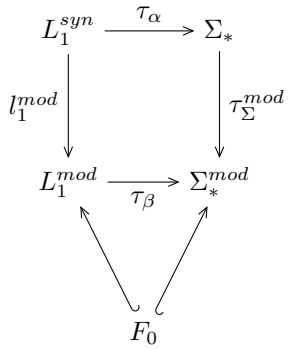
This definition in particular guarantees that there exists a comorphism from I to J with model expansion up to elementary equivalence and that borrowing is sound in both directions. Assuming the existence of suitable patterns, it is possible to show that the encodings of FOL and modal logic as given in this section are adequate. However, while the construction of the required institution morphism is fairly straightforward, proving that the morphism has model expansion up to elementary equivalence is nontrivial. Here we only present the main idea.

To prove that each model is elementary equivalent to a model representable using the axioms of ZFC, we use a canonical model construction such as e.g. the Henkin models [16] for FOL. For a given theory (Σ, Γ), we first construct a syntactic model satisfying precisely the sentences from Γ. We then show that the constructed syntactic model is representable, proving model expansion up to elementary equivalence.

In the remaining sections we will only consider logics arising from Twelf encodings and will not refer to their model-theoretic counterparts anymore.

6 Encoding Logic Translations

To encode logic translations in Twelf, we build on the work done in [32]. Let $(L_1^{syn}, L_1^{mod}, l_1^{truth}, l_1^{mod})$ and $(L_2^{syn}, L_2^{mod}, l_2^{truth}, l_2^{mod})$ be two LF diagrams representing the institutions I and J respectively. A translation from I to J will be represented as a triple $(\tau_\Sigma, \tau_\alpha, \tau_\beta)$, where $\tau_\Sigma : L_2^{syn} \hookrightarrow \Sigma_*$ is a signature in J and τ_α, τ_β are two LF morphisms such that the following diagram commutes



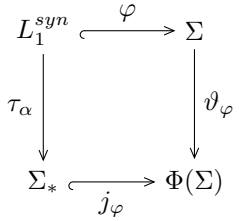
Furthermore, we require that $\tau_\alpha(ded_1)$ has the form $[f]\ ded_2\ G$ for some G, where ded_i denotes $l_i^{truth}(ded)$ for $i = 1, 2$.

The intuition is that τ_Σ declares the symbols which must be included in every translated signature, τ_α provides the translation of syntax, and τ_β provides the translation of model theory while preserving the foundation. The condition imposed on τ_α guarantees that τ_α induces a sentence translation, as will be shown later on.

We will now show how $(\tau_\Sigma, \tau_\alpha, \tau_\beta)$ induces a comorphism (Φ, α, β) from I to J.

6.1 Signature Translation

Given a signature $\varphi : L_1^{syn} \to \Sigma$ of I, we construct an LF morphism ϑ_φ by a pushout along φ and τ_α, where $\Phi(\Sigma)$ denotes the pushout:



The translated signature $\Phi(\varphi)$ is then obtained as the composition of τ_Σ with the inclusion j_φ. Let $\sigma : \varphi_1 \to \varphi_2$ be a signature morphism in I. Then we have

$$\varphi_1\ ;\ \sigma\ ;\ \vartheta_{\varphi_2} = \varphi_2\ ;\ \vartheta_{\varphi_2}$$

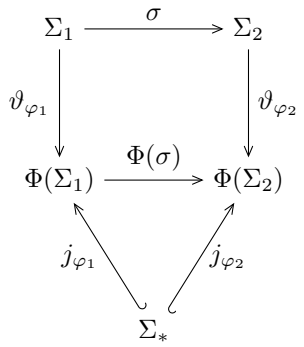
by definition of σ and

$$\varphi_2\ ;\ \vartheta_{\varphi_2} = \tau_\alpha\ ;\ j_{\varphi_2}$$

by definition of ϑ_{φ_2}. Thus

$$\varphi_1\ ;\ \sigma\ ;\ \vartheta_{\varphi_2} = \tau_\alpha\ ;\ j_{\varphi_2} \tag{2}$$

and by the universal property of the pushout $\Phi(\Sigma_1)$ there exists a unique morphism, which we define to be $\Phi(\sigma)$, such that the following LF diagram commutes



By the uniqueness property of $\Phi(\sigma)$ it follows that Φ preserves morphism identities and compositions, and hence is a functor from Sig^I to Sig^J.

This definition of Φ by means of a pushout, however, is not always applicable. In the case of the translation of many-sorted first-order logic to FOL, for example, one needs to introduce an additional axiom in $\Phi(\Sigma)$ for each translated function symbol in Σ. The resulting signature $\Phi(\Sigma)$ is not obtained by a pushout but rather via pattern-matching on the declarations of Σ. To accomodate such translations, we will again make use of patterns. For now we leave this issue as future work and only consider signature translations representable by pushouts as described above.

Example 28 (Translating Modal Logic to FOL - τ_Σ). The signature *Sig*$_*$ which has the role of Σ_* declares a binary predicate symbol representing the accessibility relation:

```
%sig Sig* = {
   %include FOL.
   rel : i → i → o.   %infix rel.
}.
```

Example 29 (Translating Modal Logic to FOL - τ_α). The syntax translation from modal logic to FOL is represented by the morphism *ML-Sig*$_*$, which translates modal logic sentences to unary first-order predicates in the usual way. The translated truth judgement then applies universal quantification.

```
%view Base-Sig* : Base → Sig* = {
   o := i → o.
   ded := [f : i → o] ded forall f.
}.

%view MPL-Sig* : MPL → Sig* = {
   %include Base-Sig*.
   ⊥ := [x : i] false.
   ⇒ := [f : i → o][g : i → o][x : i] (f x) imp (g x).
}.

%view Necessity-Sig* : Necessity → Sig* = {
   %include Base-Sig*.
   □ := [f : i → o][x : i] forall [y : i] (x rel y) imp (f y).
}.
```

```
%view Possibility-Sig* : Possibility → Sig* = {
    %include Base-Sig* .
    ◇ := [f : i → o][x : i] exists [y : i] (x rel y) and (f y).
}.

%view ML-Sig* : ML → Sig* = {
    %include MPL-Sig* .
    %include Necessity-Sig* .
    %include Possibility-Sig* .
}.
```

It is obvious that *ML-Sig*$_*$ satisfies the condition imposed on τ_α.

