Extensional Signature Morphisms

Florian Rabe

Jacobs University, Bremen, Germany

Abstract

We introduce extensional signature morphisms, a generalization of the signature morphisms that are routinely used in declarative languages. Out of the two key properties of normal signature morphisms – homomorphic extensions and preservation of judgments – we waive the former. Thus, extensional morphisms become more difficult to represent in computer systems, but much more expressive, in particularly enjoying stronger closure properties.

Our work includes a generalization of logical relations to the setting of extensional morphisms and a notion of 2-cells that yields a 2-category of signatures and extensional morphisms. Moreover, it improves on the representation of denotational in type theoretical frameworks. In all cases, the theory proves "nice" indicating a sweet spot between expressivity and

Our treatment is generic in the sense that it applies to a wide variety of declarative languages. We exemplify our work using the language of dependent type theory. Moreover, we apply our theory by giving a concrete extensional morphisms that represents the erase-types-and-relativize-quantifiers translation from typed to untyped logic – a logic translation that previously could not be expressed and mechanically verified in a declarative logical framework.

1 Introduction and Related Work

In declarative languages, especially logics and type theories, we often encapsulate a group of related declarations into a **signature**. For example, in first-order logic the signatures are lists of function and predicate symbols declarations. For example, the signature Mon of monoids consists of a binary function symbol $comp$ for composition and a nullary function symbol e for the unit. In type theories based on the Curry-Howard correspondence [CF58, How80], it is possible to treat even theories as special cases of signatures because axioms are treated as normal declarations, and we will adopt this convention in this paper.

A signature morphism σ from a signature Σ to a signature Σ' then is a function that translates Σ-expressions (terms, types, formulas, proofs, etc.) to corresponding Σ'-expressions.

Even though the exact definition of **signature morphisms** depends on the declarative language, they can intuitively be characterized by two properties:

- Homomorphic extension: A signature morphism is determined by its action on the Σ-symbols only, which is then extended homomorphically to all Σ-expressions. For example, a morphism from Mon to Σ' is given by mapping $comp$ to a binary Σ'-function $comp'(x,y)$ and e to a Σ'-term e', and the homomorphic extension induces that $\sigma(comp(e,e)) = comp'(e',e')$.
- Judgment preservation: A signature morphism preserves all judgments about expressions, in particular typing and provability. For example, if p is a Σ-proof of F, then $\sigma(p)$ is a Σ'-proof of $\sigma(F)$.

The interplay of these two properties has made them a crucial tool for relating and structuring theories in mathematics and computer science. Firstly, the homomorphic property makes it easy to give signature morphisms: Instead of defining a function on all Σ-expressions, we only have to define it on symbols. This is particularly in computer systems because (in the typical case where signatures have finitely many symbols) it permits finitary representations that can be checked and applied automatically. Secondly, the preservation property makes signature morphisms useful for reusing knowledge: All theorems proved in Σ give rise to theorems in Σ' by translation along σ. Moreover, it permits stating every result (e.g., a definition, theorem, or algorithm) in the smallest possible theory Σ and then reusing it in many larger theories Σ'.

Historically, the systematic use of theory morphisms to build mathematical theories goes back to the works by Bourbaki [Bou64] although they did not make the method explicit. To the author's knowledge, the first formal definition of theory morphisms was given in [End72] (under the name interpretation). The little theories methodology [FGT92] first advocated the systematic use of theory morphisms to build mathematical theories.

Signature morphisms have proved especially powerful in computer-supported **module systems** because the key concepts of extension and refinement can be formalized as morphisms. Among programming languages, this includes the SML module system [MTHM97] (whose functors can be seen as signature morphisms). In algebraic specification, the framework of institutions [GB92] is based on a category of signatures and morphisms, and the ASL module system [SW83] provides a generic module system for any institution. In formal deduction, the IMPS proof system [FGT93] was built to exploit theory morphisms in a computer system. A survey can be found in [RK13].

Type theoretical **logical frameworks** like LF [HHP93] permit representing declarative languages as signatures of the framework and language translations as signature morphisms of the framework [HST94]. This permits relating and structuring whole logics and type theories in the same way as the signatures of a fixed logic/type theory. The author has applied this approach systematically using a module system for LF [RS09] and applied it to building the LATIN atlas [CHK+11].

