Journal of Logic and Computation Advance Access published December 14, 2014

How to identify, translate and combine logics?

FLORIAN RABE, *Computer Science, Jacobs University, Bremen, Germany.*
Email: florian.rabe@gmail.com

Abstract

We give general definitions of logical frameworks and logics. Examples include the logical frameworks LF and Isabelle and the logics represented in them. We apply this to give general definitions for equivalence of logics, translation between logics and combination of logics. We also establish general criteria for the soundness and completeness of these. Our key messages are that the syntax and proof systems of logics are theories; that both semantics and translations are theory morphisms; and that combinations are colimits. Our approach is based on the MMT language, which lets us combine formalist declarative representations (and thus the associated tool support) with abstract categorical conceptualizations.

Keywords: MMT, logical framework, universal logic, translation, combination, semantics, soundness, completeness.

1 Introduction

Universal logic is the field of logic that investigates the common features of logics. Even though the field has arguably existed for decades, no single conceptualization has become dominant. In fact, some of the most fundamental questions (quoted in this article's title) have served as contest problems in the series of World Congresses on Universal Logic[1]. The present author's research has provided possible answers to these questions (never in time for submission to the contests though), and this article coherently presents them in a very general setting.

Our approach is formalist in nature, i.e. we use type theories to define the grammars and inference systems of formal languages. This has two motivations. First, it is part of a general trend towards formalizing and mechanically verifying theorems in proof assistants. These are growingly used to verify software (e.g. [26]), mathematics (e.g. [15]) and to a lesser extent logics. When applied to logics, this approach requires a formal meta-language, in which logics are defined, usually called the **logical framework** [40]. Therefore, we begin by investigating the common features of logical frameworks (Section 2) including the dependent type theory LF [21] and the higher-order logic underlying Isabelle [39]. Our main result here is a simple and general definition of the notion of logical framework.

Secondly, the formalist approach enables complementing universal logical concepts with generic tool support. This lets researchers build new logics by combining reusable components. And generic tools provide *rapid prototyping* where, e.g. parser, checker, module system and user interface are provided uniformly at low cost. Thus, researchers can apply and evaluate logics easily and at large scales.

In the later sections, we assume a fixed, arbitrary logical framework and give general definitions for logics (Section 3), translations (Section 4), and combinations (Section 5). First we define the syntax and proof theory of a **logic as theories** of the logical framework. A major achievement is that we can then uniformly represent **semantics and translations as theory morphisms**. Indeed, both are inductive functions that interpret one formal system in another one; the only difference is that the

[1] http://www.uni-log.org/, 2005, 2007, 2010, 2013.

doi:10.1093/logcom/exu079

codomain of semantics is usually a rich mathematical language such as axiomatic set theory. Central results in these sections are criteria for soundness and completeness of logics and translations. Finally, we show how we can build theories modularly. This lets us define logic **combinations as colimits** in the logical framework. We conclude in Section 7 after reviewing related work in Section 6.

Our central **contribution** is to integrate several independent lines of research into a coherent framework. This required so many generalizations and simplifications that most of our results are novel.

- We use MMT [44] as the universal representation language. But where [44] represented logical frameworks as black boxes, we develop a novel definition that represents them both categorically and declaratively. This permits in particular proving the preservation of judgements along theory morphisms, which is well known for individual languages, in appropriate generality.
- We use the ideas of [42] to give formalist representations of logics and model theory in LF. But we generalize them to arbitrary logical frameworks and greatly reduce their complexity. This yields more general and deeper results than [42]. For example, we can give a Henkin-style model construction and a completeness criterion for arbitrary logics. We also introduce the paradigm of representing semantics as a chain of refinements and show when and how semantics is preserved by refinement steps.
- In [46], we already sketched an example for using logical relations in LF to reason about logic translations. Here, we state the method systematically for arbitrary logical frameworks and develop the soundness and completeness criteria as well as a novel notion of equivalence between logics.

Finally, while this article deals solely with the theoretical aspects, our framework is maturely implemented [43] and has been applied to the formalization of a large library of logics [7, 27]. The present article aligns the theoretical background with the implementation and library, which have evolved for several years.

2 What is a logical framework?

Our approach is independent of the specific logical framework. Therefore, we first give a general definition of logical framework. Incidentally, this definition is relatively simple because we can abstract from most of the type theoretical technicalities usually needed to define individual frameworks.

Our definition couples abstract categorical and concrete declarative aspects. The former are described in Section 2.1, the latter in Section 2.2, and we combine the two in Section 2.3. These definitions are inspired by MMT, which was introduced as a Module system for Mathematical Theories in [44].

2.1 Logical frameworks as categories

Categorically, we can see logical frameworks as categories of theories and theory morphisms. Here we specify the common features that we have observed about these categories.

DEFINITION 2.1 (Category with Inclusions)
A category with inclusions consists of a category together with a broad subcategory that is a partial order.

We write $A \hookrightarrow B$ for the morphisms of the subcategory, and if $f : B \to C$, we write $f|_A$ for the restriction $f \circ (A \hookrightarrow B) : A \to C$.

Our categories with inclusions are reminiscent of inclusion systems [11, 12] but weaker in that there is no unique factorization of morphisms into the composition of an inclusion and an epimorphism.

We call the objects in these categories **theories** and the morphisms **theory morphisms**. We call the morphisms $A \hookrightarrow B$ **inclusions** and say that B is an **extension** of A.

The distinguished subcategory simply means that we can read $A \hookrightarrow B$ as a partial order on the theories.

Definition 2.2 (Pushouts of Inclusions)
A category with **pushouts of inclusions** consists of a category with inclusions and two *partial* operators that define for a morphism $m : A \to B$ and an inclusion $A \hookrightarrow X$

- an object $m(X)$ that includes B
- a morphism $m^X : X \to m(X)$

such that the left diagram below is a pushout.

$$\begin{tikzcd}
X \arrow[r, "m^X"] & m(X) \\
A \arrow[u, hook] \arrow[r, "m"] & B \arrow[u, hook]
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[rrr, "m^X"] \arrow[dr, "f"] & & & m(X) \arrow[dl, "m(f)"'] \\
& X' \arrow[r, "m^{X'}"] & m(X') & \\
A \arrow[uu, hook] \arrow[ur] \arrow[rrr, "m"] & & & B \arrow[uu, hook] \arrow[ul]
\end{tikzcd}$$

In that case, given a morphism $f : X \to X'$ such that $f|_A = id_A$, we write $m(f)$ for the universal morphism $m(X) \to m(X')$ induced by the pushout as shown on the right.

Remark 2.3 (Partiality of Pushouts)
Our use of partial a partial pushout operator may appear surprising because there are many categories that naturally admit total pushout operators. Our choice is motivated by the observation that it is difficult (we conjecture: impossible) to combine totality with two other desirable properties, one of them being coherence as in Definition 2.4. We will discuss this further in Remark 2.28.

We would like to use $m(-)$ like a functor that maps extensions of A to extensions of B. However, in general, the functoriality laws only hold up to isomorphism. Therefore, we define:

Definition 2.4 (Coherent Pushouts)
We say that pushouts of inclusions are **coherent** if they commute with identifies and composition in the sense that

$$\begin{array}{rclcrcll}
id_A(X) & = & X & \text{and} & id_A^X & = & id_X & \\
m(A) & = & B & \text{and} & m^A & = & m & \\
\\
(n \circ m)(X) & = & n(m(X)) & \text{and} & (n \circ m)^X & = & n^{m(X)} \circ m^X & \\
m(Y) & = & m^X(Y) & \text{and} & m^Y & = & (m^X)^Y & \text{for } X \hookrightarrow Y
\end{array}$$

and such that the left-hand sides of the above equations are defined whenever the respective right-hand side is.

Here the equations on the right are between morphisms and imply the ones between their codomains on the left. The diagrams below give the commutative diagrams for the cases regarding composition:

$$
\begin{tikzcd}
X \arrow[r, "m^X"'] \arrow[rr, bend left, "(n\circ m)^X"] & m(X) \arrow[r, "n^{m(X)}"'] & (n\circ m)(X) \\
A \arrow[u, hook] \arrow[r, "m"'] & B \arrow[u, hook] \arrow[r, "n"'] & C \arrow[u, hook]
\end{tikzcd}
\qquad
\begin{tikzcd}
Y \arrow[r, bend left, "m^Y"] \arrow[r, bend right, "(m^X)^Y"'] & m^X(Y) \\
X \arrow[u, hook] \arrow[r, "m^X"'] & m(X) \arrow[u, hook] \\
A \arrow[u, hook] \arrow[r, "m"'] & B \arrow[u, hook]
\end{tikzcd}
$$

Coherence is crucial for implementations, where we want to give the pushout as an algorithm that computes $m(X)$ from X. That requires making a canonical choice among all the isomorphic pushouts. And if this choice is not made coherently, the implementation becomes awfully complex.

Definition 2.5 (Judgements)
Let $\mathcal{JUDG}$ be the category of sets with subsets, i.e.

- $\mathcal{JUDG}$-objects are pairs (a,A) where $a \subseteq A$.
- $\mathcal{JUDG}$-morphisms $f:(a,A)\to(b,B)$ are maps $f:A\to B$ that preserve the subset, i.e. if $x\in a$ then $f(x)\in b$.

$\mathcal{JUDG}$ has inclusion morphisms $(a,A)\hookrightarrow(b,B)$ for $a\subseteq b$ and $A\subseteq B$.

Definition 2.6 (Abstract Mmt Language)
An **abstract MMT Language** is a pair $(\mathbf{Th},\mathbf{Jd})$ where

- $\mathbf{Th}$ is a category with inclusions and coherent pushouts of inclusions,
- $\mathbf{Jd}$ is a functor $\mathbf{Th}\to\mathcal{JUDG}$ that preserves inclusions.

If $\mathbf{Jd}(X)=(a,A)$, we call the elements of A X-**judgements** and the elements of a the **true** judgements. With that intuition, the morphisms of $\mathcal{JUDG}$ preserve the truth of judgements.

2.2 *Logical frameworks as declarative languages*

Intuitively, Mmt is a declarative language whose set of theories and judgements is large enough to subsume those of specific frameworks. Thus, we can define specific frameworks simply by picking the theories we need.

Figure 1 gives an overview of the concepts we will introduce. We first define Mmt theories Σ, Σ-expressions E, and the judgements about these expressions in Section 2.2.1. Expressions are formed from the identifiers declared in Σ and a set $\mathcal{C}$ of fixed identifiers provided by the logical framework. Correspondingly, the true judgements are defined by derivations, which are formed from the declarations in Σ and a set $\mathcal{R}$ of fixed rules of the logical framework.

Then we define theory morphisms $\sigma:\Sigma\to\Sigma'$ in Section 2.2.2. Using the homomorphic extension and pushout, we translate Σ-objects to Σ'-objects and show that they preserve the true judgements.

Logical framework: sets $\mathcal{C}$ and $\mathcal{R}$ of fixed identifiers and rules		
	Theory Σ	Morphism $\sigma : \Sigma \to \Sigma'$
Set of	typed identifier declarations $c : E$	assignments $c \mapsto E'$
Σ-expressions E	formed from $\mathcal{C}$ and Σ-identifiers	mapped to Σ' by homomorphic extension
Σ-extensions	inclusions $\Sigma \hookrightarrow \Sigma, \Gamma$	mapped to Σ' by pushout
Σ-judgements	derived from $\mathcal{R}$ and Σ-declarations	preserved along σ

FIGURE 1. Overview of MMT concepts.

2.2.1 Theories and expressions

Identifiers. We will use MMT URIs [44] as identifiers. For our purposes, the following definition is sufficient:

DEFINITION 2.7 (Identifiers)
An identifier is either of the form $T?x$ (**global identifier**) or of the form x (**local identifier**), where T is a URI and x a string.

MMT does not have any built-in identifiers. Therefore, a set of identifiers must be provided to get off the ground:

DEFINITION 2.8 (Urtheory)
An **urtheory** is a fixed set of global identifiers.

For the remainder of this section, we will assume a fixed urtheory $\mathcal{C}$.

Theories. An MMT theory Σ consists of declarations $c:E$ of typed identifiers, where the type E is a Σ-expression as defined in Definition 2.12. MMT declarations and expressions are very general: expressions subsume terms, types, formulas, proofs, etc., and declarations subsume type declarations, function symbols, axioms, rules, etc.

To capture an acyclic dependency between declarations, we use a strict order between them:

DEFINITION 2.9 (Theories)
An MMT **theory** Σ consists of

- a well-founded strictly ordered set $(\mathrm{dom}(\Sigma), <)$ of identifiers, called the **domain**,
- a mapping that maps every $c \in \mathrm{dom}(\Sigma)$ to a Σ^c-expression $\Sigma(c)$, which is called the **type** of c.

Here Σ^c is the restriction of Σ to the set $\{d \in \mathrm{dom}(\Sigma) | d < c\}$.

Intuitively, Σ^c is the subtheory of Σ containing the declarations before c. The well-foundedness guarantees that every declaration depends on only finitely many preceding declarations.

REMARK 2.10 (Infinite Theories)
MMT as defined in [44] describes theories using a grammar and an inference system. That is also our primary interest for logical frameworks.

However, it can be useful to consider infinite theories as well, especially when defining models. Therefore, we word all definitions and theorems in such a way that they apply to the infinite case as well.

NOTATION 2.11 (Theories)
We write Σ, Γ for the theory arising by appending the declarations of Γ to Σ. Here 'append' means that $c < x$ whenever $c \in \mathrm{dom}(\Sigma)$ and $x \in \mathrm{dom}(\Gamma)$.

Moreover, we usually write finite theories Σ as $\Sigma = c_1 : \Sigma(c_1), \ldots, c_n : \Sigma(c_n)$.

Expressions. Now we define the judgements and the true judgements over an MMT theory. Judgements will be predicates about expressions, and the true judgements will be defined by an inference system. Therefore, most of the work lies in defining the expressions and the rules of the inference system.

The expressions over an MMT theory Σ are similar to S-expressions [31] whose leaves are the identifiers $c \in \mathrm{dom}(\Sigma)$. However, we generalize S-expressions to permit variable binding, which yields a definition similar to OPENMATH objects [3].

DEFINITION 2.12 (Expressions)
Consider an MMT-theory Σ. Then a Σ-expression is:

- an identifier c declared in Σ, or
- of the form $C(\Gamma; A_1, \ldots, A_n)$ where
  - $C \in \mathcal{C}$, called the **constructor**,
  - $\Gamma = \ldots, x_i : T_i, \ldots$ is a list of declarations of local identifiers, called the **bound variables**, such that Σ, Γ is a theory,
  - the A_i are Σ, Γ-expressions, called the **arguments**.

EXAMPLE 2.13 (λ-calculus)
To give an urtheory for the simply typed λ-calculus in MMT, we use four constructors $\mathcal{C} = \{\mathtt{type}, \rightarrow, \lambda, @\}$. Alternatively, we can add constructors `kind` and Π to obtain an urtheory for the dependently typed λ-calculus.

We introduce the usual notations for them as follows:

Constructor	Abstract MMT expression	Concrete notation
Universe of types	$\mathtt{type}(\cdot;\cdot)$	$\mathtt{type}$
Function types	$\rightarrow(\cdot; A, B)$	$A \rightarrow B$
Abstraction	$\lambda(x : A; t)$	$\lambda x : A.t$
Application	$@(\cdot; f, t)$	$f\, t$
Universe of kinds	$\mathtt{kind}(\cdot;\cdot)$	$\mathtt{kind}$
Dependent function types	$\Pi(x : A; t)$	$\Pi x : A.t$

NOTATION 2.14 (Free Variables)
To regain the usual notations for free variables, we allow writing $E[x_1, \ldots, x_n]$ to emphasize that E is an expression over a theory $\Sigma, x_1 : A_1, \ldots, x_n : A_n$. In that case, we write $E[t_1, \ldots, t_n]$ for the result of **substitution** of t_i for x_i.

Using Definition 2.24, we can define this as an abbreviation for $\overline{id_\Sigma, x_1 \mapsto t_1, \ldots, x_n \mapsto t_n}(E)$.

Inference System. MMT does not define a specific typing relation between expressions. Instead, individual languages supply their own typing relation. Here, we give a novel formulation that improves upon the one used in [44] by using a generic inference system consisting of judgements and rules:

DEFINITION 2.15 (Judgements)
Given a theory Σ, the MMT judgements are

$\vdash_\Sigma E : T$	expresses 'E has type T'
$\vdash_\Sigma \_ : T$	expresses 'T may occur as the type of an identifier'

where E and T are Σ-expressions. Moreover, we use $\vdash_\Sigma T$ as abbreviation for 'there is an E such that $\vdash_\Sigma E:T$'.

We also write $\vdash_\Sigma J$ for an arbitrary judgement and $\nvdash_\Sigma J$ for the corresponding negated judgements.

REMARK 2.16 (Equality)
It is possible (and reasonable) to generalize Definition 2.15 to also include an equality judgement $\vdash_\Sigma E = E'$. We avoid this here only for brevity and occasionally remark on the adaptations necessary to add equality.