6.2 Sentence Translation

Let φ be a signature of I. By definition of τ_α we have that $\tau_\alpha(ded_1)$ has the form $[f]\ ded_2\ G$ for some G. By definition of l_i^{truth} we have that ded_i is a symbol in L_i^{syn}, what implies that the G above is unique.

Moreover, by definition of ϑ_φ we have that for any sentence S over φ, there exists a unique sentence S' over $\Phi(\varphi)$ such that $\vartheta_\varphi(ded_1\ S) = ded_2\ S'$. In particular, $S' = G[f/\vartheta_\varphi(S)]$. The sentence translation α_Σ is then defined by $\alpha_\Sigma(S) = S'$.

Note: We cannot simply define α_Σ by restricting ϑ_φ to Sen^I, as there is no guarantee that ϑ_φ maps the type of sentences over φ_1 to the type of sentences over φ_2. In fact, as we saw in the previous section this is indeed not the case for the translation of modal logic to FOL.

We now verify that $\alpha : Sen^I \to \Phi; Sen^J$ defined in this way is a natural transformation. Let $\sigma : \varphi_1 \to \varphi_2$ be a signature morphism in Sig^I and S be a sentence over φ_1. We have

$$\vartheta_{\varphi_2}(ded_1\ \sigma(S)) = \vartheta_{\varphi_2}(\sigma(ded_1\ S))$$

by definition of σ. Now

$$\vartheta_{\varphi_2}(\sigma(ded_1\ S)) = \Phi(\sigma)(\vartheta_{\varphi_1}(ded_1\ S))$$

by definition of $\Phi(\sigma)$. By definition of α_{φ_1} we have

$$\Phi(\sigma)(\vartheta_{\varphi_1}(ded_1\ S)) = \Phi(\sigma)(ded_2\ \alpha_{\varphi_1}(S))$$

and again by definition of $\Phi(\sigma)$

$$\Phi(\sigma)(ded_2\ \alpha_{\varphi_1}(S)) = ded_2\ \Phi(\sigma)(\alpha_{\varphi_1}(S))$$

Thus we have $\alpha_{\varphi_2}(\sigma(S)) = \Phi(\sigma)(\alpha_{\varphi_1}(S))$ as desired.

6.3 Model Translation

Let φ be a signature of I. We have

$$\tau_\Sigma\ ;\ j_\varphi\ ;\ \Phi(\varphi)^{mod} = \Phi(\varphi)\ ;\ \Phi(\varphi)^{mod}$$

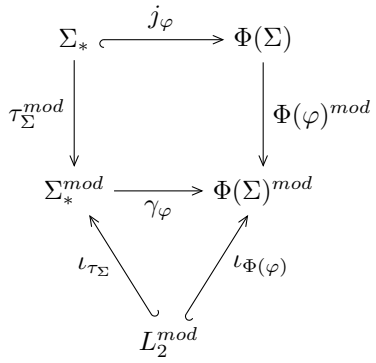
by definition of $\Phi(\varphi)$ and

$$\Phi(\varphi)\ ;\ \Phi(\varphi)^{mod} = l_2^{mod}\ ;\ \iota_{\Phi(\varphi)}$$

by definition of $\Phi(\varphi)^{mod}$. Thus

$$\tau_\Sigma\ ;\ j_\varphi\ ;\ \Phi(\varphi)^{mod} = l_2^{mod}\ ;\ \iota_{\Phi(\varphi)} \tag{3}$$

and by the universal property of the pushout Σ_*^{mod} there exists a unique morphism γ_φ which makes the following LF diagram commute



Remark 30. The computation of LF pushouts along inclusions described in Section 2 shows that γ_φ is in fact an inclusion. Hence $\Phi(\Sigma)^{mod}$ is an extension of Σ_*^{mod}, as one would expect.

Now we have

$$\varphi\ ;\ \vartheta_\varphi\ ;\ \Phi(\varphi)^{mod} = \tau_\alpha\ ;\ j_\varphi\ ;\ \Phi(\varphi)^{mod}$$

by definition of ϑ_φ and

$$\tau_\alpha\ ;\ j_\varphi\ ;\ \Phi(\varphi)^{mod} = \tau_\alpha\ ;\ \tau_\Sigma^{mod}\ ;\ \gamma_\varphi$$

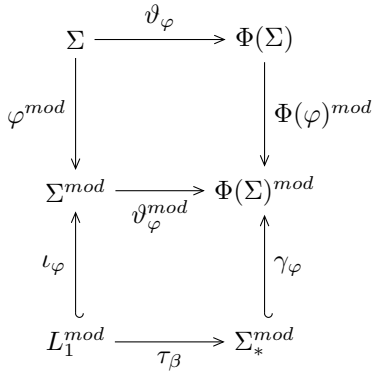
by definition of γ_φ. Furthermore,

$$\tau_\alpha\ ;\ \tau_\Sigma^{mod}\ ;\ \gamma_\varphi = l_1^{mod}\ ;\ \tau_\beta\ ;\ \gamma_\varphi$$

by definition of τ_α and τ_β. Thus

$$\varphi\ ;\ \vartheta_\varphi\ ;\ \Phi(\varphi)^{mod} = l_1^{mod}\ ;\ \tau_\beta\ ;\ \gamma_\varphi \tag{4}$$

and by the universal property of the pushout Σ^{mod} there exists a unique morphism ϑ_φ^{mod} such that the following LF diagram commutes



The model translation functor β_φ is then defined as mapping each model M of $\Phi(\varphi)$ to the model $\vartheta_\varphi^{mod}; M$ of φ.
To prove that β defined in this way is a natural transformation, let $\sigma : \varphi_1 \to \varphi_2$ be a signature morphism in Sig^I. Then we have

$$j_{\varphi_2} \ ; \ \Phi(\varphi_2)^{mod} = j_{\varphi_1} \ ; \ \Phi(\sigma) \ ; \ \Phi(\varphi_2)^{mod}$$

by definition of $\Phi(\sigma)$ and

$$j_{\varphi_1} \ ; \ \Phi(\sigma) \ ; \ \Phi(\varphi_2)^{mod} = j_{\varphi_1} \ ; \ \Phi(\varphi_1)^{mod} \ ; \ \Phi(\sigma)^{mod}$$

by definition of $\Phi(\sigma)^{mod}$. Furthermore,

$$j_{\varphi_1} \ ; \ \Phi(\varphi_1)^{mod} \ ; \ \Phi(\sigma)^{mod} = \tau_\Sigma^{mod} \ ; \ \gamma_{\varphi_1} \ ; \ \Phi(\sigma)^{mod}$$

by definition of γ_{φ_1}. Thus

$$j_{\varphi_2} \ ; \ \Phi(\varphi_2)^{mod} = \tau_\Sigma^{mod} \ ; \ \gamma_{\varphi_1} \ ; \ \Phi(\sigma)^{mod} \tag{5}$$