However, the two properties of signature morphisms restrict the complexity of the translations that can be represented as framework-level signature morphisms. Therefore, practical implementations of logic translations usually use the freedom of a general purpose programming language both for systematic networks of translations as in the Hets system [MML07] and for ad-hoc translations as, e.g., in [OS06]. Meachnically verified logic translations have been limited to simple examples (e.g., propositional modal logic to propositional logic in [RS13]), a programming language on top of a logical framework (e.g., HOL to Nuprl in [SS04]), or explicit formalizations that are independent of the original logic translation (HOL to FOL in Isabelle [MP08] verified in [BP13]). Generally, we see a trade-off between declarative signature morphisms of limited expressivity that can be automatically verified and applied and flexible implementations that must be verified independently.

This paper **contributes** to the investigation of this trade-off by introducing a generalization of signature morphisms. Essentially, we waive the homomorphic extension property and insist only on the judgment preservation property. We call the resulting objects *extensional morphisms* (short: e-morphisms) because their representation relies on set theoretical functions rather than a finitary formal object.

In addition to permitting natural representations of more language translations, the theory of e-morphisms turns out to be very interesting itself. Naturally, some properties are lost compared to normal morphisms – in particular not all e-morphisms have effective representations in computer systems (although we can still represent many individual e-morphisms and constructions on them). But at the same time we gain new properties: e-morphisms exhibit nicer closure properties than ordinary morphisms (including products and sub-morphisms); and they admit a natural notion of 2-cells between them that extends signatures and morphisms into a 2-category.

This makes e-morphisms a good candidate for the representation of models in logical frameworks. In [Rab13, HR11], we used normal morphisms $\Sigma \to F$ to represent models or implementations of Σ, where F formalizes the background theory of mathematics or the respective programming language. This can be seen as a formalist variant of the functorial models [Law63] in categorical logic. But this approach did not carry over to an elegant representation of model morphisms. Neither did it permit a representation of Henkin-models of higher-order type theories (which functorial models achieve by waiving the semantic analogue of the homomorphic extension property). E-morphisms remedy both of these short-comings.

E-morphisms are reminiscent of and similar to lax logical relations [PPST00]. Logical relations have primarily been studied from a semantic perspective, including the use of logical relations on models going back to [Rey74]. *Lax* logical relations [PPST00] waive the homomorphic extension property. This generalization has very similar advantages and disadvantages as our generalization to e-morphisms have: They are more difficult to represent but enjoy stronger closure properties.

Recently *syntactic* logical relations have been studied, where the logical relation is formulated as a syntactic object in an appropriately expressive type theory [BJP12, RS13]. Syntactic logical relations behave very similarly to morphisms: They are also defined homomorphically (albeit in a different way) and also preserve judgments (in a corresponding different way).

In addition to e-morphisms, we also introduce e-relations. These can be seen as merging the ideas of lax and syntactic logical relations and generalizing them to e-morphisms.

Concretely, we develop three new concepts. In Sect. 4, we introduce *e-morphisms*. In Sect. 5, we generalize the concept of logical relations (both with and without the homomorphic extension property) to *e-relations* on e-morphisms. In Sect. 6, we define morphisms between e-morphisms, which serve as the 2-cells of a 2-category; we call these 2-cells *connections*. In all three cases, we give both an abstract categorical and a concrete type theoretical description and prove them to be equivalent. Moreover, we investigate their basic categorical and algebraic properties.

One results are stated as generically as possible in order to maximize reusability and to emphasize that the general ideas do not depend on the specific underlying language. Therefore, we begin by introducing a generic type theory in Sect. 2, which permits adding individual type theoretical features as modules. In Sect. 3, we give a few example modules, which we will as examples throughout: dependent products and function types. All our results can be generalized to many other type theoretical features.

As a running **example**, we represent the proof theoretical version of the erase-types-and-relativize-quantifiers translation from a typed first-order logic to the corresponding untyped one in a logical framework based on dependent type theory. This example is a novel contribution in itself: It is the first time that such a complex logic translation is verified in a declarative logical framework.

The example is quite substantial and split up into multiple interrelated examples. Therefore, we decided to give the whole example in a separate section (Sect. 7) instead of interspersing it into the text.