DEFINITION 2.17 (Rules)
A $\mathcal{C}$-**rule** is an inference rule of the form

$$\text{for all } S, e_1, \ldots, e_n \qquad \frac{\ldots \quad \vdash_{S,\Gamma_i} J_i \quad \ldots}{\vdash_S J_0}$$

Here S is a meta-variable for an arbitrary theory S, and the e_i are meta-variables for arbitrary expressions. The theories Γ_i and the expressions in J_i may use the identifiers in Γ_i, the constructors in $\mathcal{C}$, the meta-variables e_i, and substitution.

As usual, we will omit the list $S, e_1, \ldots, e_n$ of meta-variables when giving rules.

Concrete Languages. Finally, individual languages are obtained by fixing the constructors and rules:

DEFINITION 2.18 (Concrete MMT Language)
A **concrete MMT language** consists of an urtheory $\mathcal{C}$ and a set $\mathcal{R}$ of $\mathcal{C}$-rules.

As defined in Definition 2.12, expressions and thus the judgements are generated from the identifiers in $\mathcal{C}$. Similarly, the derivations and thus the true judgements are generated from the $\mathcal{C}$-rules and one fixed additional rule provided by MMT:

DEFINITION 2.19 (Well-Formedness)
Given a concrete MMT language $(\mathcal{C}, \mathcal{R})$, the true judgements are defined by the inference system consisting of the $\mathcal{C}$-rules in $\mathcal{R}$ and the rule

$$\frac{S \text{ well-formed}}{\vdash_S c : S(c)}$$

A theory Σ is **well-formed** if $\vdash_{\Sigma^c} \_ : \Sigma(c)$ for all $c \in \mathrm{dom}(\Sigma)$.

Intuitively, a theory is well-formed if for every declaration $c:E$, the previous declarations can prove that E may occur as the type of an identifier.

EXAMPLE 2.20 (Simply Typed λ-Calculus)
To obtain a concrete MMT language for the simply typed λ-calculus, we extend Example 2.13 with the well-known typing rules

$$\frac{\vdash_S A : \mathtt{type} \qquad \vdash_S B : \mathtt{type}}{\vdash_S A \to B : \mathtt{type}}$$

$$\frac{\vdash_S A \to B : \mathtt{type} \qquad \vdash_{S, x:A} t : B}{\vdash_S \lambda x : A.t : A \to B} \qquad \frac{\vdash_S f : A \to B \qquad \vdash_S t : A}{\vdash_S f\, t : B}$$

and the rules

$$\frac{\vdash_S A : \mathtt{type}}{\vdash_S \_ : A} \qquad \frac{}{\vdash_S \_ : \mathtt{type}}$$

The latter two rules make theories well-formed iff they contain only typed constant declarations $c:A$ for a type A or type declarations $a:\mathtt{type}$.

If, following Remark 2.16, we use an equality judgement, we also add the rules for β and η-equality.

EXAMPLE 2.21 (Dependently Typed λ-Calculus)
For the dependent λ-calculus, we extend Example 2.13 with

$$\frac{\vdash_S A:\mathtt{type} \qquad \vdash_{S,x:A} B:\mathtt{type}}{\vdash_S \Pi x:A.B:\mathtt{type}}$$

$$\frac{\vdash_S \Pi x:A.B:\mathtt{type} \qquad \vdash_{S,x:A} t:B}{\vdash_S \lambda x:A.t:\Pi x:A.B} \qquad \frac{\vdash_S f:\Pi x:A.B \qquad \vdash_S t:A}{\vdash_S f\,t:B[t]}$$

and the corresponding triplet of rules for kind-level λ-abstraction as well as the rules

$$\frac{}{\vdash_S \mathtt{type}:\mathtt{kind}} \qquad \frac{\vdash_S A:\mathtt{type}}{\vdash_S \_:A} \qquad \frac{\vdash_S K:\mathtt{kind}}{\vdash_S \_:K}$$

The latter two rules differ from their simply typed counterparts by allowing kinded declarations $c:K$ for any kind $K:\mathtt{kind}$.

Equality is treated as in Example 2.20.

2.2.2 Category structure

Theory Morphisms. Theory morphisms $\sigma:\Sigma\to\Sigma'$ map all identifiers of Σ to Σ'-expressions:

DEFINITION 2.22 (Morphism)
An MMT theory **morphism** σ from Σ to Σ' is a mapping of identifiers $c\in\mathrm{dom}(\Sigma)$ to Σ'-expressions $\sigma(c)$.

NOTATION 2.23 (Morphisms)
Like in Notation 2.11, we write σ,γ for the morphism that appends the cases of γ to the ones of σ.

Accordingly, we write morphisms out of a finite theory as $\sigma = c_1\mapsto\sigma(c_1),\ldots,c_n\mapsto\sigma(c_n)$.

Homomorphic Extension. A theory morphism $\sigma:\Sigma\to\Sigma'$ is extended homomorphically to a mapping $\overline{\sigma}$ of Σ-expressions to Σ'-expressions:

DEFINITION 2.24 (Homomorphic Extension)
Consider a theory morphism $\sigma:\Sigma\to\Sigma'$. Then the mapping $\overline{\sigma}(-)$ from Σ-expressions to Σ'-expressions is defined by:

$$\overline{\sigma}(c)=\sigma(c)$$

$$\overline{\sigma}(C(\Gamma;\ldots,A_i,\ldots)=C(\sigma(\Gamma);\ldots,\overline{\sigma^\Gamma}(A_i),\ldots)$$

where σ^Γ extends σ with $x\mapsto x$ for all $x\in\mathrm{dom}(\Gamma)$, and $\sigma(\Gamma)$ is as in Definition 2.26.

Pushouts. The MMT theories and morphisms form a category with inclusions in the following way:

DEFINITION 2.25 (Category Structure)
For a theory Σ, we define the **identity** morphism as

$$id_\Sigma : c\mapsto c \qquad \text{for } c\in\mathrm{dom}(\Sigma)$$

And given $\sigma:\Sigma\to\Sigma'$ and $\sigma':\Sigma'\to\Sigma'$, we define the **composition** as

$$\sigma'\circ\sigma : c\mapsto\overline{\sigma'}(\sigma(c)) \qquad \text{for } c\in\mathrm{dom}(\Sigma).$$

An **inclusion** morphism $\Sigma\hookrightarrow\Sigma'$ exists whenever Σ is a restriction of Σ' to some subset of $\mathrm{dom}(\Sigma')$ and is defined by $c\mapsto c$ for $c\in\mathrm{dom}(\Sigma)$. Therefore, we will occasionally use the notation $id_\Sigma:\Sigma\hookrightarrow\Sigma'$.

We can also define coherent pushouts of inclusions:

DEFINITION 2.26 (Pushouts)
Consider $\sigma:\Sigma\to\Sigma'$ and $\Sigma\hookrightarrow\Sigma,\Gamma$ such that $\mathrm{dom}(\Sigma')\cap\mathrm{dom}(\Gamma)=\varnothing$. Then the **pushouts** in MMT are defined by:

$$\mathrm{dom}(\sigma(\Sigma,\Gamma))=\mathrm{dom}(\Sigma')\cup\mathrm{dom}(\Gamma)$$

$$\sigma(\Sigma,\Gamma)(c)=\begin{cases}\Sigma'(c) & \text{if } c\in\mathrm{dom}(\Sigma')\\ \overline{\sigma^{\Sigma,\Gamma}}(\Gamma(c)) & \text{if } c\in\mathrm{dom}(\Gamma)\end{cases} \qquad \sigma^{\Sigma,\Gamma}:c\mapsto\begin{cases}\sigma(c) & \text{if } c\in\mathrm{dom}(\Sigma)\\ c & \text{if } c\in\mathrm{dom}(\Gamma)\end{cases}$$

Consider morphisms $\varphi:\Sigma,\Gamma\to\Phi$ and $\varphi':\Sigma'\to\Phi$ such that $\varphi|_\Sigma=\varphi'\circ\sigma$. Then φ must be of the form $\varphi|_\Sigma,\gamma$, and we obtain the universal morphism $u:\sigma(\Sigma,\Gamma)\to\Phi$ as φ',γ.

The uniqueness of the universal morphism is immediate. The coherence properties can be verified directly.

$$\begin{tikzcd}
 & & \Phi \\
\Sigma,\Gamma \arrow[r, "\sigma^{\Sigma,\Gamma}"'] \arrow[urr, "\varphi"] & \sigma(\Sigma,\Gamma) \arrow[ur, "u"] & \\
\Sigma \arrow[u, hook] \arrow[r, "\sigma"'] & \Sigma' \arrow[u, hook] \arrow[uur, "\varphi'"'] &
\end{tikzcd} \qquad u=\varphi|_\Sigma,\gamma$$

NOTATION 2.27 (Pushouts)
The notations $\sigma(\Sigma,\Gamma)$ and $\sigma^{\Sigma,\Gamma}$ are rather unwieldy.

Therefore, we write σ^Γ instead of $\sigma^{\Sigma,\Gamma}$. And we write $\sigma(\Gamma)$ for the fragment of $\sigma(\Sigma,\Gamma)$ that is appended to Σ', i.e. we have

$$\sigma(\Sigma,\Gamma)=\Sigma',\sigma(\Gamma)$$

With this notation, MMT pushouts $\sigma(\Gamma)$ can be seen as the homomorphic extension of σ to theory fragments Γ. In particular, we have $\sigma(\Gamma,x:T)=\sigma(\Gamma),x:\sigma^\Gamma(T)$.

REMARK 2.28 (Totality, Coherence and Natural Identifiers)
Our abstract and concrete definitions of pushout are motivated by three desirable properties: the totality of the pushout operators, their coherence and the use of natural identifiers.

Here by 'natural identifiers', we mean that the identifiers in $\sigma(\Gamma)$ are obtained naturally from those of Γ. Definition 2.26 is an extreme example of a pushout with natural identifiers by using the same identifiers in the two theories.

We conjecture that it is not possible to define pushouts in a way that realizes all three properties at once.

We can obtain coherent total pushouts if we sacrifice natural identifiers. For example, we can use de Bruijn indices instead of identifiers, which quotients out the choice of identifiers. We can also obtain total pushouts with natural identifiers if we sacrifice coherence. For example, we can prefix all identifiers in $\sigma(\Gamma)$ with some $p \notin \mathrm{dom}(\Sigma')$. But both options would make the pushouts highly impractical to work with.

Therefore, our approach sacrifices totality instead. This has the drawback that we have to check applicability of pushout every time. But both in theory and in practice, we can work around that relatively easily and effectively by using namespaces as described in Notation 2.29.

The following convention helps us construct pushouts in MMT without bothering about the partiality:

NOTATION 2.29 (Namespace Convention)
Whenever we work with a theory whose name consists of Latin letters we assume that all identifiers declared in that theory are global. Moreover, we assume that the declarations in theories with different names use different URIs. We always omit those URIs from the notation though.

Whenever, we work with a theory fragment whose name consists of Greek letters, we assume that all identifiers declared in that theory are local.

For example, in Theorem 2.31, we will use a morphism $\sigma : S \to S'$ and theories S, Γ. Notation 2.29 guarantees that $\mathrm{dom}(\Gamma)$ is disjoint from both $\mathrm{dom}(S)$ and $\mathrm{dom}(S')$.

Well-Formedness. Intuitively, a theory morphism is well-formed if it preserves the type of every identifier:

DEFINITION 2.30 (Well-Formed Morphisms)
A theory morphism $\sigma : \Sigma \to \Sigma'$ is **well-formed** if $\vdash_{\Sigma'} \sigma(c) : \overline{\sigma}(\Sigma(c))$ for all $c \in \mathrm{dom}(\Sigma)$.

Concrete MMT languages provide an extremely general setting in which we can prove that well-formed theory morphisms in fact preserve all judgements:

THEOREM 2.31 (Preservation of judgements)
Consider a concrete MMT language, well-formed theories S and S', and a well-formed morphism $\sigma : S \to S'$. Let $\overline{\sigma}(J)$ be the result of applying $\overline{\sigma}(-)$ to the expressions in J.

Then

$$\text{if} \quad \vdash_S J \quad \text{then} \quad \vdash_{S'} \overline{\sigma}(J)$$

Moreover, if S, Γ is well-formed, so are $S', \sigma(\Gamma)$ and $\sigma^{\Gamma} : S, \Gamma \to S', \sigma(\Gamma)$.

PROOF. It is sufficient to prove the statement for finite Γ: because all expressions and derivations can only refer to finitely many identifiers of Γ, any counter-example for an infinite Γ would give rise to a counter-example for a finite Γ.

Then we prove all claims by mutual induction on the derivation D of $\vdash_S J$ and the finite set of derivations establishing the well-formedness of S, Γ relative to the well-formedness of S.

If D consists only of the rule for constants from Definition 2.19, the needed property follows immediately from the well-formedness of σ.

Otherwise, D is of the form

$$\frac{\cdots \quad \dfrac{D_i}{\vdash_{S,\Gamma_i} J_i} \quad \cdots}{\vdash_S J}\, r$$

for some derivations D_i. Here $r \in \mathcal{R}$, the meta-variable S of r is instantiated with S, and the meta-variables e_i of r are instantiated with some expressions E_i.

Applying the induction hypothesis to the D_i using the morphisms $\sigma^{\Gamma_i}: S,\Gamma_i \to S',\sigma(\Gamma_i)$ yields derivations

$$\frac{D'_i}{\vdash_{S',\sigma(\Gamma_i)} \overline{\sigma^{\Gamma_i}}(J_i)}$$

Then we obtain the needed derivation by applying r to the D'_i. This time we instantiate S with S' and each e_i with $\overline{\sigma}(E_i)$.

For each of the remaining two well-formedness claims, we have to establish one S'-judgement for each identifier in $\mathrm{dom}(S,\Gamma)$. For the identifiers in $\mathrm{dom}(S)$, these follow immediately from the assumptions. For an identifier $x \in \mathrm{dom}(\Gamma)$, they follow from the corresponding S-judgement by applying the induction hypotheses to S,Γ^x. ■

Remark 2.32 (Equality Rules)
If, following Remark 2.16, we add an equality judgement we have to adapt Definition 2.19 by adding rules for equality. These are α-conversion (renaming of bound variables), reflexivity, symmetry, transitivity and congruence.

The congruence rules guarantee that all operations preserve equality. There is one congruence rule for each primitive judgement

$$\frac{\vdash_S E = E' \quad \vdash_S T = T' \quad \vdash_S E : T}{\vdash_S E' : T'} \qquad \frac{\vdash_S T = T' \quad \vdash_S \_ : T}{\vdash_S \_ : T'}$$

and one congruence rule scheme for composed expressions of any arity

$$\frac{\ldots \quad \vdash_{S,\Gamma^{x_i}} T_i = T'_i \quad \ldots \quad \vdash_{S,\Gamma} E_j = E'_j \quad \ldots}{\vdash_S C(\underbrace{\ldots, x_i : T_i, \ldots}_{\Gamma}; \ldots, E_j, \ldots) = C(\ldots, x_i : T'_i, \ldots; \ldots, E'_j, \ldots)}$$

Theorem 2.31 can be extended to the equality judgement in a straightforward way.

The above rules have the effect that Mmt languages must admit subject reduction (i.e. if $E : T$ and $E = E'$, then $E' : T$) because it is subsumed by the congruence of typing. Similarly, Mmt languages for λ-calculi must admit the ξ-rule (i.e. if $E[x] = E'[x]$, then $\lambda x : T.E[x] = \lambda x : T.E'[x]$) because it is subsumed by the congruence of expression formation.

Notation 2.33
From now on, we simply write $\sigma(E)$ instead of $\overline{\sigma}(E)$.

2.3 *A logical framework is an MMT language*

Finally, we have:

Theorem 2.34
Every concrete Mmt language induces an abstract Mmt language.

Proof. The category of theories consists of the well-formed Mmt-theories and the well-formed theory morphisms between them. Inclusions and pushouts are as defined above. It is straightforward to show that all constructions (identity, composition, inclusion, pushouts) preserve well-formedness.

The (true) Σ-judgements are the ones of Mmt. Theorem 2.31 shows the well-definedness. ■

Then we can finally make the main definition of this section:

DEFINITION 2.35 (Logical Framework)
A **logical framework** is a concrete MMT language with distinguished constructors $\mathtt{type}$ and $\mathtt{prop}$. We call expressions $E:\mathtt{type}$ and $E:\mathtt{prop}$ **types** and **propositions**, respectively.

A logical framework has **hypothetical reasoning** if it provides identifiers $\Pi, \rightarrow, \lambda, @$ and rules

$$\frac{\vdash_S A:\mathtt{prop} \qquad \vdash_S B:\mathtt{prop}}{\vdash_S A \rightarrow B:\mathtt{prop}} \qquad\qquad \frac{\vdash_S A:\mathtt{type} \qquad \vdash_{S,x:A} B:\mathtt{prop}}{\vdash_S \Pi x:A.B:\mathtt{prop}}$$

as well as the corresponding rules for λ-abstraction and application @ akin to Example 2.21.

REMARK 2.36 (Hypothetical Reasoning)
Hypothetical reasoning corresponds to the rules $(\mathtt{prop},\mathtt{prop})$ and $(\mathtt{type},\mathtt{prop})$ of pure type systems [2]. The former provides **implication**, the latter **universal quantification** over typed variables.

EXAMPLE 2.37 (LF)
LF [21] is the language given in Example 2.21. LF follows the judgements-as-types paradigm and is a logical framework via $\mathtt{type}=\mathtt{prop}$.