Likewise, we have

$$\iota_{\tau_\Sigma} \ ; \ \gamma_{\varphi_1} \ ; \ \Phi(\sigma)^{mod} = \iota_{\Phi(\varphi_1)} \ ; \ \Phi(\sigma)^{mod}$$

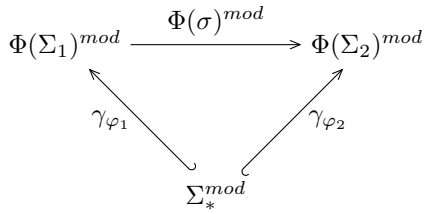
by definition of γ_{φ_1} and

$$\iota_{\Phi(\varphi_1)} \ ; \ \Phi(\sigma)^{mod} = \iota_{\Phi(\varphi_1)}$$

by definition of $\Phi(\sigma)^{mod}$. Thus

$$\iota_{\tau_\Sigma} \ ; \ \gamma_{\varphi_1} \ ; \ \Phi(\sigma)^{mod} = \iota_{\Phi(\varphi_1)} \tag{6}$$

By uniqueness of γ_{φ_2} the equalities (5) and (6) imply that the following LF diagram commutes



Now we have

$$\varphi_1^{mod} \; ; \; \sigma^{mod} \; ; \; \vartheta_{\varphi_2}^{mod} = \sigma \; ; \; \varphi_2^{mod} \; ; \; \vartheta_{\varphi_2}^{mod}$$

by definition of σ^{mod} and

$$\sigma \; ; \; \varphi_2^{mod} \; ; \; \vartheta_{\varphi_2}^{mod} = \sigma \; ; \; \vartheta_{\varphi_2} \; ; \; \Phi(\varphi_2)^{mod}$$

by definition of $\vartheta_{\varphi_2}^{mod}$. Furthermore,

$$\sigma \; ; \; \vartheta_{\varphi_2} \; ; \; \Phi(\varphi_2)^{mod} = \vartheta_{\varphi_1} \; ; \; \Phi(\sigma) \; ; \; \Phi(\varphi_2)^{mod}$$

by definition of $\Phi(\sigma)$ and

$$\vartheta_{\varphi_1} \; ; \; \Phi(\sigma) \; ; \; \Phi(\varphi_2)^{mod} = \vartheta_{\varphi_1} \; ; \; \Phi(\varphi_1)^{mod} \; ; \; \Phi(\sigma)^{mod}$$

by definition of $\Phi(\sigma)^{mod}$. Finally,

$$\vartheta_{\varphi_1} \; ; \; \Phi(\varphi_1)^{mod} \; ; \; \Phi(\sigma)^{mod} = \varphi_1^{mod} \; ; \; \vartheta_{\varphi_1}^{mod} \; ; \; \Phi(\sigma)^{mod}$$

by definition of $\vartheta_{\varphi_1}^{mod}$. Thus

$$\varphi_1^{mod} \; ; \; \sigma^{mod} \; ; \; \vartheta_{\varphi_2}^{mod} = \varphi_1^{mod} \; ; \; \vartheta_{\varphi_1}^{mod} \; ; \; \Phi(\sigma)^{mod} \tag{7}$$

Likewise,

$$\iota_{\varphi_1} \; ; \; \sigma^{mod} \; ; \; \vartheta_{\varphi_2}^{mod} = \iota_{\varphi_2} \; ; \; \vartheta_{\varphi_2}^{mod}$$

by definition of σ^{mod} and

$$\iota_{\varphi_2} \; ; \; \vartheta_{\varphi_2}^{mod} = \tau_\beta \; ; \; \gamma_{\varphi_2}$$

by definition of $\vartheta_{\varphi_2}^{mod}$. Furthermore,

$$\tau_\beta \; ; \; \gamma_{\varphi_2} = \tau_\beta \; ; \; \gamma_{\varphi_1} \; ; \; \Phi(\sigma)^{mod}$$

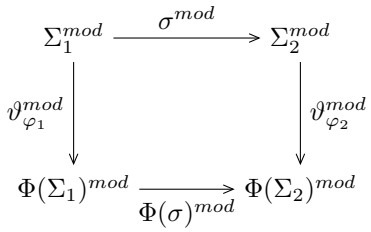
as shown before and

$$\tau_\beta \; ; \; \gamma_{\varphi_1} \; ; \; \Phi(\sigma)^{mod} = \iota_{\varphi_1} \; ; \; \vartheta_{\varphi_1}^{mod} \; ; \; \Phi(\sigma)^{mod}$$

by definition of $\vartheta_{\varphi_1}^{mod}$. Thus

$$\iota_{\varphi_1} \; ; \; \sigma^{mod} \; ; \; \vartheta_{\varphi_2}^{mod} = \iota_{\varphi_1} \; ; \; \vartheta_{\varphi_1}^{mod} \; ; \; \Phi(\sigma)^{mod} \tag{8}$$

By uniqueness of the universal morphism out of the pushout Σ_1^{mod}, the equalities (7) and (8) imply that the following LF diagram commutes



This shows that for any model M of $\Phi(\varphi_2)$, we have $\beta_{\varphi_2}(M)|_\sigma = \beta_{\varphi_1}(M|_{\Phi(\sigma)})$ as desired.