We conclude by foreshadowing 2 future lines of work that will apply e-morphisms. Sect. 9 sketches how e-morphisms can be implemented in computer systems by suggesting a concrete grammar that generates many important e-morphisms and equips them effective representations. And in Sect. 8, we describe how we expect to capture reflection as a special case of an e-morphism. It was in fact this application to reflection that originally motivated the present investigation

2 A Basic Type Theory

2.1 Syntax

	Grammar	Typing judgment	Equality judgment
Signatures	$\Sigma ::= \cdot \mid \Sigma, c : E$	$\vdash \Sigma\ \mathtt{Sig}$	
Morphisms	$\sigma ::= \cdot \mid \sigma, c \mapsto E$	$\vdash \sigma : \Sigma \to \Sigma'$	
Contexts	$\Gamma ::= \cdot \mid \Gamma, x : E$	$\vdash_\Sigma \Gamma\ \mathtt{Ctx}$	$\vdash_\Sigma \Gamma = \Gamma'$
Substitutions	$\gamma ::= \cdot \mid \gamma, E$	$\vdash_\Sigma \gamma : \Gamma_1 \to \Gamma_2$	$\vdash_\Sigma \gamma = \gamma' : \Gamma_1 \to \Gamma_2$
Expressions	$E ::= c \mid x \mid \mathtt{type} \mid \mathtt{kind} \mid C(E^*;\Gamma;E^*)$	$\Gamma \vdash_\Sigma E : E'$ $\Gamma \vdash_\Sigma E\ \mathtt{Univ}$	$\Gamma \vdash_\Sigma E = E'$

Figure 1: Basic Syntax and Judgments

As the starting point of our investigation, we use a basic type theory whose syntax and judgments are given in Fig. 1. Our type theory is unusual in that it formalizes the structural levels that handle declarations but is generic at the expression level.

At the structural level, the grammar defines **signatures** Σ for globally declaring **constants** at any universe and – relative to such a signature – **contexts** Γ for locally declaring typed **variables** (which are available for variable binding). Signatures and contexts are ordered, and once declared, a constant or variable may occur in later declarations.

Both signatures and contexts come with homomorphic mappings: **morphisms** map between signatures and **substitutions** between contexts. A morphism $\vdash \sigma : \Sigma \to \Sigma'$ assigns a Σ-expression of the appropriate type to each Σ'-constant. Similarly, a substitution $\Gamma \vdash_\Sigma \gamma : \Gamma_1 \to \Gamma_2$ assigns a Γ_2-expression of the appropriate type to each Γ_1-variable.

At the expression level, genericity means that we do not fix the non-terminals or productions for complex expressions. **Expressions** are used to generically represent terms, types, kinds, universes etc. They are subject to typing and equality, and some expressions are universes. The only expressions over a signature Σ in a context Γ that we fix are the constants c and the variables x declared in Σ and Γ as well as the **universes** of types and kinds. We will usually use the meta-variables t (intuitively: terms) and A (intuitively: types) for the expressions on the left and right hand side of the typing judgment.

All other expressions must be provided by adding expression constructors C. We use a very general form of **complex expressions**: The expression $C(\vec{E};\Gamma;\vec{E'})$ takes a list of arguments $\vec{E}$, then a context of bound variables Γ, and then a list of scopes $\vec{E'}$, i.e., a list of arguments that may refer to the bound variables. In practice, we usually encounter

- first-order constructors where Γ and $\vec{E'}$ are empty,
  for example, application can be written as $@(f,x;\cdot;\cdot)$ (here $\vec{E}$ has length 2), or

- binders where $\vec{E}$ is empty and $\vec{E}'$ has length 1,
  for example, abstraction can be written as $\lambda(\cdot; x : A; t)$ (here Γ has length 1).
- universes which are usually nullary or take just a single argument (a natural number).

Notation 1 (Antecedents). As usual, we will omit the antecedent of a judgment if it is the empty context.

Moreover, it is often convenient to use a fixed context as the antecedent in the judgments for contexts and substitutions. This permits talking about context and substitution fragments that are well-formed relative to a fixed initial fragment. Therefore, we define some abbreviations:

$$\Gamma \vdash_\Sigma \Gamma' \ \texttt{Ctx} \qquad \text{for} \qquad \cdot \vdash_\Sigma \Gamma, \Gamma' \ \texttt{Ctx}$$

$$\Gamma \vdash_\Sigma \gamma : \Gamma_1 \to \Gamma_2 \qquad \text{for} \qquad \cdot \vdash_\Sigma id_\Gamma, \gamma : \Gamma, \Gamma_1 \to \Gamma, \Gamma_2$$

$$\Gamma \vdash_\Sigma \gamma = \gamma' : \Gamma_1 \to \Gamma_2 \qquad \text{for} \qquad \cdot \vdash_\Sigma id_\Gamma, \gamma = id_\Gamma, \gamma' : \Gamma, \Gamma_1 \to \Gamma, \Gamma_2$$

where the identity substitution id_Γ is defined in Def. 2.