LF has hypothetical reasoning. The type/proposition representing implication is the simple function type $F \rightarrow G$. The type/proposition representing universal quantification is the dependent function type $\Pi x:A.F(x)$.

In the sequel, we use LF for the running examples of this article.

EXAMPLE 2.38 (Isabelle)
The intuitionistic higher-order logic Pure, which underlies Isabelle [39], is also a logical framework in our sense. More precisely, we obtain Pure by extending Example 2.20 with:

- constructors for a base type $\mathtt{prop}$, implication $\Longrightarrow$, and universal quantification $\forall$;
- one constructor for the name of each proof rule so that each proof can be written as an expression; and
- appropriate typing and proof rules such that in particular $\vdash_\Sigma p:F$ holds whenever p is a proof of $\vdash_\Sigma F:\mathtt{prop}$.

Pure also has hypothetical reasoning via $\Longrightarrow$ and $\forall$, and we use the usual notations $F \Longrightarrow G$ and $\forall x:A.F(x)$.

From now on, we assume an arbitrary fixed logical framework with hypothetical reasoning. Unless mentioned otherwise, all theories, morphisms and expressions are well-formed with respect to this framework.

3 What is a logic?

3.1 Syntax is a theory

Figure 2 summarizes the basic intuitions that we will use in this section to formalize a logic L in a given logical framework. The syntax and inference system of L are represented as a theory *Syn* (which declares the logical symbols) and L-theories as extensions Σ of *Syn* (which extend *Syn* with declarations of non-logical symbols).

Concept	Representation
Syntax	Theory Syn of the logical framework
Signatures/theories	Theories Σ that extend Syn
Formulas	Σ-expressions of a certain fixed type o

FIGURE 2. Intuitions behind our representation of syntax.

We follow the Curry–Howard representation so that both logical symbols and axioms are represented as declarations, and both formulas and proofs are represented as expressions. In particular, axioms asserting F are just declarations of the form $a : thm F$.

DEFINITION 3.1 (Logical Theories)
A **logical theory** consists of:

- a theory Syn, which we call the **syntax**;
- a distinguished type $\vdash_{Syn} o : \mathtt{type}$, which we call the type of **sentences**; and
- a distinguished proposition $\vdash_{Syn, F:o} thm[F] : \mathtt{prop}$, which we call the **truth judgement** for the formula F.

such that $\vdash_{Syn} \_ : o$ and $\vdash_{Syn, F:o} \_ : thm[F]$.

We think of expressions $F : o$ as sentences and of derivations of $\vdash_\Sigma thm[F]$ as proofs of F.

NOTATION 3.2 (Truth Judgement)
In Definition 3.1, we demand that thm is a proposition with a free variable F. In practice, this is almost always achieved by declaring an identifier $thm' : o \to \mathtt{prop}$, in which case $thm[F] = thm' F$. For example, in Isabelle thm' is usually called *Trueprop*; in LF, it is often called *nd* or *true*. Therefore, we will often simply write $thm F$ instead of $thm[F]$.

EXAMPLE 3.3 (Propositional and First-Order Logic)
Using the logical framework LF from Example 2.37, we define propositional logic PL as the following logical theory:

$$\begin{array}{lcl} o & : & \mathtt{type} \\ thm & : & o \to \mathtt{type} \\ \top & : & o \\ \bot & : & o \\ \neg & : & o \to o \\ \wedge & : & o \to o \to o \\ \vee & : & o \to o \to o \\ \Rightarrow & : & o \to o \to o \end{array}$$

where o and $thm x$ are the distinguished expressions.
We obtain first-order logic FOL by adding

$$\begin{array}{lcl} i & : & \mathtt{type} \\ \doteq & : & i \to i \to o \\ \forall & : & (i \to o) \to o \\ \exists & : & (i \to o) \to o \end{array}$$

These use currying to represent the connectives: using the notations from Example 2.13, the expression $(\wedge F) G$ represents the sentence $F \wedge G$. Similarly, they use higher-order abstract syntax to represent the binders: the expression $\forall(\lambda x : i.F(x))$ represents the sentence $\forall x.F(x)$. In future examples, we will use the usual notations instead of the ones technically prescribed by our encoding in LF.

For the remainder of this section, we fix a logical theory $\mathcal{L} = (Syn, o, thm)$. Relative to $\mathcal{L}$, we give generic definitions of the syntax of a logic.

Definition 3.4 (Non-Logical Theories)
$\mathcal{L}$-**theories** are well-formed extensions $Syn \hookrightarrow Syn, \Sigma$ of Syn.

The $\mathcal{L}$-**theory morphisms** between $\mathcal{L}$-theories are the morphisms $\sigma : Syn, \Sigma \to Syn, \Sigma'$ satisfying $\sigma|_{Syn} = id_{Syn}$.

While logical theories $\mathcal{L}$ represent logics, the non-logical $\mathcal{L}$-theories Σ represent the theories of these logics. It is customary to call the identifiers in *Syn logical* and the ones in Σ *non-logical*. Therefore, we use the according terminology to speak of logical and non-logical theories, which declare the logical and non-logical symbols, respectively. The phrase 'non-logical theory' is not ideal but yields a very clear terminology in the sequel.

A typical example of non-logical theories are the algebraic theories of first-order logic:

Example 3.5 (Monoids)
The *FOL*-theory of monoids contains the following non-logical declarations

$$\begin{array}{lll}
\circ & : & i \to i \to i \\
e & : & i \\
leftNeutral & : & thm[\forall x.e \circ x \doteq x] \\
rightNeutral & : & thm[\forall x.x \circ e \doteq x] \\
associative & : & thm[\forall x.\forall y.\forall z.(x \circ y) \circ z \doteq x \circ (y \circ z)]
\end{array}$$

We use the usual infix notation for $\circ$.

Note that there is no need to distinguish between *FOL*-signatures and *FOL*-theories: Axioms have the same status as the declarations of function symbols.

Remark 3.6 (Logical and Non-Logical Identifiers)
Note that every $\mathcal{L}$-theory Σ is itself a logical theory. Thus, the distinction between logical and non-logical identifiers is sometimes blurred. This corresponds to a blurred distinction in practice. For example, in first-order logic, equality is sometimes considered as a logical and sometimes as a non-logical identifier.

The difference becomes relevant only when we consider $\mathcal{L}$-theory morphisms, which must keep the logical identifiers fixed.

Remark 3.7 (Restricting the Non-Logical Theories)
Definition 3.4 defines any extension of *Syn* to be an $\mathcal{L}$-theory. If we use, e.g. plain LF as the logical framework, this usually yields more non-logical theories than desirable. For example, the *PL*-theories should only declare propositional variables $p : o$ and axioms $a : thm F$. Similarly, the *FOL*-theories should only declare function symbols $p : i \to \ldots \to i \to i$, predicate symbols $f : i \to \ldots \to i \to o$ and axioms.

However, this limitation only affects our example frameworks. Other logical frameworks can use modified rules for the judgement $\vdash_\Sigma \_ : T$ to make only certain $\mathcal{L}$-theories well formed. For example, we can define a variant of LF along the lines of [22].

The only requirement Definition 3.1 makes is that o and $thm F$ may occur as types, i.e. that we are at least able to declare propositional variables $p:o$ and axioms $a:thm F$. That is a very mild condition that will help in several proofs below.

THEOREM 3.8 (Category of $\mathcal{L}$-Theories)
The $\mathcal{L}$-theories and $\mathcal{L}$-theory morphisms form a category that inherits inclusions and (where defined) pushouts from MMT.

PROOF. This is straightforward. In particular, if $\mathrm{dom}(\Gamma)$ and $\mathrm{dom}(\Sigma')$ are disjoint, the pushout of the $\mathcal{L}$-inclusion $\Sigma \hookrightarrow \Sigma, \Gamma$ (which is an inclusion $Syn, \Sigma \hookrightarrow Syn, \Sigma, \Gamma$ of MMT theories) along $\sigma : \Sigma \to \Sigma'$ is the same as the MMT pushout $(id_{Syn}, \sigma)(\Gamma)$. ■

DEFINITION 3.9 (Sentences)
Given an $\mathcal{L}$-theory Σ, the **Σ-sentences** are the expressions F such that $\vdash_\Sigma F : o$.

REMARK 3.10 (Syntax modulo Equality)
If, following Remark 2.16, we also use an equality judgement, Definitions 3.4 and 3.9 are adapted by taking the quotient modulo the equality judgement. Thus, expressions are identified up to equality, and consequently theories and morphisms are identified up to equality of the expressions occurring in them.

If the logical framework admits a canonical form theorem, we can alternatively restrict our attention to canonical expressions.

3.2 *Semantics is a theory morphism*

We give two abstract definitions of semantics as summarized in Figure 3. First, *constructive* semantics is inspired by proof theory: It is absolute in the sense that there either is a proof for a sentence or not. A sentence is *constructively valid* if it has a proof. More precisely, we use $thm F$ as the type of proofs of F so that proofs are represented as Σ-expressions of type $thm F$ and validity as the non-emptiness of this type.

Secondly, the *denotational* semantics is inspired by model theory: it is relative in the sense that the truth of a sentence depends on the model, which interprets the theory. A sentence is *denotationally valid* if it is true in all models. More precisely, theory morphisms M out of Σ represent Σ-models, and the non-emptiness of the type $M(thm F)$ represents the truth of F in M.

Concept	Representation
Syntax	Theory *Syn* of the logical framework
Signatures/theories	Theories Σ that extend *Syn*
Formulas	Σ-expressions of a certain fixed type o
Constructive, proof-theoretical semantics	
Proofs of formula F	Σ-expressions of type $thm\ F$
Theorems	Formulas for which there is a proof
Denotational, model-theoretical semantics	
Semantics	Theory morphism *sem* out of *Syn*
Models	Theory morphisms M out of Σ that extend *sem*
Truth of F in M	Existence of an expression of type $M(thm\ F)$
Theorems	Formulas that are true in all models

FIGURE 3. Intuitions behind our representation of semantics.

For both definitions, we proceed in two steps. First, we give deliberately simple definitions in Section 3.2.1. These capture the key intuitions and are already sufficient to establish some far-reaching theorems as we see in Section 3.2.2.

Then we introduce logical morphisms in Section 3.2.3 and use them to generalize the semantics in Section 3.2.4. Most importantly, Section 3.2.4 will split the morphisms M into two parts. First, a fixed theory morphism sem maps the logical symbols of Syn to their fixed interpretation. Secondly, models M extend sem with interpretations for the non-logical symbols of Σ. The definitions of Section 3.2.1 will be recovered as the special case where $sem = id_{Syn}$.

3.2.1 Proofs and models

Definition 3.11 (Constructive Semantics)
Consider an $\mathcal{L}$-theory Σ and a Σ-sentence F. Then:

1. A Σ-**proof** of F is an expression p such that $\vdash_{Syn,\Sigma} p : thm F$.
2. A Σ-**disproof** of F is an expression $p[a,g]$ such that $\vdash_{Syn,\Sigma,a:thm F,g:o} p[a,g] : thm g$.
3. F is **constructively valid** if there is a proof of F.

Remark 3.12 (Disproofs)
Our notion of disproofs is not common but straightforward. A disproof is a witness $p[a,g]$, which proves any formula g under an assumption a that F is true. Intuitively, this means that F is a contradiction.

If we had a negation connective $\neg$, we could simply define disproofs of F as proofs of $\neg F$. Our definition has the same effect but avoids assuming a distinguished negation connective.

Definition 3.13 (Denotational Semantics)
Consider an $\mathcal{L}$-theory Σ and a Σ-sentence F. Let $M : Syn, \Sigma \to Syn, \Gamma$ be an $\mathcal{L}$-theory morphism. Then:

1. F is **true** in M if there is a Γ-proof of $M(F)$.
2. F is **false** in M if there is a Γ-disproof of $M(F)$.
3. M is a Σ-**model** if every Σ-sentence is either true or false in M.
4. F is **denotationally valid** if it is true in all models.

We use $\mathcal{L}$-theory morphisms $M : Syn, \Sigma \to Syn, \Gamma$ as models. Intuitively, Γ defines the universe(s) of the model, and M maps every Σ-symbol to its denotation. Then the homomorphic extension of M represents the inductively defined interpretation function that interprets all Σ-expressions in Γ. This idea goes back to the models-as-functors perspective of Lawvere [29].

Remark 3.14 (Models and Falsity)
Usually, models and truth are defined first, and falsity is just the opposite of truth. We proceed differently and define falsity first and use it to define models. This has the same effect but is more convenient in our setting.

Example 3.15 (Propositional Models)
Let $\mathcal{L} = PL$ from Example 3.3. Then Boolean-valued models of propositional logic can be written as theory morphisms into an empty Γ. Given the PL-theory $\Sigma = p_1 : o, \ldots, p_n : o$, a model $M : PL, \Sigma \to PL$ maps $M(p_i) = \top$ or $M(p_i) = \bot$.

However, at this point, these are not technically models because we cannot show that every sentence is either true or false. In fact, no sentence is true and no sentence is false because the types $thm F$

are always empty. We have two options to finish the example: we can add proof rules to *PL* or use a codomain Γ that adds computation rules for the Booleans. We will get back to that in Section 3.2.4.

EXAMPLE 3.16 (Algebraic Presentations)
Consider *FOL* and *Monoid* from Examples 3.3 and 3.5. Models are often given as presentations, e.g. $\langle x|x^n = e\rangle$ for the cyclic monoid of n elements.

We can define it as the inclusion morphism

$$FOL, Group \hookrightarrow FOL, Group, \Gamma$$

where

$$\Gamma = x : i,\ a_1 : thm \neg C_1, \ldots,\ a_{n-1} : thm \neg C_{n-1},\ a_n : thm\, C_n$$

and

$$C_n = \underbrace{x \circ \ldots \circ x}_{n} \doteq e.$$

Just like in Example 3.15, we are still missing the rules that make sure all sentences are true or false in these models.

REMARK 3.17 (More Complex Models)
Usually, $\mathcal{L}$-theory morphisms cannot express all interesting models elegantly because $\mathcal{L}$ lacks the syntactic material to build them. For example, to give the monoid of real numbers under addition, we would have to add one constant for every real number and axioms that define the sum of any two of these constants. Our definitions technically cover this by allowing infinite theories, but obviously this is not always desirable. A better way is to use, e.g. set theory to define the set of real numbers. The more general definition of Section 3.2.4 will permit exactly that.

EXAMPLE 3.18 (Natural Numbers)
Let $\mathcal{L} = FOL$ be first-order logic and $Succ = 0 : i,\ succ : i \to i$ be a *FOL*-theory for the natural numbers. Then the morphism $FOL, Succ \hookrightarrow FOL, Succ, PAx$ where *PAx* are the Peano axioms is a model of the standard natural numbers. (Note that it is straightforward to write the axiom schema for induction in LF.)

Due to Gödel's first incompleteness theorem, we know that standard models for the theory

$$Arith = Succ,\ + : i \to i \to i,\ \cdot : i \to i \to i$$

must have a non-recursively enumerable codomain. Using a recursively enumerable codomain, we can only approximate it using, e.g. the theory morphism $FOL, Arith \hookrightarrow FOL, Arith, PAr$ where *PAr* contains the axioms of Peano arithmetic. This morphism is not a model because there are sentences that are neither true nor false.

REMARK 3.19 (Theory Morphisms versus Models)
Not every theory morphisms is a model. In Examples 3.15 and 3.16, we already pointed out that we have to add rules to ensure every sentence is true or false. In general, there may be sentences that are

- **undetermined**, i.e. that are neither true nor false,
- **over-determined**, i.e. that are both true and false.

Both are well-known problems of logic and usually undecidable. Even showing that a single theory morphism really is a model can be very hard. For example, consider a logic based on set theory with a single non-logical identifier $p:o$, and a model $M:p\mapsto P$. To show that p is not undetermined, we have to show that set theory can prove or disprove P, i.e. we have to prove that P is not independent of the axioms of set theory. Similarly, to show that p is not over-determined, we have to show set theory extended with an axiom P is consistent.

When defining denotational models, we usually avoid this problem by assuming a platonic universe of objects in which the truth/falsity of all properties is determined (although possibly unknown). This amounts to assuming a fixed model of set theory.

Below we will see that many results that we want to state about models can already be stated for theory morphisms. Moreover, for finite theory morphisms, well-formedness is decidable if type-checking in the logical framework is. Therefore, we will formulate definitions for theory morphisms instead of models whenever possible.

REMARK 3.20 (Theory Morphisms versus Theories)
An advantage of using theory morphisms is that every $\mathcal{L}$-theory Σ can itself be seen as a theory morphism via the identity morphism id_Σ. Thus, the concept of theory morphisms unifies theories and models, and we can state many definitions for theory morphisms to apply them to both theories and models.

For example, the sentences that are true in id_Σ are just the Σ-theorems, and the sentences that are false in id_Σ are just the Σ-contradictions. Similarly, the notion of (in)consistent theory morphisms in Definition 3.23 specializes to the usual definition of (in)consistent theories.

For an $\mathcal{L}$-theory Σ, the morphism id_Σ is usually not a model. Therefore, we define:

DEFINITION 3.21
A theory Σ is called **maximal** if id_Σ is a Σ-model.

A maximal theory determines the truth/falsity of every sentence so that all models satisfy the same sentences. An example is the *FOL*-theory of unbounded dense total orders (an example model being the rational numbers). Such theories are occasionally called *maximally consistent* or *complete* theories.