Example 31 (Translating Modal Logic to FOL - τ_β). The following signature has the role of Σ_*^{mod}:

```
%sig Sig_*^mod = {
    %include FOL^mod .
    rel : elem univ → elem univ → elem bool .
} .
```

The following morphism has the role of τ_Σ^{mod}:

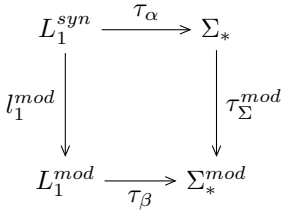
```
%view Sig_*-Sig_*^mod : Sig_* → Sig_*^mod = {
    %include FOL-FOL^mod .
    rel := rel .
} .
```

The model translation is then encoded straightforwardly as the following morphism:

```
%view ML^mod-Sig_*^mod : ML^mod → Sig_*^mod = {
    world := univ .
    acc'  := λ[x] λ[y] x rel y .
    exists_world := non_empty_universe .
} .
```

We notice, however, that the commutativity requirement fails: the morphism $l_1^{mod};\tau_\beta$ interprets the type o of formulas as the set-theoretic function space *elem* (*univ* $\Longrightarrow$ *bool*) whereas the morphism $\tau_\alpha;\tau_\Sigma^{mod}$ interprets o as the LF function space *elem univ* $\rightarrow$ *elem bool*. This problem could be avoided if we changed the encoding of modal logic semantics to use the LF function space instead of the set-theoretic one. However, one of the aims of the encodings is to formalize as much of the semantics of logics in the foundation as possible, in order to facilitate reasoning about models. We will thus take a different approach, as explained below.

The above example shows that for some translations the commutativity of the following diagram fails:



In order to adapt our definition to accomodate such translations, we will again need patterns. Once they are implemented for Twelf, the commutativity requirement for the diagram above will be dropped. In the absence of commutativity, however, the morphism ϑ_φ^{mod} can no longer be formed by a pushout. Instead, the morphism τ_β will contain patterns giving its extension to ϑ_φ^{mod} for a specific signature φ in I. The model translation β_φ will then be formed by a composition with ϑ_φ^{mod}, as done currently.

The naturality of the resulting model translation is not guaranteed in the absence of commutativity. The verification of the naturality of a translation specified by patterns is currently an open problem, which is outside the scope of our work. For our purposes we shall assume that the translation β induced by patterns given in τ_β is a natural transformation.

Example 32 (Translation of Modal Logic to FOL using Patterns). We will first extend the signature *ML*, which encodes the syntax of modal logic, by patterns specifying that the only permitted declarations of new symbols are those of the form $p : o$ and the only permitted signature morphisms are those mapping symbols to symbols. We then extend the morphism τ_β by a pattern indicating that any symbol p of Σ^{mod} which is not in ML^{mod} should be translated to λp. It is possible to show the resulting model translation will be a natural transformation.

6.4 Satisfaction Condition

Let φ be a signature in I, S be a sentence over φ, and M be a model over $\Phi(\varphi)$. Then we have

$$\varphi^{mod}; \beta_\varphi(M)\ (ded_1\ S) = \varphi^{mod}; \vartheta_\varphi^{mod}; M\ (ded_1\ S)$$

by definition of β_φ and

$$\Phi(\varphi)^{mod}; M\ (ded_2\ \alpha_\varphi(S)) = \vartheta_\varphi; \Phi(\varphi)^{mod}; M\ (ded_1\ S)$$

by definition of α_φ. Now $\varphi^{mod}; \vartheta_\varphi^{mod} = \vartheta_\varphi; \Phi(\varphi)^{mod}$ by definition of ϑ_φ^{mod}, what implies

$$\varphi^{mod}; \beta_\varphi(M)\ (ded_1\ S) = \Phi(\varphi)^{mod}; M\ (ded_2\ \alpha_\varphi(S))$$

This proves that $M \models^J_{\Phi(\varphi)} \alpha_\varphi(S)$ iff $\beta_\varphi(M) \models^I_\varphi S$ as desired and we have the following theorem.

Theorem 33. *The translation from I to J induced by $(\tau_\Sigma, \tau_\alpha, \tau_\beta)$ as described in this section is an institution comorphism.*

The above proof, however, does not apply to translations which fail to meet the commutativity requirement, such as the translation of modal to first-order logic we gave in this section. To prove the satisfaction condition for these translations, we will instead use the concept of logical relations introduced in Section 4. We have the following theorem.

Theorem 34. *Assume there exists a logical relation $\rho : l_1^{mod}; \tau_\beta \Rightarrow \tau_\alpha; \tau_\Sigma^{mod}$ and a family of logical relations $\{\rho_\varphi\}$, where each $\rho_\varphi : \varphi^{mod}; \vartheta_\varphi^{mod} \Rightarrow \vartheta_\varphi; \Phi(\varphi)^{mod}$ is equal to ρ when restricted to L_1^{syn}. Furthermore, assume there exists a transition pair (Ψ, Ξ) under ρ for the type family ded_1. Then the translation from I to J induced by $(\tau_\Sigma, \tau_\alpha, \tau_\beta)$ is an institution comorphism.*

Proof. Let φ be a signature in I, S be a sentence over φ, and M be a model of $\Phi(\varphi)$. Denote the morphism $l_1^{mod}; \tau_\beta$ by U and the morphism $\tau_\alpha; \tau_\Sigma^{mod}$ by V. Likewise, denote the morphism $\varphi^{mod}; \vartheta_\varphi^{mod}$ by U_φ and the morphism $\vartheta_\varphi; \Phi(\varphi)^{mod}$ by V_φ.

By definition ρ_φ agrees with ρ on L_1^{syn}. Furthermore, U_φ agrees with U on L_1^{syn} since l_1^{mod} is a restriction of φ^{mod} to L_1^{syn} and τ_β is a restriction of ϑ_φ^{mod} to L_1^{mod}. Likewise, V_φ agrees with V on L_1^{syn} since τ_α is a restriction of ϑ_φ to L_1^{syn} and τ_Σ^{mod} is a restriction of $\Phi(\varphi)^{mod}$ to Σ_*.

This implies that $(M(\Psi), M(\Xi))$ is a transition pair for the logical relation $\rho; M$. By Lemma 20 this implies that the type $U_\varphi; M\ (ded_1\ S)$ is inhabited if and only if the type $V_\varphi; M\ (ded_1\ S)$ is. This proves the satisfaction condition. □

The Twelf module system has recently been extended by Florian Rabe and Carsten Schürmann to include logical relations. The modular syntax for logical relations is

$$\begin{array}{llll} \textit{Toplevel} & G & := & \ldots \mid \%\texttt{rel}\ R : \mu \to \mu = \{\rho\} \\ \textit{Relations} & \rho & := & \cdot \mid \rho,\ \%\texttt{include}\ R \mid \rho,\ c := E \end{array}$$

where R stands for a logical relation name.