3.2.2 Relating constructive and denotational validity

Consistency. We can define (in)consistent theories generically, but we need one definition first:

DEFINITION 3.22 (Degenerate Cases)
An $\mathcal{L}$-theory Σ is **non-trivial** if there is a Σ-sentence.

An $\mathcal{L}$-theory morphism M is **proper** if some sentence is true in M and some sentence is false in M.

It is easy to make sure that all $\mathcal{L}$-theories are non-trivial and that all $\mathcal{L}$-theory morphisms are proper, e.g. by having a provable sentence $\top$ and a disprovable sentence $\bot$ in $\mathcal{L}$. But in some logics, trivial theories or improper theory morphisms exist, and these occasionally have to be excluded. Consistency is one of those occasions:

DEFINITION 3.23 (Consistency)
An $\mathcal{L}$-theory morphism $M:Syn,\Sigma\to Syn,\Gamma$ is **inconsistent** if there is a sentence that is both true and false in M.

THEOREM 3.24 (Consistency)
Consider an $\mathcal{L}$-theory morphism $M:Syn,\Sigma\to Syn,\Gamma$.
If Σ is non-trivial, M is inconsistent iff Γ is.

Moreover, if M is proper, the following are equivalent:

1. M is inconsistent.
2. We have $\vdash_{Syn,\Gamma,F:o} thm F$.
3. All sentences are true in M.
4. All sentences are false in M.

PROOF. We prove the second statement first. Assume M is proper. Then there are a true sentence F^+ and a false sentence F^-. We prove:

- (1) implies (2): Let p^+ and $p^-[a,g]$ be the witnesses of the truth and falsity of one sentence. Then, for $F:o$, we can use substitution to obtain a witness $p^-[p^+, M(F)]$ of the truth of F.
- (2) implies (3): Immediate.
- (2) implies (4): Immediate.
- (3) implies (1): Choose F^-.
- (4) implies (1): Choose F^+

To prove the first statement, assume M is inconsistent, i.e. some Σ-sentence F is both true and false in M. Then $M(F)$ is both true and false in Γ (seen as the theory morphism $id_{Syn,\Gamma}$), and thus Γ is inconsistent. Conversely, assume Γ is inconsistent. Then Γ is proper, and every Σ-sentence is both true and false in M. Because Σ is non-trivial, M is inconsistent. ■

REMARK 3.25 (Degenerate Cases)
The requirement of Σ being non-trivial in Theorem 3.24 is necessary: every morphism out of a trivial theory is consistent (a model even), independently of whether the codomain is consistent.

Similarly, the requirement of M being proper is necessary. For example, consider the logical theory o:`type`, `p:o`, over which p is the only sentence. Then we can give an improper model interpreting p as true (false), in which property 3 (4) holds but not property (1).

Classical Logic. We can give a general definition of when a logic is classical:

DEFINITION 3.26 (Classical Logic)
Let us write $\forall$ and $\Longrightarrow$ for the hypothetical reasoning of the logical framework. Then we define for any logical theory $\mathcal{L}$:

$$\lightning = \forall F:o.thm F$$

$$\overline{A} = A \Longrightarrow \lightning$$

And we say that $\mathcal{L}$ is **classical** if for all Σ and all $\vdash_\Sigma F:o$

$$\vdash_\Sigma \overline{\overline{thm F}} \quad \text{iff} \quad \vdash_\Sigma thm F$$

Intuitively, $\lightning$ is the proposition of contradiction, which is provable iff a logical theory is inconsistent. And $\overline{A}$ is negation in the logical framework: logical theories usually introduce negation in such a way that $\overline{thm F}$ is equivalent to $thm(\neg F)$. Thus, our classicality captures the double-negation elimination property. Note that the right-to-left implication in Definition 3.26 always holds, and only the left-to-right implication is special for classical logics.

EXAMPLE 3.27 (Intuitionistic and Classical Propositional Logic)
We continue Example 3.3 by adding proof rules to PL. Such encodings have been extensively studied (see e.g. [23]), and we only give the rules for negation and disjunction as examples:

$$\begin{array}{lcl} \neg I & : & \Pi A:o.(thm A \to \lightning) \to thm[\neg A] \\ \neg E & : & \Pi A:o.thm[\neg A] \to thm A \to \lightning \end{array}$$

$$\begin{array}{lcl} \vee I_l & : & \Pi A, B\!:\!o.thm A \to thm[A \vee B] \\ \vee I_r & : & \Pi A, B\!:\!o.thm B \to thm[A \vee B] \\ \vee E & : & \Pi A, B, C\!:\!o.thm[A \vee B] \to (thm A \to thm C) \to (thm B \to thm C) \to thm C \end{array}$$

The proof rules of intuitionistic propositional logic *IPL* differ from those of classical propositional logc *CPL* in only one declaration: *CPL* additionally has the axiom schema for *tertium non datur*:

$$tnd \quad : \quad \Pi F\!:\!o.thm[F \vee \neg F]$$

Using the above proof rules, we see that $\vdash_{IPL} \Pi F\!:\!o.thm(F \vee \neg F)$ is indeed equivalent to $\vdash_{IPL} \Pi F\!:\!o.\overline{\overline{thm F}} \to thm F$.

REMARK 3.28
The last observation of Example 3.27 prompted us to change the definition of classical logic in the LATIN logic atlas [7] from $tnd\!:\!\Pi F\!:\!o.thm(F \vee \neg F)$ to $classical\!:\!\Pi F\!:\!o.\overline{\overline{thm F}} \to thm F$. The latter has the advantage that it does not depend on any connective and can thus be combined with any logic. This fits in well with the modular development in LATIN, where every logical feature is formalized individually. *tnd* remains as a theorem that is proved in all classical logics that import disjunction and negation.

Model Existence. We can now state the common theorem about extending consistent theories to maximal theories in an extremely general form. First we establish:

THEOREM 3.29
Assume a consistent theory Σ and a sentence F. Then:

1. if $\not\vdash_\Sigma \overline{thm F}$, then $\Sigma, a\!:\!thm F$ is consistent,
2. if $\mathcal{L}$ is classical:
   if $\not\vdash_\Sigma thm F$, then $\Sigma, a\!:\!\overline{thm F}$ is consistent.

Here $a \notin \mathrm{dom}(\Sigma)$ is an arbitrary fresh identifier.

PROOF. First note that inconsistency implies (by applying λ-abstraction to the second property in Theorem 3.24) the existence of a term p of type $\lightning$. Then we prove both claims indirectly:

1. If we had $\vdash_{\Sigma, a:thm F} p\!:\!\lightning$, we could form $\vdash_\Sigma \lambda a\!:\!thm F.p\!:\!(thm F) \Longrightarrow \lightning$, from which we would get $\vdash_\Sigma \overline{thm F}$.
2. If we had $\vdash_{\Sigma, a:\overline{thm F}} p\!:\!\lightning$, we could form $\vdash_\Sigma \lambda a\!:\!\overline{thm F}.p\!:\!\overline{thm F} \Longrightarrow \lightning$, from which we would get $\vdash_\Sigma \overline{\overline{thm F}}$. Then classicality would yield $\vdash_\Sigma thm F$.

■

REMARK 3.30
The assumption of $\mathcal{L}$ being classical in Theorem 3.29 (2) is necessary in the following sense: If the statement holds for all Σ, then $\mathcal{L}$ is classical.

Now we can iterate Theorem 3.29 to extend a consistent theory until it is maximal:

THEOREM 3.31
Every countable consistent theory can be extended to a maximal theory.

PROOF. We start with a consistent theory $X := \Sigma$ and iteratively extend X to a maximal theory by adding declarations. We enumerate all the sentences and for each sentence F,

- if $\vdash_X thm F$ or $\vdash_X \overline{thm F}$, we do nothing

- otherwise, we replace X with $X, a : thm F$ (for some fresh identifier a).

The resulting theory X is the limit over these countably many iterations. Clearly X extends Σ.

Now assume X were inconsistent, i.e. there is a term $\vdash_X p : \lightning$. p can only use finitely many identifiers of X, so there must be an inconsistent fragment of X obtained after finitely many iterations. But every iteration preserves consistency due to Theorem 3.29. Therefore, X is consistent if Σ is.

To show that id_X is a model, we have to show that X is consistent (which we did above) and that every sentence F is determined in X. The latter holds because F must have occurred in the iteration, and therefore $\vdash_X thm F$ (i.e. F is true) or $\vdash_X \overline{thm F}$ (in which case F is false). ■

Recall that theories can be seen as theory morphisms and maximal theories as models. Thus, this corresponds to the model existence theorem well-known from Henkin-style completeness proofs [20].

Remark 3.32
The restriction to countable theories in Theorem 3.31 is a simplification to avoid cardinality issues because the size of our Mmt theories is not restricted. In practice, theories are countable anyway.

The above theorems lead up to the main theorem about semantics:

Theorem 3.33 (Semantics)
Consider a theory Σ and a Σ-sentence F. Then:

1. If F is constructively valid, then F is denotationally valid.
2. If $\mathcal{L}$ is classical:
   if F is denotationally valid, then F is constructively valid.

Proof. 1. Every model M maps the proof p of F to an expression witnessing the truth of F.
2. If Σ is inconsistent, then F is anyway constructively valid. So assume it is consistent. We proceed indirectly and assume that $\nvdash_\Sigma thm F$. Then $\Sigma, a : \overline{thm F}$ is consistent due to Theorem 3.29 and has a maximal extension X due to Theorem 3.31. But by construction, F is false in X, which violates the assumption that X is denotationally valid.

■

Example 3.34 (First-Order Logic)
Let $\mathcal{L} = FOL$ be first-order logic. The maximal theories constructed by Theorem 3.31 are the usual ones known for FOL. They form a system of representatives for model classes modulo elementary equivalence.

We obtain the same maximal theories independent of whether we use intuitionistic or classical FOL. Thus, the sentence $p \vee \neg p$ is denotationally valid in both cases. But it is constructively valid only in the classical case.

3.2.3 Logical morphisms

We now supplement logical theories with a notion of morphism:

Definition 3.35
Given $\mathcal{L} = (Syn, o, thm)$ and $\mathcal{L}' = (Syn', o', thm')$, a **logical morphism** $l : \mathcal{L} \to \mathcal{L}'$ consists of a morphism $l : Syn \to Syn'$ such that $l(thm[x]) = thm'[k[x]]$ for some expression $\vdash_{Syn', x:l(o)} k[x] : o'$.

k is uniquely determined if it exists so that it can be omitted from the notation.

Every $\mathcal{L}$-theory morphism is logical with $k[x]=x$ and therefore $l(o)=o$ and $l(thm)=thm$. More complex logical morphisms arise if $k[x]\neq x$:

EXAMPLE 3.36 (Model Theory as a Logical Morphism)
Using LF, we sketch a logical morphism from first-order logic $Syn=FOL$ to a logical theory ZF for axiomatic set theory.

ZF is a FOL-theory that declares the binary predicate $\in : i\to i\to o$ and adds the axioms of set theory. Besides the usual set theoretical operations, ZF defines in particular the two-element set $bool:i$ of Booleans. Moreover, we add a type constructor $Elem:i\to \mathtt{type}$ such that essentially $\vdash_{ZF} a:ElemA$ holds if $\vdash_{ZF} thm[a\in A]$. The complete definition of ZF can be found in [25].

Let $\Delta = univ:i, nonempty:thm[\exists x.x\in univ]$. Then we define $FOLZF:FOL\to ZF,\Delta$ by

- $FOLZF(i)=Elem\,univ$, i.e. $univ$ is an arbitrary non-empty set representing the universe of the model and terms are interpreted as elements of $univ$,
- $FOLZF(o)=Elem\,bool$, i.e. every formula is interpreted as a Boolean truth value,
- $FOLZF(thm)=\lambda x:Elem\,bool.thm[x\doteq 1]$, i.e. $thm\,F$ is interpreted as $FOLZF(F)$ being equal to the Boolean truth value 1.

Here, we have $k[x]=x\doteq 1$.

EXAMPLE 3.37 (Logic Translation as a Logical Morphism)
Using LF, we give a logical morphism from modal logic $Syn=ML$ to FOL.

The syntax of modal logic ML extends PL from Example 3.3 with $\Box:o\to o$ and $\diamond:o\to o$.

Let $\Delta = acc:i\to i\to o$. We define $MLFOL:ML\to FOL,\Delta$ by

- $MLFOL(o)=i\to o$, i.e. every modal formula is interpreted as a unary predicate on FOL-terms, which represent the worlds of a Kripke model,
- $MLFOL(\neg)=\lambda f:i\to o.\lambda x:o.\neg(f\,x)$, i.e. negation is interpreted world-wise,
- the other PL-connectives are translated accordingly,
- $MLFOL(\Box)=\lambda f:i\to o.\lambda x:o.\forall y.acc(x,y)\Rightarrow f(y)$, i.e. $MLFOL(\Box F)$ holds in x if $MLFOL(F)$ holds in all y that are accessible from x,
- $MLFOL(\diamond)$ is defined accordingly,
- $MLFOL(thm)=\lambda f:i\to o.thm\forall x.f\,x$, i.e. the truth of a modal formula is interpreted as the truth in all worlds.

Here, we have $k[f]=\forall x.f\,x$.

This leads naturally to a category structure:

THEOREM 3.38
Logical theories and logical morphisms form a category that inherits inclusions and pushouts from MMT.

PROOF. Identity and composition are as for MMT. The inclusions are the morphisms $(\Sigma,o,thm)\hookrightarrow(\Sigma,\Gamma,o,thm)$. If $\mathrm{dom}(\Gamma)$ and $\mathrm{dom}(\Sigma')$ are disjoint, the pushout of such an inclusion along l: $(\Sigma,o,thm)\to(\Sigma',o',thm')$ is $(\Sigma',l(\Gamma),o',thm')$.

We only have to show that all involved morphisms are logical. To make this precise, we write $K(l)[x]$ for the term $k[x]$ that is uniquely determined by a logical morphism l:

- The identity morphisms and inclusions are logical with $K(id_{(\Sigma,o,thm)})[x]=x$.
- Given two morphisms l_1 and l_2 with $K(l_i)=k_i$, the composition $l_2\circ l_1$ is logical with $K(l_2\circ l_1)[x]=k_2[l_2(k_1[x])]$.

- l^Σ is logical with $K(l^\Sigma) = K(l)$.
- Consider the diagram in Definition 2.26, seen as a diagram of logical theories. If all involved morphisms other than the unique factorization u are logical, then so is u with $K(u) = K(\varphi')$.

■

It is of particular interest whether a logical morphism is conservative—we will later relate this property to the completeness of a logic. In our framework, we can give two alternative definitions, one based on proofs and one based on models:

Definition 3.39 (Conservativity)
Consider a logical morphism $l : \mathcal{L} \to \mathcal{L}'$.
We say l is **proof-conservative** if for all $\mathcal{L}$-theories Σ and Σ-sentences F:

$$\text{if there is a } l(\Sigma)\text{-(dis)proof of } k[l^\Sigma(F)], \\ \text{then there is a } \Sigma\text{-(dis)proof of } F$$

We say l is **model-conservative** if for all $\mathcal{L}$-theories Σ and $\mathcal{L}$-theory morphisms $M : Syn, \Sigma \to Syn, \Gamma$:

$$\text{if } M \text{ is a } \Sigma\text{-model, then every } k[l^\Sigma(F)] \text{ is either true or false in } l(M).$$

$$\begin{tikzcd}
Syn, \Sigma \arrow[rrr, "l^\Sigma"] \arrow[dr, "M"] & & & Syn', l(\Sigma) \arrow[dl, "l(M)"'] \\
& Syn, \Gamma \arrow[r, "l^\Gamma"] & Syn', l(\Gamma) & \\
Syn \arrow[uu, hook] \arrow[ur] \arrow[rrr, "l"] & & & Syn' \arrow[uu, hook'] \arrow[ul]
\end{tikzcd}$$

Thus, proof-conservativity means to *reflect* (dis)proofs. (Like all well-formed morphisms, logical morphisms *preserve* (dis)proofs in any case.) Model-conservativity is a bit more complicated:

Remark 3.40 (Model-Conservativity)
Intuitively, model-conservativity means to preserve models. Therefore, one might expect the following simpler definition: if M is a Σ-model, then $l(M)$ is a $l(\Sigma)$-model. That condition would be stronger than the one we chose: it requires *every* $l(\Sigma)$-sentence to be determined in $l(M)$. Our condition only requires it for those sentences that are in the image of $k[l^\Sigma(-)]$.

The distinction is important because logical morphisms that do not map sentences surjectively are very common, e.g. the ones from Example 3.36 and Example 3.37. Intuitively, the $l(\Sigma)$-sentences that are not in the image are irrelevant from the perspective of $\mathcal{L}$. Therefore, our definition ignores them.

We have the following analogon to Theorem 3.33:

Theorem 3.41 (Conservativity)
In the situation of Definition 3.39:

1. If l is proof-conservative, then l is model-conservative.
2. If $\mathcal{L}$ is classical:
   if l is model-conservative, then l is proof-conservative.

Proof. Consider an $\mathcal{L}$-theory Σ.

1. Assume proof-conservativity and consider a model M. In general, every Σ-(dis)proof gives rise to a $l(\Sigma)$-(dis)proof. Therefore, we only have to show that no $k[l^\Sigma(F)]$ is over-determined

in $l(M)$. If some sentence were over-determined, then by proof-conservativity there would also be an F that is over-determined in M. That would violate the assumption that M is a model.

2. Assume model-conservativity and consider a proof p of $l^{\Sigma}(F)$ $(*)$. If Σ is inconsistent, $\vdash_{\Sigma} thm F$ holds anyway. So we can assume Σ is consistent. Proceeding indirectly, we assume $\not\vdash_{\Sigma} thm F$. Then $\Sigma, a:\overline{thm F}$ is consistent by Theorem 3.29 (using the classicality of $\mathcal{L}$) and by Theorem 3.31 has a model X, in which F is false. By restricting X, we obtain a model $M: Syn, \Sigma \hookrightarrow Syn, \Gamma$, in which F is false.
   Then $k[l^{\Sigma}(F)]$ is also false in $l(M)$; but by $(*)$ and Theorem 3.33, it must be true in $l(M)$. That contradicts model-conservativity.
   The case of disproofs proceeds analogously, except for not requiring classicality.

■

And we have the following important criterion for model-conservativity:

THEOREM 3.42 (Consistency Preservation)
l is model-conservative iff it preserves non-trivial consistency (i.e. $l(\Sigma)$ is consistent if Σ is non-trivial and consistent).

PROOF. Left-to-right: assume l is model-conservative and Σ is non-trivial and consistent.

- Σ can be extended to a maximal theory X by Theorem 3.31. $Syn, \Sigma \hookrightarrow Syn, X$ is a model and by model-conservativity $l(X)$ does not over-determine any sentence of the form $k[l^{\Sigma}(F)]$ $(*)$.
- Next we show indirectly that $l(X)$ is consistent. If $l(X)$ is inconsistent, it is also proper and by Theorem 3.24 all sentences are over-determined. Because Σ is non-trivial, this includes a sentence of the form $k[l^{\Sigma}(F)]$, which contradicts $(*)$.
- Finally, $l(\Sigma)$ is a subtheory of $l(X)$ and therefore also consistent.

Right-to-left: assume l preserves non-trivial consistency and $M: Syn, \Sigma \to Syn, \Gamma$ is a model. It suffices to show that no $k[l^{\Sigma}(F)]$ is over-determined in $l(M): Syn', l(\Sigma) \to Syn', l(\Gamma)$.

- If Σ is trivial, this holds vacuously.
- So assume Σ is non-trivial. It suffices to show that $l(M)$ is consistent. M is consistent; therefore by Theorem 3.24 also Γ; therefore by consistency preservation also $l(\Gamma)$; therefore by Theorem 3.24 also $l(M)$.

■

EXAMPLE 3.43 (Intuitionistic and Classical Logic)
The inclusion from *IPL* to *CPL* from Example 3.27 is not proof-conservative. If $\Sigma = p:o$, we have $\vdash_{CPL,\Sigma} thm(p \vee \neg p)$ but not $\vdash_{IPL,\Sigma} thm(p \vee \neg p)$.

But the inclusion is model-conservative. Indeed, it is well-known that the logical morphism $IPL \hookrightarrow CPL$ preserves consistency: $\vdash_{CPL,\Sigma} thm\bot$ is equivalent to $\vdash_{IPL,\Sigma} thm[\neg\neg\bot]$ and thus to $\vdash_{IPL,\Sigma} thm\bot$.

REMARK 3.44 (Non-Trivial Consistency)
The restriction to non-trivial Σ in Theorem 3.42 is necessary. For example, consider a logical theory $\mathcal{L}$ with an axiom $incon : thm \lightning$ but without any sentences. Then all non-trivial $\mathcal{L}$-theories are inconsistent and have no models.

Consequently, every logical morphism l out of $\mathcal{L}$ is model-conservative and preserves the consistency of non-trivial theories (independently of the codomain of l). But l can preserve the consistency of trivial theories only if its codomain is consistent.

3.2.4 Semantics through logical morphisms

Definitions 3.11 and 3.13 are simpler than needed in practice, and we will now build on them to develop more expressive definitions. To motivate our definitions, we first consider the two reasons why they are too simple.

Firstly, the constructive semantics must be allowed to depend on an inference system. We already discussed in Example 3.15 that we need inference rules to ensure all sentences are true or false. It is often very reasonable to simply assume the inference system to be a part of *Syn*. But occasionally, we do not want to do that, e.g. to use two different inference systems for the same syntax (e.g. classical and intuitionistic logic).

Secondly, we already discussed in Remark 3.17 that it is not enough to consider only theory morphisms $Syn,\Sigma \to Syn,\Gamma$. Instead, the denotational semantics must be allowed to use a rich language to define the models. In mathematics, this language is usually an implicitly assumed variant of axiomatic set theory. In formalized mathematics, more specific languages are used such as Tarski–Grothendieck set theory in Mizar [51], higher-order logic [18, 19, 38] or the calculus of constructions in Coq [10]. It could also be a programming language.

Both problems have a similar flavour: without an inference system, we do not have as many proofs as we want (possibly none); without a rich language for the models, we do not have as many models as we want (possibly none).

We can remedy both problems uniformly by using a logical morphism $sem : Syn \to Sem,\Delta$, where *Sem* is a logical theory *Sem* and Δ is a *Sem*-theory. Intuitively, *Sem* is a fixed, named language (e.g. *FOL* or *ZF*) and Δ provides some additional material needed for a specific *sem*.

Constructively, *Sem* extends *Syn* with an inference system, and *sem* is (typically) an inclusion. Then we can define proofs by using the expressions over *Sem* instead of *Syn*. Δ can provide additional rules or axioms that create variants of the inference system.

Denotationally, *Sem* defines the rich language in which to describe models, and Δ describes the common properties of all models. Then *sem* translates *Syn* into Sem,Δ, and we represent the interpretation function of a Σ-model as a morphism $Syn,\Sigma \to Sem,\Gamma$. The restriction of this morphism to *Syn* is the fixed interpretation of the logical identifiers in the semantic language *Sem*, and the model adds the interpretation of the non-logical identifiers in Σ.

This leads us to the following refinements of Definitions 3.11 and 3.13:

Definition 3.45 (Refined Constructive Semantics)
Consider a logical morphism $sem : Syn \to Sem,\Delta$ for a *Sem*-theory Δ, an $\mathcal{L}$-theory Σ, and a Σ-sentence F. Then:

1. A Σ-**(dis)proof** of F via *sem* is a $sem(\Sigma)$-(dis)proof of $k[sem^{\Sigma}(F)]$.
2. F is **constructively valid** via *sem* if there is a proof of F via *sem*.

Definition 3.46 (Refined Denotational Semantics)
Consider a logical morphism $sem : Syn \to Sem,\Delta$ for a *Sem*-theory Δ, an $\mathcal{L}$-theory Σ, and a Σ-sentence F. Then:

1. A Σ-**premodel** via *sem* is a *Sem*-theory morphism $M : Sem,\Delta,sem(\Sigma) \to Sem,\Gamma$ such that $M|_{Sem} = id_{Sem}$.
2. F is **true** (**false**) in a premodel M via *sem* if $k[sem^{\Sigma}(F)]$ is true (false) in M.
3. M is a **model** via *sem* if every Σ-sentence is either true or false in M.
4. F is **denotationally valid** via *sem* if it is true in all models via *sem*.

$$\begin{tikzcd}
Syn,\Sigma \arrow[r, "sem^{\Sigma}"] & Sem,\Delta,sem(\Sigma) \arrow[r, "M"] & Sem,\Gamma \\
Syn \arrow[u, hook] \arrow[r, "sem"] & Sem,\Delta \arrow[u, hook] & Sem \arrow[u, hook] \arrow[l, hook']
\end{tikzcd}$$

NOTATION 3.47 (Namespaces)
When giving logical morphisms $sem : Syn \to Sem, \Delta$, we usually find that *sem*, *Syn*, and *Sem* are named theories/morphisms, whereas Δ is an anonymous list of declarations. Moreover, Δ is usually given together with *sem*. This can be seen very well in Examples 3.36 and 3.37.

Therefore, we can assume that all identifiers in Δ are implicitly global and reside in the namespace of *sem*. That ensures that the pushout of Σ along *sem*, as used in Definition 3.46, is defined.

THEOREM 3.48
In the situation of Definitions 3.45 and 3.46:

1. F is constructively valid via *sem* iff $k[sem^{\Sigma}(F)]$ is constructively valid.
2. F is denotationally valid via *sem* iff $k[sem^{\Sigma}(F)]$ is denotationally valid.

Here, $k[sem^{\Sigma}(F)]$ is a sentence of the *Sem*-theory $\Delta, sem(\Sigma)$.

PROOF. 1. This is just a reformulation of the definition.

2. Left-to-right: this follows after observing that, by definition, a model of the *Sem*-theory $\Delta, sem(\Sigma)$ is also a model of Σ via *sem*.
   Right-to-left: a model M of Σ via *sem* is not necessarily a model of $\Delta, sem(\Sigma)$ because sentences that are not in the image of $k[sem^{\Sigma}(-)]$ may be over-determined or undetermined in M.
   Over-determination is not problematic because it would make M inconsistent. If Σ is non-trivial, that contradicts the assumption that M is a model of Σ via *sem*. If Σ is trivial, the theorem holds vacuously anyway.
   But it is possible that M is consistent and leaves sentences undetermined. In that case, we extend M to a model M' of $\Delta, sem(\Sigma)$ by Theorem 3.31. Then the implication is easy to prove.

■

EXAMPLE 3.49 (Refined Constructive Semantics)
Consider Example 3.37 and assume that the definition of *ML* contains only the syntax and *FOL* also contains declarations of proof rules. Then *ML* has no constructive theorems because it has no proofs.

The morphism $MLFOL : ML \to FOL, \Delta$ refines the constructive semantics of *ML* by using the proof system of *FOL* to define a proof system for *ML*. Note how the *FOL*-theory Δ sets up the accessibility relation needed to express the translation.

EXAMPLE 3.50 (Refined Denotational Semantics)
Consider Example 3.36 and let Σ be the *FOL*-theory of monoids from Example 3.5. A model M of Σ via *FOLZF* is a *ZF*-morphism $ZF, \Delta, FOLZF(\Sigma) \to ZF, \Gamma$.

Note that M is a *Sem*-theory morphisms and not a *Sem*, Δ-theory morphism. Thus, M must provide cases for *univ* and *non-empty* declared in Δ and for $\circ$, e, etc. declared in $FOLZF(\Sigma)$. This corresponds exactly to the usual *FOL*-models given as a pair of a non-empty set and an interpretation function that satisfies the axioms.

Note that we recover Definitions 3.11 and 3.13 as the special cases of Definitions 3.45 and 3.46 where $sem = id_{Syn}$, in which case the premodels are simply the $\mathcal{L}$-theory morphisms. The case $sem = id_{Syn}$ can be seen as the initial semantics. (Categorically, it is indeed an initial object in the slice category of morphisms out of Syn.)

Therefore, in the sequel, we only consider Definitions 3.45 and 3.46. Moreover, we drop the qualifier 'via *sem*' if *sem* is clear from the context.

The intuition of being an initial semantics is fortified by the following theorem:

THEOREM 3.51 (Preservation of Proofs and Truth)
Consider logical morphisms $sem : Syn \to Sem, \Delta$ and $sem' = r^{\Delta} \circ sem$ for some $r : Sem \to Sem', \Delta'$. Then:

1. If there is a Σ-(dis)proof of F via *sem*, there is a Σ-(dis)proof of F via sem'.
2. If F is true (false) in a Σ-premodel M via *sem*, then F is true (false) in the Σ-premodel $r(M)$ via sem'.

PROOF. 1. If p is the (dis)proof via *sem*, then $r^{sem(\Sigma)}(p)$ is a (dis)proof via sem'.

2. For $M : Sem, \Delta, sem(\Sigma) \to Sem, \Gamma$, the premodel $r(M)$ is the universal morphism out of the pushout $Sem', \Delta', r(\Delta, sem(\Sigma))$ as in the diagram below. Note that in this diagram all rectangles are pushouts, and the diagram commutes because the pushouts are coherent. The preservation of truth and falsity follows by moving the witnesses along the arrows of the diagram.

$$\begin{tikzcd}
Syn, \Sigma \arrow[r, "sem^{\Sigma}"] \arrow[rrrr, bend left, "sem'^{\Sigma}"] & Sem, \Delta, sem(\Sigma) \arrow[rrr, "r^{\Delta, sem(\Sigma)}"] \arrow[dr, "M"] & & & Sem', \Delta', r(\Delta, sem(\Sigma)) = Sem', \Delta', r(\Delta), sem'(\Sigma) \arrow[dl, "r(M)"'] \\
& & Sem, \Gamma \arrow[r, "r^{\Gamma}"] & Sem', \Delta', r(\Gamma) & \\
& & Sem \arrow[u, hook] \arrow[r, "r"] \arrow[dl, hook] & Sem', \Delta' \arrow[u, hook] \arrow[dr] & \\
Syn \arrow[uuu, hook] \arrow[r, "sem"] \arrow[rrrr, bend right, "sem'"'] & Sem, \Delta \arrow[uuu, hook] \arrow[rrr, "r^{\Delta}"] & & & Sem', \Delta', r(\Delta) \arrow[uuu, hook]
\end{tikzcd}$$

At this point the proof is complete, but it is worth pointing out that the codomain of $r(M)$ always contains Δ', whereas an arbitrary Σ-premodel via sem' is a morphism $M' : Sem', \Delta', r(\Delta), sem'(\Sigma) \to Sem', \Gamma'$ for some Γ'. If we call those M' that factor through $r(M)$ the refinements of M, then truth (falsity) in M implies truth (falsity) in any refinement of M. ■

The preservation of proofs means that constructive validity is preserved by refinements r. Similarly, the preservation of truth/falsity means that counter-examples are preserved so that denotational validity is reflected. The latter statement has one caveat though: if we refine too much, i.e. if the codomain becomes too strong, we may map a model M to an inconsistent premodel $r(M)$, i.e. truth/falsity is preserved, but not necessarily the property of being a model. The following theorem makes this precise:

THEOREM 3.52 (Preservation/Reflection of Semantics)
Consider the situation of Theorem 3.51.

1. If F is constructively valid via *sem*, then F is constructively valid via *sem*$'$.
2. If r is model-conservative:
   (a) If M is a model via *sem*, then $r(M)$ is a model via *sem*$'$ and makes true the same sentences as M.
   (b) If F is denotationally valid via *sem*$'$, then F is denotationally valid via *sem*.

PROOF. 1. This follows immediately from Theorem 3.51.
2. Assume r is model-conservative:
   (a) Assume a model M via *sem*. By Theorem 3.51, every sentence that is true (false) in M is also true (false) in $r(M)$. We only have to show that no Σ-sentence is both true and false in $r(M)$. That follows from the consistency of M, which permits applying Theorem 3.31 to extend M to a model of $\Delta, sem(\Sigma)$, and the model-conservativity of r.
   (b) This is proved indirectly: if there were a model via *sem* that makes F false, then by (2a) there would also be one via *sem*$'$. ■

3.3 *Logics are pairs of syntax and semantics*

We can finally define logics for some fixed logical framework:

DEFINITION 3.53 (Logic)
A **logic** consists of a logical theory $\mathcal{L} = (Syn, o, thm)$ and a logical morphism $sem : Syn \to Sem, \Delta$.

We think of *Syn* as providing the syntax and of *sem* as providing the semantics. In the following, we describe three different perspectives within this general intuition. All three are reasonable, but they are arranged to lead towards the last one, which we will subscribe to in the following sections.

Logics as Pairs of Proof and Model Theory. If we want to work with a constructive and a denotational semantics at the same time, we can use a span of two logical morphisms:

DEFINITION 3.54 (Bilogic)
A **bilogic** consists of a logical theory $\mathcal{L}$ and two logical morphisms $pf : Syn \to Pf$ and $mod : Syn \to Mod, \Delta$. We call *Syn* the **syntax**, *Pf* the **proof theory**, and *Mod*, Δ the **model theory**.

Here, we use an *Mod*-theory Δ for the model theory, but assume the analogous *Pf*-theory to be empty.

$$\begin{tikzcd} & Pf \\ Syn \arrow[ur, "pf"] \arrow[dr, "mod"'] & \\ & Mod, \Delta \end{tikzcd}$$

DEFINITION 3.55 (Soundess/Completeness)
We say that a bilogic is **sound** if for all theories Σ and all sentences F: if F is constructively valid via *pf*, then it is denotationally valid via *mod*. And we say it is **complete** if the opposite implication holds.

This has the appeal that proof theory and model theory are treated symmetrically. Thus, the same syntax can be combined with a diverse set of model and proof theories.

We can now derive general criteria for soundness and completeness:

THEOREM 3.56 (Soundness/Completeness)
In the situation of Definition 3.54, consider a morphism $r: Pf \to Mod, \Delta$ such that $r \circ pf = mod$. Then:

1. (Syn, pf, mod) is sound,
2. if Pf is classical and r is model-conservative, then (Syn, pf, mod) is complete.

PROOF. Both results follow by combining the respective cases of Theorems 3.52, 3.48 and 3.33. ■

$$
\begin{tikzcd}
 & Pf \arrow[dd, "r"] \\
Syn \arrow[ur, "pf"] \arrow[dr, "mod"'] & \\
 & Mod, \Delta
\end{tikzcd}
$$

EXAMPLE 3.57 (First-Order Logic)
We can combine Examples 3.3, 3.27 (enriched with appropriate proof rules for the quantifiers and equality) and 3.36 to obtain first-order logic as a bilogic.