Example 35 (Translating Modal Logic to FOL - Satisfaction Condition). The logical relation SC_{ML}, which has the role of ρ, is specified in a modular way with the obvious semantics:

```
%rel SC_Base : Base-ML^mod ML^mod-Sig*^mod → Base-Sig* Sig*-Sig*^mod = {
  o   := [f : elem (univ ⟹ bool)]
         [g : elem univ → elem bool]
         ded f eq (λ g).

  ded := [f : elem (univ ⟹ bool)]
         [g : elem univ → elem bool]
         [p : ded f eq (λ g)]
         [r : ded (∀[w] f @ w) eq 1]
         [s : ded (∀ g) eq 1]
         ded true.
}.

%rel SC_MPL : MPL-ML^mod ML^mod-Sig*^mod → MPL-Sig* Sig*-Sig*^mod = {
  %include SC_Base.

  ⊥   := refl.

  ⇒   := [f1 : elem (univ ⟹ bool)]
         [g1 : elem univ → elem bool]
         [p1 : ded f1 eq (λ g1)]
         [f2 : elem (univ ⟹ bool)]
         [g2 : elem univ → elem bool]
         [p2 : ded f2 eq (λ g2)]
         fun_ext λ [w] congF2
            (trans (congF p1 ([h] h @ w)) beta)
            (trans (congF p2 ([h] h @ w)) beta)
            [x][y] x ⇒ y.
}.
```

```
%rel SC_Nec : Necessity-ML^mod ML^mod-Sig_*^mod → Necessity-Sig_* Sig_*-Sig_*^mod = {
  %include SC_Base .

  □    := [f : elem (univ ⟹ bool)]
          [g : elem univ → elem bool]
          [p : ded f eq (λ g)]
          fun_ext λ [x] fun_ext ∀ [y] congF2
             (trans (congF beta ([h] h @ y)) beta)
             (trans (congF p ([h] h @ y)) beta)
             [x][y] x ⇒ y.
}.

%rel SC_Pos : Possibility-ML^mod ML^mod-Sig_*^mod → Possibility-Sig_* Sig_*-Sig_*^mod = {
  %include SC_Base .

  ◇    := [f : elem (univ ⟹ bool)]
          [g : elem univ → elem bool]
          [p : ded f eq (λ g)]
          fun_ext λ [x] fun_ext ∃ [y] congF2
             (trans (congF beta ([h] h @ y)) beta)
             (trans (congF p ([h] h @ y)) beta)
             [x][y] x ∧ y.
}.

%rel SC_ML : ML-ML^mod ML^mod-Sig_*^mod → ML-Sig_* Sig_*-Sig_*^mod = {
  %include SC_MPL .
  %include SC_Nec .
  %include SC_Pos .
}.
```

The relation between the set-theoretic and the LF function space is expressed in the case for o. The case for *ded* describes a proof irrelevance condition. The other cases prove that all terms respect the logical relation.

The logical relation ρ_φ extends ρ by associating with each symbol p not in ML the proof term *refl*. Finally, the terms Ψ and Ξ are given as follows:

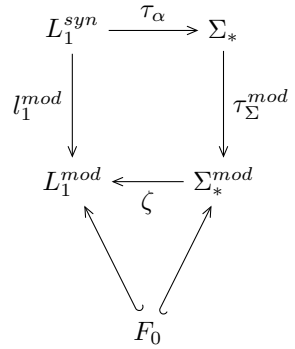
```
%sig Satisfaction = {
  %include Sig_*^mod .

  Ψ    : {f : elem (univ ⟹ bool)}
         {g : elem univ → elem bool}
         {p : ded f eq (λ g)}
         ded (∀[w] f @ w) eq 1 → ded (∀ g) eq 1
       = [f][g][p][r] forall1I (forallI [w] trans
            (sym (trans (congF p ([h] h @ w)) beta))
            (forallE (forall1E r) w)).

  Ξ    : {f : elem (world ⟹ bool)}
         {g : elem univ → elem bool}
         {p : ded f eq (λ g)}
         ded (∀ g) eq 1 → ded (∀[w] f @ w) eq 1
       = [f][g][p][s] forall1I (forallI [w] trans
            (trans (congF p ([h] h @ w)) beta)
            (forallE (forall1E s) w)).
}.
```

6.5 Model Expansion

Proof of the model expansion property is given by providing a retraction ζ of τ_β which makes the following LF diagram commute



In other words, $\tau_\beta; \zeta$ must be identity on L_1^{mod}.

To prove that such ζ indeed guarantees model expansion, let φ be a signature in I. Then we have

$$\varphi \ ; \ \varphi^{mod} = l_1^{mod} \ ; \ \iota_\varphi$$

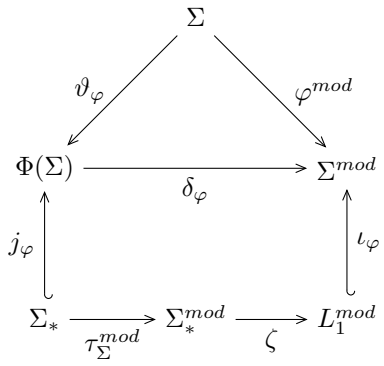
by definition of φ^{mod} and

$$l_1^{mod} \ ; \ \iota_\varphi = \tau_\alpha \ ; \ \tau_\Sigma^{mod} \ ; \ \zeta \ ; \ \iota_\varphi$$

by definition of ζ. Thus

$$\varphi \ ; \ \varphi^{mod} = \tau_\alpha \ ; \ \tau_\Sigma^{mod} \ ; \ \zeta \ ; \ \iota_\varphi \tag{9}$$

and by the universal property of the pushout $\Phi(\Sigma)$ there exists a unique morphism δ_φ such that the following LF diagram commutes



Furthermore, we have

$$\Phi(\varphi)\ ;\ \delta_\varphi = \tau_\Sigma\ ;\ j_\varphi\ ;\ \delta_\varphi$$

by definition of $\Phi(\varphi)$ and

$$\tau_\Sigma\ ;\ j_\varphi\ ;\ \delta_\varphi = \tau_\Sigma\ ;\ \tau_\Sigma^{mod}\ ;\ \zeta\ ;\ \iota_\varphi$$

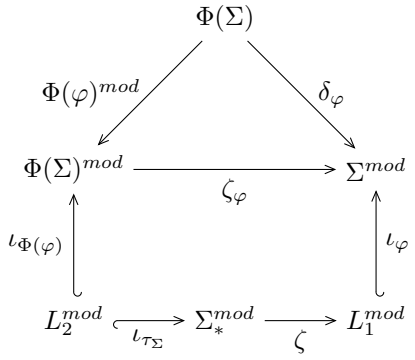
by definition of δ_φ. Moreover,

$$\tau_\Sigma\ ;\ \tau_\Sigma^{mod}\ ;\ \zeta\ ;\ \iota_\varphi = l_2^{mod}\ ;\ \iota_{\tau_\Sigma}\ ;\ \zeta\ ;\ \iota_\varphi$$

by definition of τ_Σ^{mod}. Thus

$$\Phi(\varphi)\ ;\ \delta_\varphi = l_2^{mod}\ ;\ \iota_{\tau_\Sigma}\ ;\ \zeta\ ;\ \iota_\varphi \tag{10}$$

and by the universal property of the pushout $\Phi(\Sigma)^{mod}$ there exists a unique morphism ζ_φ such that the following LF diagram commutes



Now we show that ζ_φ is a retraction of ϑ_φ. We have

$$\tau_\Sigma^{mod}\ ;\ \gamma_\varphi\ ;\ \zeta_\varphi = j_\varphi\ ;\ \Phi(\varphi)^{mod}\ ;\ \zeta_\varphi$$

by definition of γ_φ and

$$j_\varphi\ ;\ \Phi(\varphi)^{mod}\ ;\ \zeta_\varphi = j_\varphi\ ;\ \delta_\varphi$$

by definition of δ_φ. Now

$$j_\varphi\ ;\ \delta_\varphi = \tau_\Sigma^{mod}\ ;\ \zeta\ ;\ \iota_\varphi$$

again by definition of δ_φ, what shows that

$$\tau_\Sigma^{mod}\ ;\ \gamma_\varphi\ ;\ \zeta_\varphi = \tau_\Sigma^{mod}\ ;\ \zeta\ ;\ \iota_\varphi \tag{11}$$

Likewise, we have

$$\iota_{\tau_\Sigma} \; ; \; \gamma_\varphi \; ; \; \zeta_\varphi = \iota_{\Phi(\varphi)} \; ; \; \zeta_\varphi$$

by definition of γ_φ and

$$\iota_{\Phi(\varphi)} \; ; \; \zeta_\varphi = \iota_{\tau_\Sigma} \; ; \; \zeta \; ; \; \iota_\varphi$$

by definition of ζ_φ. Thus

$$\iota_{\tau_\Sigma} \; ; \; \gamma_\varphi \; ; \; \zeta_\varphi = \iota_{\tau_\Sigma} \; ; \; \zeta \; ; \; \iota_\varphi \tag{12}$$

By uniqueness of the universal morphism out of the pushout Σ_*^{mod}, equalities (11) and (12) imply that the following LF diagram commutes

$$\begin{array}{ccc} \Sigma_*^{mod} & \xrightarrow{\gamma_\varphi} & \Phi(\Sigma)^{mod} \\ \downarrow{\scriptstyle \zeta} & & \downarrow{\scriptstyle \zeta_\varphi} \\ L_1^{mod} & \xrightarrow{\iota_\varphi} & \Sigma^{mod} \end{array}$$

Now we have

$$\varphi^{mod} = \vartheta_\varphi \; ; \; \delta_\varphi$$

by definition of δ_φ. Furthermore,

$$\vartheta_\varphi \; ; \; \delta_\varphi = \vartheta_\varphi \; ; \; \Phi(\varphi)^{mod} \; ; \; \zeta_\varphi$$

by definition of ζ_φ and

$$\vartheta_\varphi \; ; \; \Phi(\varphi)^{mod} \; ; \; \zeta_\varphi = \varphi^{mod} \; ; \; \vartheta_\varphi^{mod} \; ; \zeta_\varphi$$

by definition of ϑ_φ^{mod}. This shows that

$$\varphi^{mod} = \varphi^{mod} \; ; \; \vartheta_\varphi^{mod} \; ; \; \zeta_\varphi \tag{13}$$

Likewise, we have

$$\iota_\varphi = \tau_\beta \; ; \; \zeta \; ; \; \iota_\varphi$$

by definition of ζ and

$$\tau_\beta \; ; \; \zeta \; ; \; \iota_\varphi = \tau_\beta \; ; \; \gamma_\varphi \; ; \; \zeta_\varphi$$

by what we have just shown. Furthermore,

$$\tau_\beta \ ; \ \gamma_\varphi \ ; \ \zeta_\varphi = \iota_\varphi \ ; \ \vartheta_\varphi^{mod} \ ; \ \zeta_\varphi$$

by definition of ϑ_φ^{mod}, what shows

$$\iota_\varphi = \iota_\varphi \ ; \ \vartheta_\varphi^{mod} \ ; \ \zeta_\varphi \tag{14}$$

By uniqueness of the universal morphism out of the pushout Σ^{mod}, equalities (13) and (14) imply that $\vartheta_\varphi^{mod}; \zeta_\varphi$ is an identity on Σ^{mod} as desired. This means that for any model M of φ, $M' = \zeta_\varphi; M$ is a model of $\Phi(\varphi)$ for which we have $M = \vartheta_\varphi^{mod}; M'$. This proves model expansion and gives us the following theorem.[2]

Theorem 36. *If there exists an LF morphism ζ as described in this section, then the comorphism from I to J induced by $(\tau_\Sigma, \tau_\alpha, \tau_\beta)$ has the model expansion property.*

As was the case with the satisfaction condition, the above proof does not apply to translations which fail to meet the commutativity requirement. To prove the model expansion property for these translations, we will again make use of patterns and logical relations. We assume that the morphism ζ includes patterns which give its extension to ζ_φ for a specific signature φ in I. We have the following theorem.