This is essentially the representation developed in [23], which also gives a morphism r that witnesses the soundness according to Theorem 3.56.

If we use the classical variant of the *FOL* proof theory, then we see from Theorems 3.41, and 3.56 that the completeness of *FOL* is reduced to r being proof-conservative.

Proof Theory as Initial Semantics. In a bilogic, $pf: Syn \to Pf$ is typically an inclusion morphism, and declarative logical frameworks are very good at representing *Syn* and *Pf* together. Moreover, inductive arguments often treat syntax and proof theory similarly. Therefore, it is possible to couple *Syn* and *Pf* more tightly than *Syn* and *Mod*.

Concretely, we can assume that the inference system is already a part of the theory *Syn* so that *Pf* is not needed. Then we can simply think of logics $(\mathcal{L}, sem: Syn \to Sem, \Delta)$ as follows:

- *Syn* defines both the syntax and the proof theory;
- the semantics via id_{Syn} is the initial semantics, i.e. it defines the proofs and premodels that are present irrespective of the model theory;
- *Sem* defines the rich language in which models are formulated, e.g. a more expressive logic or set theory;
- Δ is a *Sem*-theory that specifies the structure of models; and
- *sem* interprets the syntax and proves soundness at the same time.

$$
\begin{tikzcd}
Syn, \Sigma \arrow[r, "sem^{\Sigma}"] & Sem, \Delta, sem(\Sigma) \arrow[r, "M"] & Sem, \Gamma \\
Syn \arrow[u, hook] \arrow[r, "sem"] & Sem, \Delta \arrow[u, hook] & Sem \arrow[l, hook'] \arrow[u, hook]
\end{tikzcd}
$$

This asymmetric perspective in which the proof theory is primary was strongly influenced by [35] and [1]. It is closely related to the following asymmetry: Soundness is usually easier and more important than completeness. Concretely, we have:

DEFINITION 3.58 (Soundness/Completeness)
We say that a logic $(\mathcal{L}, sem)$ is **sound/complete** if the bilogic $(\mathcal{L}, id_{Syn}, sem)$ is.

THEOREM 3.59 (Soundness/Completeness)
Every logic $(\mathcal{L}, sem : Syn \to Sem)$ is sound in the sense of Definition 3.58. If Syn is classical and sem is model-conservative, it is also complete.

PROOF. This is a special case of Theorem 3.56. ■

Semantics as a Chain of Refinements. Expanding on the idea that proof theory is the initial semantics, we can consider different model theories and refinements between them. For example, we can give chains of logical morphism $sem_i : Sem_{i-1} \to Sem_i$ (with $Sem_0 = Syn$) that interpret the syntax in increasingly richer languages Sem_i. Along a chain of logical morphisms, the distinction between syntax and semantics can become blurred because every Sem_i is a language that interprets Sem_{i-1} and is itself interpreted by Sem_{i+1}.

EXAMPLE 3.60
Higher-order logic HOL is sufficient to define the model theory of first-order logic FOL. The semantics of HOL itself can be defined by a logical morphism $HOL \to ZF$. Set theory itself is actually a family of increasingly richer languages including, e.g. refinements of ZF to ZF with choice or to ZF with large cardinals:

$$\begin{tikzcd}
 & & & ZF + Ch \\
FOL \arrow[r, "sem_1"] & HOL \arrow[r, "sem_2"] & ZF \arrow[ur, hook] \arrow[dr, hook] & \\
 & & & ZF + LC
\end{tikzcd}$$

We give such logical morphisms $FOL \to HOL \to ZF$ in [23] using the logical framework LF. Corresponding developments can be done in Isabelle, where Isabelle/HOL [38] is usually used to describe models of a logic and is translated to Isabelle/ZF in [28].

We hold that this perspective adequately captures the mathematical practice of choosing formal languages. Instead of fixing one syntax and one semantics, we have a multi-graph of formal languages at different degrees of expressivity.

The formalization of a non-logical theory should always be done relative to the weakest possible logical theory in this multi-graph. Then pushouts can be used to move the non-logical theory along refinements.

Finally, the usual model theory defined in terms of an implicit platonic universe can be understood as the hypothetical colimit of an underspecified infinite multi-graph. Using countable theories, the platonic universe itself can only be formalized approximately. But we can refine our approximate formalizations as needed until, in the hypothetical colimit, we obtain the universe.

4 What is a logic translation?

4.1 Translations are theory morphisms

Translations. We now define translations between logics as defined in Definition 3.53. Throughout this section, we assume two logics $L = (\mathcal{L}, sem : Syn \to Sem, \Delta)$ and $L' = (\mathcal{L}', sem' : Syn' \to Sem, \Delta')$.

We follow the asymmetric perspective from Section 3.3, i.e. the inference system (if any) is already part of the syntax.

It is reasonable to use the same *Sem* for L and L' because *Sem* is usually assumed as a fixed background language. The differences between the models of L and L' are captured by the *Sem*-theories Δ and Δ'.

The main idea is that translations are logical morphisms:

DEFINITION 4.1 (Translations)
A **syntax translation** T is a logical morphism $Syn \to Syn'$. In that case, we abbreviate $sem^* = sem' \circ T$.

A **semantics translation** (T,t) additionally provides a family t of *Sem*-morphisms $t_\Sigma : Sem, \Delta, sem(\Sigma) \to Sem, \Delta', sem^*(\Sigma)$ indexed by $\mathcal{L}$-theories Σ such that for every *Syn*-theory morphisms $\sigma : \Sigma_1 \to \Sigma_2$, we have $t_{\Sigma_2} \circ sem(\sigma) = sem^*(\sigma) \circ t_{\Sigma_1}$.

The situation of Definition 4.1 leads to the following diagram where $\varnothing$ refers to the empty $\mathcal{L}$-theory. Here, the four trapezoids and the triangle commute, but not necessarily the central and outer rectangle.

$$\begin{tikzcd}
 & & Sem \arrow[dll, hook'] \arrow[drr, hook] & & \\
Sem, \Delta \arrow[rrrr, "t_\varnothing"] \arrow[dr, hook] & & & & Sem, \Delta' \arrow[dl, hook'] \\
 & Sem, \Delta, sem(\Sigma) \arrow[rr, "t_\Sigma"] & & Sem, \Delta', sem^*(\Sigma) & \\
 & Syn, \Sigma \arrow[u, "sem^\Sigma"] \arrow[rr, "T^\Sigma"] & & Syn', T(\Sigma) \arrow[u, "sem'^{T(\Sigma)}"'] & \\
Syn \arrow[uuu, "sem"] \arrow[ur, hook] \arrow[rrrr, "T"] & & & & Syn' \arrow[uuu, "sem'"'] \arrow[ul, hook']
\end{tikzcd}$$

REMARK 4.2 (Categorical Interpretation)
We can think of a logical morphism $l : A \to B$ as inducing a functor from A-theories to B-theories. Then *sem* and sem^* induce functors from *Syn*-theories to *Sem*-theories, and t induces a natural transformation between them.

Let us now fix a translation $(T,t) : L \to L'$ as in Definition 4.1. The homomorphic extension $T^\Sigma(-)$ translates L-Σ-sentences and proofs to L'-$T(\Sigma)$-sentences and proofs. Similarly, pre-composition with t_Σ translates L'-$T(\Sigma)$-models via sem' to L-Σ-models via *sem*.

DEFINITION 4.3
(T,t) is called **sound** if for all $\mathcal{L}$-theories Σ and Σ-sentences F, if F is denotationally valid via sem^*, then it is denotationally valid via *sem*. (T,t) is called **complete** if the opposite implication holds.

A translation is particularly well behaved if the above diagram commutes. Then, e.g. it is sufficient to give T and $t_\varnothing$ because the requirement of commutativity determines the remaining t_Σ as universal morphisms out of the pushout $sem(\Sigma)$. Moreover, we immediately obtain completeness because each model via sem^* induces a model via *sem*.

However, such commuting translations occur rarely. The most important examples are inclusion morphisms $T : Syn \hookrightarrow Syn'$ with $sem = sem'|_{Syn}$, e.g. if $T : PL \hookrightarrow FOL$ and $sem' = FOLZF$.

In general, commutativity is not necessary for sound and complete translations, and many interesting translations do not satisfy it. Therefore, we establish stronger criteria in the following. The basic idea is to relate the judgements $\vdash_{sem(\Sigma)} sem^{\Sigma}(thm\,F)$ and $\vdash_{sem^*(\Sigma)} sem^{*\Sigma}(thm\,F)$ for arbitrary $\vdash_{\Sigma} F:o$. If one of them implies the other, we can leverage that to obtain soundness or completeness. But establishing such an implication usually requires a difficult induction on not just F but on all Σ-expressions. Formalizing the recipe of these inductions leads us to logical relations.

Logical Relations. The method of logical relations picks the right induction hypothesis and takes care of the bureaucracy of certain inductive arguments.

A logical relation r is similar to a theory morphism in that it maps Σ-identifiers to Σ'-expressions and is extended compositionally to a map $\overline{r}$ of all Σ-expressions. In particular, if Σ is finite, so is r. Moreover if type-checking in Σ' is decidable, so is the property of r being a logical relation. However, contrary to theory morphisms, $\overline{r}$ cannot be defined generically for arbitrary expressions: For every constructor C, a separate insight is needed to define $\overline{r}(C(\Gamma;A_1,\ldots,A_n))$.

Therefore, it is difficult to define $\overline{r}$ even for a single logical framework, and we currently do not know how to define it for an arbitrary one. For example, [46] is concerned exclusively with formulating logical relations for LF. Here we briefly define the main concepts abstractly to state our criteria.

DEFINITION 4.4 (Logical Relations)
Given two MMT theory morphisms $l,m:\Sigma\to\Sigma'$, a *relation* r between l and m maps every identifier $c\in\mathrm{dom}(\Sigma)$ to a Σ'-expression $r(c)$.

A logical framework *has relations* if it defines for every r an extension $\overline{r}$ that maps all Σ-expressions to Σ'-expressions.

A relation r is called a **logical relation** if

$$c:T\text{ in }\Sigma \qquad \text{implies} \qquad \vdash_{\Sigma'} r(c):\overline{r}(T)[l(c),m(c)].$$

A logical framework **has logical relations** if all logical relations satisfy that

$$\vdash_{\Sigma} E:T \qquad \text{implies} \qquad \vdash_{\Sigma'} \overline{r}(E):\overline{r}(T)[l(E),m(E)].$$

Having logical relations means to admit a certain induction principle. The expressions $r(c)$ correspond to the cases of the inductive argument. And the preservation property of Definition 4.4 corresponds to the induction hypothesis. Therefore, we can use logical relations to formalize many inductive proofs about logics.

EXAMPLE 4.5 (Logical Relations for LF)
[46] defines $\overline{r}$ for LF. We only give the most important case, which captures the essence of logical relations:

$$\overline{r}(\Pi x:A.B)[f,g]=\Pi x:l(A),y:m(A).\Pi q:\overline{r}(A)[x,y].(\overline{r}(B)\,x\,y\,q)[f\,x,g\,y]$$

Intuitively, two functions f and g are related if they map related arguments x and y to related results $f\,x$ and $g\,y$.

EXAMPLE 4.6 (Logical Relations for Pure)
We can define logical relations for Pure from Example 2.38 in essentially the same way as for LF. In particular:

$$\overline{r}(A\to B)[f,g]=\forall x:l(A),y:m(A).\overline{r}(A)[x,y]\Longrightarrow\overline{r}(B)[f\,x,g\,y]$$

Again two functions f and g are related at $A\to B$ if they map A-related arguments to B-related results.

Verifying Translations Using Logical Relations. We can now get back to our original goal. Using LF as the logical framework, we plan to give a family of logical relations r_Σ between $l = t_\Sigma \circ sem^\Sigma$ and $m = sem^{*\Sigma} = sem'^{T(\Sigma)} \circ T^\Sigma$. The property of logical relations guarantees that if $\vdash_\Sigma F : o$, then $\overline{r_\Sigma}(F)$ is a proof that $r_\Sigma(o)$ holds about $l(F)$ and $m(F)$.

Thus, depending on how we define the base case $r_\Sigma(o)$, we can use r to show different properties:

THEOREM 4.7 (Completeness)
Assume each r_Σ is a logical relation such that

$$\overline{r_\Sigma}(o)[x,y] = (t_\Sigma \circ sem^\Sigma)(thm)x \Longrightarrow sem^{*\Sigma}(thm)y$$

Then (T,t) is complete.

PROOF. Assume Σ and f such that f is denotationally valid via sem. Assume a model M': $Sem, \Delta, sem^*(\Sigma) \to Sem, \Gamma$ via sem^*. Then $M = M' \circ t_\Sigma$ is a Σ-model via sem. By assumption, f is true in M, i.e. $\vdash_{Sem,\Gamma} M'(t_\Sigma(sem^\Sigma(thmf))$.

Applying M' to the logical relation at f yields

$$\vdash_{Sem,\Gamma} M'(\overline{r_\Sigma}(f)) : (M' \circ t_\Sigma \circ sem^\Sigma)(thmf) \Longrightarrow (M' \circ sem^{*\Sigma})(thmf)$$

Therefore, $\vdash_{Sem,\Gamma} M'(sem^{*\Sigma}(thmf))$, i.e. f is true in M'. Thus, f is denotationally valid via sem^*. ■

THEOREM 4.8 (Soundness)
Assume each r_Σ is a logical relation such that

$$\overline{r_\Sigma}(o)[x,y] = sem^{*\Sigma}(thm)y \Longrightarrow (t_\Sigma \circ sem^\Sigma)(thm)x$$

If L' is complete and every t_Σ is proof-conservative, then (T,t) is sound.

PROOF. Assume a Syn-theory Σ and a Σ-sentence f such that f is denotationally valid via sem^*. Let $thm'f' = T^\Sigma(thmf)$ and $ThmF = sem^{*\Sigma}(thmf)$, i.e. f' is the $T(\Sigma)$-sentence and F the $sem^*(\Sigma)$-sentence that arise by translating f along T and sem^*, respectively. Because every $T(\Sigma)$-model via sem' is also a Σ-model via sem^*, f' is denotationally valid via sem'.

Because L' is complete, we have $\vdash_{Syn',T(\Sigma)} thm'f'$. Thus, by applying $sem'^{T(\Sigma)}$, we obtain $\vdash_{Sem,\Delta',sem^*(\Sigma)} ThmF$. By applying the logical relation at f, we obtain $\vdash_{Sem,\Delta',sem^*(\Sigma)}$ $t_\Sigma(sem^\Sigma(thmf))$.

Thus, by the proof-conservativity of t_Σ, also $\vdash_{Sem,\Delta,sem(\Sigma)} sem^\Sigma(thmf)$. Therefore, f is true in every Σ-model via sem, i.e. f is denotationally valid via sem. ■

The proof of Theorem 4.8 would be easier if we assumed T to be proof-conservative instead of the t_Σ. But that is impractically strong: it fails if we do not have any proof system for Syn at all, or if Syn and Syn' are too different to reflect the proofs. Essentially, showing the proof-conservativity of T is as hard as showing the soundness without using Theorem 4.8. But proving proof-conservativity of the t_Σ can be very feasible because Sem, Δ and Sem, Δ' are often much more similar than Syn and Syn'.

For example, [47] studies the morphism $T = MLFOL$ from Example 3.37. Even if we add an inference system to ML, which is not always desirable or easy, the proof-conservativity of $MLFOL$ is very hard to show. But the resulting t_Σ are rather simple, and their proof-conservativity can be shown by giving morphisms in the opposite direction.

Remark 4.9 (Future Work)
In [46], we also show how the Twelf tool [41] can mechanically verify logical relations. This is important because the involved inductions grow difficult very quickly if the complexity of L increases. However, to apply it to our criteria, we still need a way to obtain each r_Σ uniformly from a single logical relation between $t_\varnothing \circ sem$ and sem^*.

Similarly, we need a finitary way to give the families t_Σ by induction on Σ and verify their proof-conservativity uniformly.

4.2 *Semantics and translations are the same thing*

We have now unified the semantics of a logic and logic translations out of it—both are special cases of logical morphisms. Specifically, Definition 3.53 defines semantics in terms of logical morphisms $sem : Syn \to Sem$, and Definition 4.1 defines syntax and semantics translations in terms of logical morphisms $T : Syn \to Syn'$ for translations.

Thus, the difference between them becomes a matter of pragmatics: if we think of the codomain of a morphism out of *Syn* as a logic, we speak of translations; otherwise, we speak of semantics.

Example 4.10 (Kripke Models for Intuitionistic Logic)
Let *Preord* be the *FOL*-theory of pre-orders using a relation $\leq : i \to i \to o$.