Theorem 37. *Assume there exists a logical relation $\rho : l_1^{mod} \Rightarrow l_1^{mod}; \tau_\beta; \zeta$ and a family of logical relations $\{\rho_\varphi\}$, where each $\rho_\varphi : \varphi^{mod} \Rightarrow \varphi^{mod}; \vartheta_\varphi^{mod}; \zeta_\varphi$ is equal to ρ when restricted to L_1^{syn}. Furthermore, assume there exists a transition pair (Ψ, Ξ) under ρ for the type family ded_1. Then the comorphism from I to J induced by $(\tau_\Sigma, \tau_\alpha, \tau_\beta)$ has model expansion up to elementary equivalence.*

Proof. Let φ be a signature in I and M be a model of φ. Analogously to the proof of Theorem 34 we can show that the type $\varphi^{mod}; M \ (ded_1 \ S)$ is inhabited if and only if the type $\varphi^{mod}; \vartheta_\varphi^{mod}; \zeta_\varphi; M \ (ded_1 \ S)$ is. This means that the models M and $\vartheta_\varphi^{mod}; \zeta_\varphi; M$ of φ are elementary equivalent. Since the latter clearly lies in the image of β_φ, this proves model expansion up to elementary equivalence. □

Example 38 (Translating Modal Logic to FOL - Model Expansion). The morphism Sig_*^{mod}-ML^{mod} which has the role of ζ is curently given as follows:

```
%view FOL^mod-ML^mod : FOL^mod → ML^mod = {
  univ := world.
  non_empty_universe := exists_world.
}.

%view Sig_*^mod-ML^mod : Sig_*^mod → ML^mod = {
  %include FOL^mod-ML^mod.
  rel := acc.
}.
```

[2] For many translations the model M' cannot be obtained by simply reducing along a fixed morphism. Proving model expansion for such translations requires the existence of functors for LF, which are currently being implemented by F. Horozal and F. Rabe.

Once patterns are implemented, ζ will be extended by a pattern specifying that each symbol p not in ML be mapped to the term $[w]\, p \mathbin{@} w$.
The logical relation ME_{ML} which has the role of ρ is specified as follows:

```
%rel ME_Base : Base-ML^mod → Base-ML^mod ML^mod-Sig*^mod Sig*-Sig*^mod = {
  o   := [f : elem (world ⟹ bool)]
         [g : elem (world ⟹ bool)]
         ded f eq g.

  ded := [f : elem (world ⟹ bool)]
         [g : elem (world ⟹ bool)]
         [p : ded f eq g]
         [r : ded (∀[w] f @ w) eq 1]
         [s : ded (∀[w] g @ w) eq 1]
         ded true.
}.

%rel ME_MPL : MPL-ML^mod → MPL-ML^mod ML^mod-Sig*^mod Sig*-Sig*^mod = {
  %include ME_Base.

  ⊥   := refl.

  ⇒   := [f1 : elem (world ⟹ bool)]
         [g1 : elem (world ⟹ bool)]
         [p1 : ded f1 eq g1]
         [f2 : elem (world ⟹ bool)]
         [g2 : elem (world ⟹ bool)]
         [p2 : ded f2 eq g2]
         fun_ext λ [w] congF2
            (congF p1 ([h] h @ w))
            (congF p2 ([h] h @ w))
            [x][y] x ⇒ y.
}.

%rel ME_Nec : Necessity-ML^mod → Necessity-ML^mod ML^mod-Sig*^mod Sig*-Sig*^mod = {
  %include ME_Base.

  □   := [f : elem (world ⟹ bool)]
         [g : elem (world ⟹ bool)]
         [p : ded f eq g]
         fun_ext λ [x] fun_ext ∀ [y] congF2
            (sym (trans (congF beta ([h] h @ y)) beta))
            (congF p ([h] h @ y))
            [x][y] x ⇒ y.
}.

%rel ME_Pos : Possibility-ML^mod → Possibility-ML^mod ML^mod-Sig*^mod Sig*-Sig*^mod =
{ %include ME_Base.

  ◇   := [f : elem (world ⟹ bool)]
         [g : elem (world ⟹ bool)]
         [p : ded f eq g]
         fun_ext λ [x] fun_ext ∃ [y] congF2
            (sym (trans (congF beta ([h] h @ y)) beta))
            (congF p ([h] h @ y))
```

```
            [x][y] x ∧ y.
}.

%rel ML_ML : ME-ML^mod → ML-ML^mod ML^mod-Sig_*^mod Sig_*-Sig_*^mod = {
  %include ME_MPL.
  %include ME_Nec.
  %include ME_Pos.
}.
```

The logical relation ρ_φ extends ρ by associating with each symbol p not in ML the proof term *eta*. Finally, the terms Ψ and Ξ are given as follows:

```
%sig ModelExpansion = {
  %include ML^mod.

  Ψ    : {f : elem (world ⟹ bool)}
         {g : elem (world ⟹ bool)}
         {p : ded f eq g}
         ded (∀[w] f @ w) eq 1 → ded (∀[w] g @ w) eq 1
       = [f][g][p][r] forall1I (forallI [w] trans
           (sym (congF p ([h] h @ w)))
           (forallE (forall1E r) w)).

  Ξ    : {f : elem (world ⟹ bool)}
         {g : elem (world ⟹ bool)}
         {p : ded f eq g}
         ded (∀[w] f @ w) eq 1 → ded (∀[w] g @ w) eq 1
       = Ψ.
}.
```

7 Conclusion and Future Work

Building on the work done in [32], we have shown how to use the Twelf module system to encode logic translations and proved that an encoding indeed yields an institution comorphism. Furthermore, we have shown how the proof of model expansion can be represented in Twelf. We have given a new criterion for the adequacy of logic encodings, improving upon some of the shortcomings of the definition given in [32]. The translation of modal to first-order logic comprises about 180 lines of Twelf code and is available as [34].

The mechanical verification of the soundness of the encoded translation is for the most part provided by the Twelf type checker, which automatically verifies the well-formedness of all signatures and morphisms involved. An exception is the commutativity check - due to the way the structuring mechanisms are handled in Twelf, we currently do not have a simple way of verifying commutativity resp. equality of morphisms.

At the moment the burden of proving commutativity is placed on the user, which is not ideal as even for simple translations the verification of commutativity requires some work. It would be desirable to have a Twelf syntax such as

```
%equal μ1 μ2.
```

available for verifying equality, where μ_1, μ_2 denote morphism composites. The above would type-check if and only if the morphisms μ_1, μ_2 are equal.

The implementation of such an equality check is not as straightforward as it may seem - checking that μ_1 and μ_2 agree on every term in the domain signature can get computationally expensive. On the other hand, comparing the modular structure of μ_1 and μ_2 and utilizing the known equalities enforced by the module system is efficient but not complete - μ_1 and μ_2 may be equal while having a very different modular structure. The two approaches could possibly be merged, with the componentwise check applied only in cases when the comparison of modular structures fails to produce a match.

Translations which fail to meet the commutativity requirement currently cannot be encoded in our framework. To remedy this problem, we need to extend Twelf with declarative patterns, which would allow us to specify families of signatures, respectively morphisms, by pattern matching. The implementation of patterns for Twelf has been the subject of ongoing work of Fulya Horozal and Florian Rabe.

In order to verify the soundness of non-commuting translations, we have developed a theory of logical relations for Twelf. We have shown how the soundness of the translation and borrowing can be established by supplying a suitable logical relation in place of the equality check. The implementation of logical relations in Twelf has already been done by Florian Rabe and Carsten Schürmann. Once the patterns are implemented, the well-formedness of the relation and hence the soundness of the encoding will be automatically verified by the Twelf type-checker.

References

[1] H. Barendregt. Lambda Calculi with Types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.

[2] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a Mathematical Assistant. In W. McCune, editor, *Conference on Automated Deduction*, volume 1249 of *LNCS*, pages 252–255. Springer, 1997.

[3] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. LEO-II - A Cooperative Automatic Theorem Prover for Higher-Order Logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *International Joint Conference on Automated Reasoning*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.

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

[5] C. Brown. Combining Type Theory and Untyped Set Theory. In N. Shankar and U. Furbach, editor, *International Joint Conference on Automated Reasoning*, volume 4130 of *LNAI*, pages 205–219. Springer, 2006.

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

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

[8] K. Claessen and N. Sörensson. New Techniques that Improve MACE-style Finite Model Finding. In *Conference on Automated Deduction: Workshop on Model Computation - Principles, Algorithms, Applications*, 2003.

[9] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, 1998.

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

[11] R. Diaconescu. *Institution-independent Model Theory*. Springer, 2008.

[12] 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.

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

[14] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[15] R. Harper, D. Sannella, and A. Tarlecki. Structured Theory Presentations and Logic Representations. *Annals of Pure and Applied Logic*, 67:113–160, 1994.

[16] L. Henkin. The Completeness of the First-Order Functional Calculus. *Journal of Symbolic Logic*, 14(3):159–166, 1949.

[17] F. Honsell and D. Sannella. Prelogical Relations. *Information and Computation*, 178(1):23–43, 2002.

[18] 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.

[19] M. Iancu and F. Rabe. Formalizing Foundations of Mathematics, 2010. `http://kwarc.info/frabe/`.

[20] B. Jacobs and T. Melham. Translating Dependent Type Theory into Higher Order Logic. In M. Bezem and J. Groote, editor, *Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, pages 209–229. Springer, 1993.

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

[22] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1998.

[23] N. Martí-Oliet and J. Meseguer. General Logics and Logical Frameworks. In D. Gabbay, editor, *What Is a Logical System?*, pages 335–392. Oxford University Press, 1994.

[24] P. Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In E. Rose and J. Sheperdson, editors, *Logic Colloquium 1973*, pages 73–118. North-Holland, 1975.

[25] J. Mitchell. Type Systems for Programming Languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, pages 365–458. North-Holland, 1990.

[26] T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 519–522. Springer, 2007.

[27] L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer, 1994.

[28] F. Pfenning and C. Schürmann. System Description: Twelf - A Meta-Logical Framework for Deductive Systems. In *Conference on Automated Deduction*, volume 1632 of *LNCS*, pages 202–206. Springer, 1999.

[29] G. Plotkin. Lambda-Definability and Logical Relations. Memorandum SAI–RM–4, University of Edinburgh, 1973.

[30] G. Plotkin, J. Power, D. Sannella, and R. Tennent. Lax Logical Relations. In *Colloquium on Automata, Languages, and Programming*, volume 1853 of *LNCS*, pages 85–102. Springer, 2000.

[31] A. Poswolsky and C. Schürmann. System Description: Delphin - A Functional Programming Language for Deductive Systems. In A. Abel and C. Urban, editor, *Logical Frameworks and Metalanguages: Theory and Practice*, pages 135–141, 2008.

[32] F. Rabe. A Logical Framework Combining Model and Proof Theory, 2010. `http://kwarc.info/frabe/`.

[33] F. Rabe and C. Schürmann. A Practical Module System for LF. In J. Cheney and A. Felty, editor, *Logical Frameworks and Metalanguages: Theory and Practice*, pages 40–48. ACM Press, 2009.

[34] F. Rabe and K. Sojakova. The Twelf Encoding of the Translation of Modal to First-Order Logic, 2010. `https://svn.kwarc.info/repos/twelf/translations/ml-fol`.

[35] J. Reynolds. On the Relation between Direct and Continuation Semantics. In *Colloquium on Automata, Languages and Programming*, volume 14 of *LNCS*, pages 141–156. Springer, 1974.

[36] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *Artificial Intelligence Communications*, 15:91–110, 2002.

[37] O. Schoett. *Data Abstraction and the Correctness of Modular Programming*. PhD thesis, Department of Computer Science, University of Edinburgh, 1987.

[38] C. Schürmann and J. Sarnat. Structural Logical Relations. In F. Pfenning, editor, *Logic in Computer Science*, pages 69–80. IEEE Computer Society, 2008.

[39] A. Tarlecki. Moving Between Logical Systems. In *Recent Trends in Data Type Specification*, volume 1130 of *LNCS*, pages 478–502. Springer, 1998.

[40] A. Trybulec and H. Blair. Computer Assisted Reasoning with Mizar. In *International Joint Conference on Artificial Intelligence*, pages 26–28. Morgan Kaufmann Publishers, 1985.

[41] J. Urban. Translating Mizar for First-Order Theorem Provers. In *Mathematical Knowledge Management*, volume 2594 of *LNCS*, pages 203–215. Springer, 2003.

[42] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topic. SPASS Version 2.0. In A. Voronkov, editor, *Conference on Automated Deduction*, volume 2392 of *LNCS*, pages 275–279. Springer, 2002.