We formalize the semantics of *IPL* from Example 3.27 in terms of Kripke models by giving a morphism $IPLFOL : IPL \to FOL, Preord$. *IPLFOL* is similar to *MLFOL*, and we also put

$$IPLFOL(o) = i \to o \qquad \text{and} \qquad IPLFOL(thm) = \lambda f : i \to o.thm[\forall x.f\,x]$$

Now some connectives are interpreted world-wise, e.g. the conjunction $f \wedge g$ is true at world w if f and g are:

$$IPLFOL(\wedge) = \lambda f, g : i \to o.\lambda w : i.(f\,w) \wedge (g\,w)$$

Others quantify over the future of the current world, e.g. the negation $\neg f$ is true at w if it is true at all worlds above w:

$$IPLFOL(\neg) = \lambda f : i \to o.\lambda w : i.\forall x.w \leq x \Rightarrow \neg(f\,x)$$

This defines the semantics of *IPL* in terms of *FOL*. Thus, *IPLFOL* combines features of a syntax translation and of semantics.

One might prefer, by the way, to give a morphism into set theory instead. For example, continuing Example 3.36, one might want to have a morphism $IPLKrZF : IPL \to ZF, FOLZF(Preord)$. But note that *IPLFOL* is much easier to define than *IPLKrZF* (because formal reasoning in *FOL* is easier than reasoning in *ZF*). Moreover, we can define *IPLKrZF* from *IPLFOL* but not the other way round: $IPLKrZF = FOLZF^{Preord} \circ IPLFOL$.

$$\begin{tikzcd}
IPL \arrow[dr, "IPLFOL"'] & FOL \arrow[d, hook] \arrow[r, "FOLZF"] & ZF \arrow[d, hook] \\
 & FOL, Preord \arrow[r, "FOLZF^{Preord}"] & ZF, FOLZF(Preord)
\end{tikzcd}$$

4.3 Identity is isomorphism up to extensionality

Once we have defined translations between logics, it is straightforward to define identity/equivalence of logics as isomorphism. However, this requirement is too strong because intensional logical frameworks like LF make very few logical theories isomorphic.

EXAMPLE 4.11 (Intensionally Non-Isomorphic)
Consider two variants $CPL^{\vee\neg}$ and $CPL^{\wedge\neg}$ of classical propositional logic from Example 3.27 that only use the indicated connectives. Intuitively, these are isomorphic via the de Morgan identities. Formally, we give logical morphisms $d_1 : CPL^{\vee\neg} \to CPL^{\wedge\neg}$ and $d_2 : CPL^{\wedge\neg} \to CPL^{\vee\neg}$, which map in particular $d_1(o) = o$ and $d_2(o) = o$, $d_1(thm) = thm$ and $d_2(thm) = thm$, as well as

$$d_1(\vee) = \lambda x, y.\neg(\neg x \wedge \neg y) \quad \text{and} \quad d_2(\wedge) = \lambda x, y.\neg(\neg x \vee \neg y).$$

Defining d_1 and d_2 for the proof rules is straightforward.

However, even after β-reduction in LF, we obtain $d_2(d_1(\vee)) = \lambda x, y.\neg\neg(\neg\neg x \vee \neg\neg y)$, which is equivalent but not equal to $\vee$ over $CPL^{\vee\neg}$.

Therefore, we use a more general definition that permits quotienting out an extensional equality. Let us assume a logical framework with logical relations. Then, as indicated for LF in [46], we can represent extensional equality as a logical relation on the identity morphism:

DEFINITION 4.12 (Extensionality)
An **extensional equality** for a logical theory Syn is a logical relation $\equiv$ on id_{Syn} and id_{Syn} such that $\equiv_A$ is an equivalence relation for all types A.

We will write $x \equiv_A y$ for $\equiv_A [x, y]$.

Given extensional equalities $\equiv$ for Syn and $\equiv'$ for Syn', a logical morphism $l : Syn \to Syn'$ **preserves** extensionality if $\vdash_{Syn} E_1 \equiv_A E_2$ implies $\vdash_{Syn'} l(E_1) \equiv'_{l(A)} l(E_2)$.

We do not have to require the closure of $\equiv$ under substitution because it is inherited from the properties of logical relations. Moreover, for many cases, in particular LF and Pure, it is enough to show that $\equiv_a$ is an equivalence for all atomic types a. This implies the equivalence properties for all types A.

EXAMPLE 4.13 (Extensionality for PL, FOL, ZF)
To define extensionality for PL from Example 3.3, we put in particular

$$x \equiv_o y = thm[x \Rightarrow y \wedge y \Rightarrow x] \quad \text{and} \quad p \equiv_{thm F} q = thm\top$$

This identifies formulas up to provable equivalence and identifies all proofs.

For FOL, we use additionally

$$x \equiv_i y = thm[x \doteq y]$$

which identifies terms up to provable equality.

Clearly, all three are equivalence relations. For composed types A, we obtain equivalence relations $\equiv_A$ from LF.

We also have to define $\equiv_c$ for the remaining identifiers c. That amounts to proving that all operations of FOL preserve $\equiv$. That is also straightforward.

The usual FOL-theories never add new base types so that the above also yields extensional equality for every FOL-theory. In particular, we obtain the usual notion of extensional equality for ZF from Example 3.36: Sets are extensionally equal if they are provably equal using the rules of FOL and the axioms of set theory.

It is now straightforward to define extensional equality of morphisms:

Definition 4.14 (Extensional Isomorphism)
Assume we have fixed extensional equalities $\equiv$ and $\equiv'$ for Syn and Syn'.

Two morphisms $f,g:Syn \to Syn'$ are **extensionally equal** if $\vdash_{Syn'} f(c) \equiv'_{f(T)} g(c)$ for all declarations $c:T$ in Syn.

Syn and Syn' are **extensionally isomorphic** if there are extensionality-preserving morphisms $f: Syn \to Syn'$ and $g:Syn' \to Syn$ such that $g \circ f$ and id_{Syn} as well as $f \circ g$ and $id_{Syn'}$ are extensionally equal.

Extensional equality implies $\vdash_{Syn'} f(E) \equiv'_{f(T)} g(E')$ whenever $\vdash_{Syn} E:T$ due to the properties of logical relations.

Definition 4.15 (Equivalence of Logics)
Assume we have fixed extensional equalities for Syn, Syn', and Sem (in a way that induces extensional equality for all Sem-theories). Consider two logics $(\mathcal{L}, sem:Syn \to Sem, \Delta)$ and $(\mathcal{L}', sem':Syn' \to Sem, \Delta')$ such that sem and sem' preserve extensionality.

Then $\mathcal{L}$ and $\mathcal{L}'$ are **equivalent** if there are extensionality-preserving f,g,F,G such that the following diagram commutes up to extensional equality. In particular, both Syn and Syn' as well as Sem, Δ and Sem, Δ' are extensionally isomorphic.

$$\begin{tikzcd}
& Sem \arrow[dl, hook] \arrow[dr, hook] & \\
Sem, \Delta \arrow[rr, bend left, "F"] & & Sem, \Delta' \arrow[ll, bend left, "G"] \\
Syn \arrow[u, "sem"] \arrow[rr, bend left, "f"] & & Syn' \arrow[u, "sem'"'] \arrow[ll, bend left, "g"]
\end{tikzcd}$$

Example 4.16 (Extensionally Isomorphic)
We continue Example 4.11 and show that the two logics are equivalent in the sense of Definition 4.15.

We use the respective extensional equalities from Example 4.13 for $Syn = CPL^{\vee\neg}$ and $Syn' = CPL^{\wedge\neg}$. We show that $f = d_1$ and $g = d_2$ are a pair of extensional isomorphisms. First, because both morphisms map o and thm to themselves, it is straightforward to show that both preserve extensionality. Secondly, we have to show that $d_2 \circ d_1 \equiv id_{CPL^{\vee\neg}}$. The crucial case is to show $\vdash_{CPL^{\vee\neg}} \vee \equiv_{o\to o\to o} d_2(d_1(\vee))$. This means to show that if $x \equiv_o x'$ and $y \equiv_o y'$, then $x \vee y \equiv_o \neg\neg(\neg\neg x' \vee \neg\neg y')$, which is straightforward. We show $d_1 \circ d_2 \equiv id_{CPL^{\wedge\neg}}$ accordingly.

We use the extensional equality from Example 4.13 for $Sem = ZF$, and put $F = G = id_{ZF}$. Let $sem = CPLZF^{\vee\neg} : CPL^{\vee\neg} \to ZF$ and $sem' = CPLZF^{\wedge\neg} : CPL^{\wedge\neg} \to ZF$ be the appropriate restrictions of $FOLZF$ (i.e. $\Delta = \Delta' = \cdot$). Both preserve extensionality: we only have to show that they map provably equivalent sentences to provably equal ZF-Booleans.

Finally, we check that $CPLZF^{\wedge\neg} \circ d_1$ and $CPLZF^{\vee\neg}$ are extensionally equal. The crucial case is to show $\vdash_{ZF} | \equiv_{Elem\ bool \to Elem\ bool \to Elem\ bool} \lambda x,y.\sim(\sim x \& \sim y)$, where $|$, $\&$, and $\sim$ are appropriately defined operations on ZF-Booleans. This follows easily because $\equiv_{Elem\ bool}$ is defined as provable equality. Accordingly, we check that $CPLZF^{\vee\neg} \circ d_1$ and $CPLZF^{\wedge\neg}$ are extensionally equal.

Thus, the two logics are equivalent.

5 What is a logic combination?

[44] also develops a module system for MMT theories. The key idea is that the concepts regarding modularity can be captured by the abstract categorical properties of MMT. In particular, we have the following correspondence:

Modularity	MMT
Inheritance	Inclusion
Refinement	Morphism
Instantiation	Pushout

This induces a module system for all logical frameworks. We can apply this to define the abstract notion of combining logics and to build concrete combinations declaratively.

5.1 Combinations of syntax are colimits

We have defined syntax of a logic as a logical theory, and following the perspective of Section 3.3, we assume these theories include the inference systems as well. Theorem 3.38 shows that the logical theories and morphisms form a category so that we can use colimits as a natural and general way of combining them.

While logical frameworks do not necessarily admit all colimits, we can define several important constructions for an arbitrary logical framework.

DEFINITION 5.1 (Union)
Two MMT theories Syn and Syn' are **compatible** if whenever $c \in \mathrm{dom}(Syn) \cap \mathrm{dom}(Syn')$ then $Syn(c) = Syn'(c)$. In that case, the **union** $Syn \cup Syn'$ arises from Syn, Syn' by removing all Syn'-declarations that are also in Syn.

EXAMPLE 5.2 (Propositional Logic)
Let $Base$ be the fragment of PL from Example 3.3 that only declares o and thm. Accordingly, let $IPL^{\neg}$, …, $IPL^{\vee}$ be the fragments of IPL that extend $Base$ with the respective connective and its proof rules. Finally, let Classical be the theory that extends $Base$ with the classicality rule from Example 3.27.

Then we have the following diagram in which IPL and CPL arise as colimits formed by taking the respective unions:

$$\begin{tikzcd}
 & Base \arrow[dl, hook] \arrow[d, hook] \arrow[dr, hook] \arrow[drr, hook] & & \\
IPL^{\neg} \arrow[dr, hook] & \cdots \arrow[d, hook] & IPL^{\vee} \arrow[dl, hook] & \text{Classical} \arrow[ddl, hook] \\
 & IPL \arrow[dr, hook] & & \\
 & & CPL &
\end{tikzcd}$$

A comprehensive version can be found in [23].

DEFINITION 5.3 (Instantiation)
Consider an MMT theory inclusion $I \hookrightarrow Syn'$ and a theory morphism $i : I \to Syn$.

If $\mathrm{dom}(Syn')\setminus\mathrm{dom}(I)$ and $\mathrm{dom}(Syn)$ are disjoint, the **instantiation** of Syn' with i is the pushout $i(Syn')$.

$$\begin{tikzcd} I \arrow[r, hook] \arrow[d, "i"'] & Syn' \arrow[d, "i^{Syn'}"] \\ Syn \arrow[r, hook] & i(Syn') \end{tikzcd}$$

We can think of the theory I as the interface of Syn' and of the morphism i as providing values for the parameters declared in the interface. Thus, Syn' behaves like an SML-style functor to which we pass named arguments i.

Definition 5.4 (Renaming)
Consider a theory Syn and a list $\rho = c_1 \rightsquigarrow c'_1, \ldots, c_n \rightsquigarrow c'_n$ where the $c_i \in \mathrm{dom}(Syn)$ are pairwise distinct and the $c'_i \notin \mathrm{dom}(Syn)$ are pairwise distinct. Then $Syn[\rho]$ arises from Syn by replacing every identifier c_i with c'_i.

For an identifier s, we write $s.Syn$ for the theory $Syn[\ldots, c \rightsquigarrow s.c, \ldots]$ where c runs over all elements of $\mathrm{dom}(Syn)$.

Clearly, $Syn[\rho]$ and $s.Syn$ are isomorphic to Syn. But renaming can be important to avoid name clashes that make the pushout undefined. In particular, if no identifier in $\mathrm{dom}(Syn)$ starts with s, then Syn and $s.Syn'$ always have disjoint domains.

Remark 5.5 (Renaming versus Instantiation)
Definition 5.4 introduces the notation $c \rightsquigarrow c'$ for the situation where renaming creates a *new* theory in which c is replaced by a new identifier c'. Definition 5.3 includes a different form of renaming as a special case, namely if i consists of cases $c \mapsto c'$. Here c is renamed to an identifier c' in an *existing* theory.

The latter is more general and allows in particular non-injective renamings. But it is not enough by itself: we need both forms for our main combination operator in Definition 5.6.

We can now recover the structure declarations used in [44] as follows:

Definition 5.6 (Generative Pushout)
Let r be the isomorphism $Syn \to s.Syn$. Given $I \hookrightarrow Syn$ and $i: I \to \Sigma$, we abbreviate $i' = i \circ r|_I^{-1}$ and define

$$\Sigma, s : Syn\{i\} = i'(s.Syn)$$

In that case, we write s for the morphism $i'^{s.Syn} \circ r$.

$$\begin{tikzcd} s.I \arrow[r, hook] \arrow[d, "r|_I^{-1}"'] & s.Syn \\ I \arrow[r, hook] \arrow[d, "i"'] & Syn \arrow[u, "r"'] \arrow[d, "s"] \\ \Sigma \arrow[r, hook] & \Sigma, s : Syn\{i\} \end{tikzcd}$$

The intuition behind $s : Syn\{i\}$ is that of a generative functor application. s is the name of an import that extends Σ with modified copies of the declarations in Syn. Name clashes are avoided because all imported identifiers $c\in\mathrm{dom}(Syn)$ are renamed to $s.c$. Moreover, some identifiers $c\in\mathrm{dom}(Syn)$ are instantiated by mapping them to Σ-expressions $i(c)$.

If we think of $s:Syn\{i\}$ as a special declaration (as done in [44]), we can build generative pushouts declaratively.

Example 5.7 (Combining Intuitionistic and Classical Logic)
We can use the colimit of the following diagram to combine the syntaxes of intuitionistic and classical propositional logic from Example 3.3:

$$\begin{tikzcd} & o:\mathtt{type} \arrow[dl, hook] \arrow[dr, hook'] & \\ PL & & PL \end{tikzcd}$$

Note that this colimit is not the union $PL\cup PL=PL$. But we can construct a colimit using generative pushouts:

$$L\;=\;a:\mathtt{type},\, int:PL\{o\mapsto a\},\, cl:PL\{o\mapsto a\}$$

Here we import PL twice using different qualifiers int and cl. The morphisms $o\mapsto a$ have the effect that the two copies of PL share the type a of sentences. This yields the syntax of a logic with two sets of propositional connectives, e.g. we have sentences $(p\, int.\wedge q)\, cl.\Rightarrow r$.

If we construct the analogous pushout of IPL and CPL, we can also combine the inference systems of intuitionistic and classical logic from Example 3.27. However, note that there are multiple reasonable ways to combine these inference systems so that we cannot expect the logical framework to pick the right combination for us by itself. For example, the theory

$$ICPL=a:\mathtt{type},\, int:IPL\{o\mapsto a\},\, cl:CPL\{o\mapsto a\}$$

is not very interesting because the two truth judgements $int.thm$ and $cl.thm$ remain unrelated. A more interesting choice might be the logical theory

$$ICPL,\; intcl:\Pi F:a.\, int.thm\,F\rightarrow cl.thm\,F$$

which adds a rule that derives classical from intuitionistic truth.

5.2 Combinations of semantics are universal morphisms

We can build combinations of morphisms accordingly by constructing the universal morphism out of a colimit.

Definition 5.8 (Union)
Given compatible theories Syn and Syn', two morphisms $sem:Syn\rightarrow Sem$ and $sem':Syn'\rightarrow Sem$ are **compatible** if whenever $c\in\mathrm{dom}(Syn)\cap\mathrm{dom}(Syn')$ then $sem(c)=sem'(c)$. In that case, the **union** $sem\cup sem':Syn\cup Syn'\rightarrow Sem$ arises by mapping every identifier according to Syn or Syn'.

Example 5.9 (Propositional Logic)
We can continue Example 5.2 by constructing the semantics of the combined theory IPL. Let $CPLZF$, $CPLZF^{\neg}$, ..., $CPLZF^{\vee}$ be the restrictions of $FOLZF$ from Example 3.36 to CPL, $CPL^{\neg}$, ..., $CPL^{\vee}$.

Then $CPLZF|_{IPL}$ arises as the universal morphism u out of the colimit IPL.

$$\begin{tikzcd}
& Base \arrow[dl, hook] \arrow[d, hook] \arrow[dr, hook'] & \\
IPL^{\neg} \arrow[dr, hook] \arrow[ddr, "CPLZF^{\neg}"'] & \cdots \arrow[d, hook] & IPL^{\vee} \arrow[dl, hook'] \arrow[ddl, "CPLZF^{\vee}"] \\
& IPL \arrow[d, "u"] & \\
& ZF &
\end{tikzcd}$$

We can obtain *CPLZF* and *FOLZF* as combinations of little morphisms accordingly. The full example is developed in [23].

We can construct the universal morphisms out of instantiations and generative pushouts accordingly. However, universal morphisms can only combine compatible semantics. This often means that some ingenuity is required to combine very different semantics. This is not surprising: just like there is not always a canonical combination of the proof theories, we cannot expect a canonical combination of the model theories.

Example 5.10 (Combining Intuitionistic and Classical Logic)
We might try to continue Example 5.7 by combining the morphisms *CPLZF* and *IPLKrZF* from Example 4.10. However, the left diagram below does not commute so that we do not obtain a universal morphism out of *ICPL*. The problem is that $IPLKrZF(o) = Elem\ univ \to Elem\ bool$ but $CPLZF(o) = Elem\ bool$.

$$\begin{tikzcd}
& o : \mathtt{type} \arrow[dl, hook] \arrow[dr, hook'] & \\
IPL \arrow[dr, hook] \arrow[ddr, "IPLKrZF"'] & & CPL \arrow[dl, hook'] \arrow[ddl, "CPLZF"] \\
& ICPL & \\
& ZF, FOLZF(Preord) &
\end{tikzcd}
\qquad
\begin{tikzcd}
& o : \mathtt{type} \arrow[dl, hook] \arrow[dr, hook'] & \\
IPL \arrow[dr, hook] \arrow[ddr, "IPLKrZF"'] & & CPL \arrow[dl, hook'] \arrow[ddl, "CPLKrZF"] \\
& ICPL \arrow[d, "u"] & \\
& ZF, FOLZF(Preord),\ m : thm\,[\forall v.\exists w. v \leq w \wedge max(w)] &
\end{tikzcd}$$

For the sake of example, we give an interesting possibility for the semantics of the combined logic.

In the right diagram above, we define a new morphism *CPLKrZF* that interprets *CPL* in a Kripke model. Because we put $CPLKrZF(o) = Elem\ univ \to Elem\ bool$, the diagram commutes. The basic idea of *CPLKrZF* is to interpret the classical truth judgement as truth in all $\leq$-maximal worlds. Thus, e.g. the classical negation of F is interpreted as 1 at world v if F is interpreted as 0 at all maximal worlds w with $v \leq w$.

To formalize this, we abbreviate $max(w) = \neg\exists x. w \leq x \wedge \neg x \doteq w$ and add an axiom m to the codomain that makes sure there is a maximal world above every world. Then we put

$$CPLKrZF(thm) = \lambda f : Elem\ bool.\ thm[\forall w. max(w) \Rightarrow f\ w \doteq 1]$$

$$CPLKrZF(\neg) = \lambda f : Elem\ univ \to Elem\ bool. \lambda v : Elem\ univ.$$
$$if\ \big(\forall w. v \leq w \wedge max(w) \Rightarrow f\ w \doteq 0\big)\ 1\ else\ 0$$

where we assume we have already defined in *ZF* an if–then–else operator of type $o \to Elembool \to Elembool \to Elembool$. The cases for the other connectives are defined accordingly.

Now we obtain a universal morphism u out of *ICPL* that combines *IPLKrZF* and *CPLKrZF*. To complete the logic definition, we extend u with one case that maps the additional rule *intcl* from Example 5.7 to the easy *ZF*-proof that intuitionistic truth (interpreted as 1 in all worlds) implies classical truth (interpreted as 1 in all maximal worlds).

Note that our point in Example 5.10 was to exemplify how logics can be combined. We have not yet studied the usefulness of that particular combination of intuitionistic and classical logic. Indeed, it is an open problem how to design a good combination. An example is given in [30], which inspired our morphism *CPLKrZF*.

REMARK 5.11 (Future Work)
It is inherent in our approach that the combination of sound logics is sound again. But it is not clear to us at this point whether similar theorems can be obtained for completeness.

6 Related work

We divide related work into formalist and abstract approaches. Roughly speaking, the former explicitly represent the syntax and proof theory, whereas the latter assume abstract sets of sentences and consequence relations. For propositional logics, this difference is not essential because the syntax is so simple. However, for logics with binders, the approaches diverge substantially.

Formalist Approaches. There are relatively few formalist approaches along our lines. Research on logical frameworks has generally focused on representing *individual* logics and translations and to develop best practices for doing so. Logic representations are commonly done ad hoc but could be formulated systematically as logical theories.

Isabelle and LF are the most commonly used frameworks. Due to our general definition of logical framework, both can be used interchangeably to formulate our examples. The main strength of Isabelle [39] is the tool support for theorem proving within logical theories. The main strength of LF implementations like Twelf [41] is the tool support for proving meta-theorems about logical theories. Other type theories that are not primarily designed as logical frameworks are also used as such occasionally. Examples include encodings of modal logic in higher-order logic [4].

Similarly, translations of logics are usually represented individually. The deepest examples are a HOL-Nuprl translation [48] in Twelf and a HOL-ZF translation [28] in Isabelle. Both are formulated ad hoc but could be written as logical morphisms.

The LATIN project [7] systematically represented logic graphs in Twelf using logical theories and logical morphisms. It also used an MMT-style module system for Twelf [45] to combine logics and morphisms. The most comprehensive case studies can be found in [23, 25].

In [42], the present author gave a theoretical framework for LF-based logical theories and morphisms. A first approach towards generalizing this framework to arbitrary logical frameworks was presented in [8]. Both followed the logics-as-spans perspective described in Section 3.3. This made them much more complex than the present approach and, therefore, prevented establishing deeper theorems such as completeness criteria.

In parallel, the present author developed the MMT language [44]. MMT has no direct connection to logic and only provides a formalist framework for representing the syntax of declarative languages. A major contribution of the present work is to apply MMT to logical frameworks. This was crucial to permit the abstract definition of logical framework we give in Section 2, in particular the combination of declarative and categorical properties. This required several novel developments: Our declarative definitions of Section 2.2 are a complete reformulation of the relevant fragment of MMT, and our categorical definitions of Section 2.1 are a novel abstraction from MMT.

The idea of representing logics as LF theories and combining them via colimits was first presented in [24]. It uses a combination language with union, translation, and hiding based on ASL [50]. The MMT module system [44] we use here is equivalent to using union and translation [6].

Abstract Approaches. A widely used abstract approach works with pairs $(S, \vdash)$ where S is the set of sentences and $\vdash \subseteq \mathcal{P}(S) \times S$ is the consequence relation. $\vdash$ must satisfy axioms that induce a closure operator. Such pairs are used, e.g. as entailment systems in [14, 33], or logical systems in [5].

Given a logical framework with hypothetical reasoning, we can define one such pair for every non-logical theory Σ: S is the set of Σ-sentences and $\{F_1, \ldots, F_n\} \vdash F$ is given by our judgements $\vdash_\Sigma thm\, F_1 \Longrightarrow \ldots \Longrightarrow thm\, F_n \Longrightarrow thm\, F$.

[5] defines an equivalence relation between logical systems called equipollence. It can be seen as a special case of our notion of extensional isomorphism between logical theories. [32] defines (conservative) morphisms between entailment relations to compare the strength of logics. These correspond to our (proof-conservative) logical morphisms.

The consequence relation $\vdash$ is often defined in terms of an abstract model theory or (less frequently) proof theory. Most closely related to our definitions is the abstract model theory formulated by institutions [17]. Each of our logics $(\mathcal{L}, sem : Syn \to Sem, \Delta)$ gives rise naturally to an institution. The signatures are the $\mathcal{L}$-theories, and the signature morphisms are the $\mathcal{L}$-theory morphisms. The sentences are like ours, and sentence translation is homomorphic extension. The Σ-models are our Σ-models via *sem*, and model reduction along σ is given by precomposition with $sem(\sigma)$. Satisfaction is given by our truth in a model, and proving the satisfaction condition is straightforward. More generally, we can obtain a logic in the sense of [34] or [42] by adding our proofs.

Similarly, logic translations (T, t) can induce an institution comorphism [17], or more generally a logic comorphism in the sense of [42]. Sentences and proofs are translated by homomorphic extension along T and models by precomposition with t_Σ. However, our logic translations do not in general yield the satisfaction condition (SC). We moved the analogue of SC to a different definition because it is often hard to establish for translations. For example, SC holds if the diagram of Definition 4.1 commutes for all Σ, but this is often too restrictive in practice. More generally, SC follows from the existence of the two logical relations used in Theorem 4.7 and 4.8. The additional assumptions made in Theorem 4.8 have a similar role as the model expansion property of institution comorphisms (e.g. [9]). Thus, our soundness and completeness together correspond to SC and model expansion.

Logic multi-graphs can be studied abstractly as diagrams in the category of institutions and can be integrated into a single institution called the Grothendieck institution [13]. The Hets system [36] implements this construction, using a programming language to define the elements of the institution multi-graph.

Parchments [16] are similar to institutions but add an explicit representation of the syntax, which brings them closer to formalist approaches. [37] generalizes and applies parchments to combine logics using limits (corresponding to our colimits because their morphisms go the other way). While the definitions and notations are very different, the basic idea is very similar to using sorted first-order logic as a logical framework in our sense.

Fibring [49] corresponds to using first-order logic as a logical framework for combining propositional logics. Unconstrained and constrained fibring correspond to our unions and pushouts.

7 Conclusion

We presented a framework in which we answer fundamental questions about the nature of logics. These include how to define and identify, translate and compare, and combine and modularize logics.

A major achievement is that our framework spans the range from declarative logical frameworks based on type theory to abstract definitions based on sets and categories. Roughly speaking, the former are great for local methods such as an induction on the inference system of a logic, and the latter are great for global methods such as diagrams in a category of logical theories. The key insight to get both benefits while keeping the logical framework flexible was to use MMT as a universal representation layer.

Additionally, via MMT, our work is connected to a mature tool [43] for developing and working with concrete logical frameworks. This includes tool support for type-checking logic definitions, verifying logic translations and combining logics using a module system. Thus, we can treat logics as data, which can be machine-processed and distributed.

A second major achievement is to provide a theoretical foundation for formalist logic multi-graphs. Here each node defines the syntax and proof theory of a formal systems such as first-order logic, higher-order logic or axiomatic set theory. And each edge interprets one system in another, which subsumes both logic translations and model theoretical semantics.

Using the logical framework LF, the LATIN project [7] has already build such a logic graph, which includes dozens of reusable modules for individual language features of logics. The present work provides the theoretical framework for this approach and establishes general correctness criteria.

References

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

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

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

[4] C. Benzmüller and L. Paulson. Quantified multimodal logics in simple type theory. *Logica Universalis*, **7**, 7–20, 2013.

[5] C. Caleiro and R. Gonçalves. Equipollent logical systems. In J.-Y. Béziau, ed., *Logica Universalis*, pp. 99–112. Birkhäuser Verlag, 2005.

[6] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski and F. Rabe. A Proof Theoretic Interpretation of Model Theoretic Hiding. In T. Mossakowski and H. Kreowski, eds, *Recent Trends in Algebraic Development Techniques 2010*, pp. 118–138. Springer, 2012.

[7] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski and F. Rabe. Project abstract: Logic Atlas and Integrator (LATIN). In J. Davenport, W. Farmer, F. Rabe, and J. Urban, eds, *Intelligent Computer Mathematics*, pp. 289–291. Springer, 2011.

[8] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, F. Rabe and K. Sojakova. Towards logical frameworks in the heterogeneous tool set hets. In T. Mossakowski and H. Kreowski, eds, *Recent Trends in Algebraic Development Techniques 2010*, pp. 139–159. Springer, 2012.

[9] M. Cerioli and J. Meseguer. May I borrow your logic? (Transporting logical structures along maps). *Theoretical Computer Science*, **173**, 311–347, 1997.

[10] Coq Development Team. The Coq proof assistant: reference manual. *Technical report*, INRIA, 2014.

[11] V. Căzănescu and G. Roşu. Weak inclusion systems. *Mathematical Structures in Computer Science*, **7**, 195–206, 1997.

[12] R. Diaconescu, J. Goguen and P. Stefaneas. Logical support for modularization. In G. Huet and G. Plotkin, eds, *Logical Environments*, pp. 83–130. Cambridge Press, 1993.

[13] R. Diaconescu. Grothendieck institutions. *Applied Categorical Structures*, **10**, 383–402, 2002.

[14] J. Fiadeiro and A. Sernadas. Structuring theories on consequence. In D. Sannella and A. Tarlecki, eds, *Recent Trends in Data Type Specification*, vol. 332, pp. 44–72. Springer, 1987.

[15] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O'Connor, S. Ould Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi and L. Théry. A machine-checked proof of the odd order theorem. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, eds, *Interactive Theorem Proving*, pp. 163–179, 2013.

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

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

[18] M. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P. Subrahmanyam, eds, *VLSI Specification, Verification and Synthesis*, pp. 73–128. Kluwer-Academic Publishers, 1988.

[19] J. Harrison. HOL light: A tutorial introduction. In *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pp. 265–269. Springer, 1996.

[20] L. Henkin. The completeness of the first-order functional calculus. *Journal of Symbolic Logic*, **14**, 159–166, 1949.

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

[22] F. Horozal, M. Kohlhase and F. Rabe. Extending MKM formats at the statement level. In J. Campbell, J. Carette, G. Dos Reis, J. Jeuring, P. Sojka, V. Sorge and M. Wenzel, eds, *Intelligent Computer Mathematics*, pp. 64–79. Springer, 2012.

[23] F. Horozal and F. Rabe. Representing model theory in a type-theoretical logical framework. *Theoretical Computer Science*, **412**, 4919–4945, 2011.

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

[25] M. Iancu and F. Rabe. Formalizing foundations of mathematics. *Mathematical Structures in Computer Science*, **21**, 883–911, 2011.

[26] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch and S. Winwood. seL4: formal verification of an operating-system kernel. *Communications of the ACM*, **53**, 107–115, 2010.

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

[28] A. Krauss and A. Schropp. A mechanized translation from higher-order logic to set theory. In *Interactive Theorem Proving*, M. Kaufmann and L. Paulson, eds, pp. 323–338. Springer, 2010.

[29] F. Lawvere. *Functional Semantics of Algebraic Theories*. PhD thesis, Columbia University, New York City, United States, 1963.

[30] C. Liang and D. Miller. Kripke semantics and proof systems for combining intuitionistic logic and classical logic. *Annals of Pure and Applied Logic*, **164**, 86–111, 2013.

[31] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, Part I. *Communications of the ACM*, **3**, 184–195, 1960.

[32] T. Mossakowski, R. Diaconescu and A. Tarlecki. What is a logic translation? *Logica Universalis*, **3**, 95–124, 2009.

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

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

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

[36] T. Mossakowski, C. Maeder and K. Lüttich. The heterogeneous tool set. In *Tools and Algorithms for the Construction and Analysis of Systems 2007*, vol. 4424 of *Lecture Notes in Computer Science*, O. Grumberg and M. Huth, ed., pp. 519–522, 2007.

[37] T. Mossakowski, A. Tarlecki and W. Pawlowski. Combining and representing logical systems using model-theoretic parchments. In *Recent Trends in Algebraic Development Techniques*, F. Parisi-Presicce, ed., pp. 349–364. Springer, 1997.

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

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

[40] F. Pfenning. Logical frameworks. In *Handbook of automated reasoning*, J. Robinson and A. Voronkov, eds, pp. 1063–1147. Elsevier, 2001.

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

[42] F. Rabe. A Logical Framework Combining Model and proof theory. *Mathematical Structures in Computer Science*, **23**, 945–1001, 2013.

[43] F. Rabe. The MMT API: A generic MKM system. In *Intelligent Computer Mathematics*, J. Carette, D. Aspinall, C. Lange, P. Sojka and W. Windsteiger, eds, pp. 339–343. Springer, 2013.

[44] F. Rabe and M. Kohlhase. A scalable module system. *Information and Computation*, **230**, 1–54, 2013.

[45] F. Rabe and C. Schürmann. A practical module system for LF. In *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, J. Cheney and A. Felty, eds, pp. 40–48. ACM Press, 2009.

[46] F. Rabe and K. Sojakova. Logical relations for a logical framework. *ACM Transactions on Computational Logic*, **14**, 1–34, 2013.

[47] K. Sojakova. Mechanically verifying logic translations. Master's thesis, Jacobs University Bremen, Germany, 2010.

[48] C. Schürmann and M. Stehr. An executable formalization of the HOL/Nuprl connection in the metalogical framework Twelf. In *11th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, F. Baader and A. Voronkov, eds, Springer, 2004.

[49] A. Sernadas, C. Sernadas and C. Caleiro. Fibring of logics as a categorial construction. *Journal of Logic and Computation*, **9**, 149–179, 1999.

[50] D. Sannella and M. Wirsing. A kernel language for algebraic specification and implementation. In *Fundamentals of Computation Theory*, M. Karpinski, ed., pp. 413–427. Springer, 1983.

[51] A. Trybulec and H. Blair. Computer assisted reasoning with MIZAR. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, A. Joshi, ed., pp. 26–28, Morgan Kaufmann, 1985.

Received 1 October 2014