We make some basic definitions for substitutions:

Definition 2 (Substitutions)**.** Given $\Gamma = x_1 : A_1, \dots, x_n : A_n$, we define

$$id_\Gamma := x_1, \dots, x_n$$

Given $\vdash_\Sigma \gamma : \Gamma \to \Delta$, we define the application $-[\gamma]$ to well-typed expressions, contexts, and substitutions by

- for an expression in context Γ:

$$\begin{array}{lll} x_i[\gamma] & := t_i & \text{if } \gamma = t_1, \dots, t_n \text{ and } \Gamma = x_1 : A_1, \dots, x_n : A_n \\ c[\gamma] & := c & \\ \texttt{type}[\gamma] & := \texttt{type} & \\ C(\dots, s_i, \dots; \Delta; \dots, t_i, \dots)[\gamma] & := C(\dots, s_i[\gamma], \dots; \Delta[\gamma]; \dots, t_i[\gamma, id_\Delta], \dots) & \end{array}$$

- for a context Δ such that $\Gamma \vdash_\Sigma \Delta \ \texttt{Ctx}$:

$$\begin{array}{ll} \cdot[\gamma] & := \cdot \\ (\Delta, x : A)[\gamma] & := \Delta[\gamma], x : A[\gamma, id_\Delta] \end{array}$$

- for a substitution δ:

$$\begin{array}{ll} \cdot[\gamma] & := \cdot \\ (\delta, t)[\gamma] & := \delta[\gamma], t[\gamma] \end{array}$$

Given $\Gamma \vdash_\Sigma \gamma : \Gamma' \to \Delta$, we define the application $-[\gamma]$ as an abbreviation of $-[id_\Gamma, \gamma]$ (i.e., left-most entries in a substitution can be omitted if they are identity maps).

We have the analogue to Def. 2 for morphisms:

Definition 3 (Morphisms)**.** Given $\vdash \sigma : \Sigma \to \Sigma'$, we define the application $\sigma(-)$ to well-typed expressions, contexts, and substitutions over Σ in the same way as for substitutions with the exception of the following two cases:

$$\begin{array}{ll} \sigma(x) & := x \\ \sigma(c) & := E \quad \text{if } c \mapsto E \text{ in } \sigma \end{array}$$

2.2 Inference Rules

The rules of our type theory are given in Fig. 2 and 3.

Fig. 2 gives the typing rules. Signatures and morphisms as well as contexts and substitutions are typed component-wise in the usual way. The only subtlety is that while signatures may declare any $c : A$ such that A is typed by any universe, contexts may only declare typed variables.

$$\frac{}{\vdash \cdot \ \texttt{Sig}} \qquad \frac{\vdash \Sigma \ \texttt{Sig} \quad c \text{ not in } \Sigma \quad \cdot \vdash_\Sigma A : U \quad \cdot \vdash_\Sigma U \ \texttt{Univ}}{\vdash \Sigma, c : A \ \texttt{Sig}}$$

$$\frac{\vdash \Sigma' \ \texttt{Sig}}{\vdash \cdot : \cdot \to \Sigma'} \qquad \frac{\vdash \sigma : \Sigma \to \Sigma' \quad \vdash_\Sigma t : \sigma(A)}{\vdash \sigma, c \mapsto t \ : \ \Sigma, c : A \ \to \ \Sigma'}$$

$$\frac{\vdash \Sigma \ \texttt{Sig}}{\vdash_\Sigma \cdot \ \texttt{Ctx}} \qquad \frac{\vdash_\Sigma \Gamma \ \texttt{Ctx} \quad \Gamma \vdash_\Sigma A : \texttt{type}}{\vdash_\Sigma \Gamma, \ x : A \ \texttt{Ctx}}$$

$$\frac{\vdash_\Sigma \Gamma \ \texttt{Ctx}}{\vdash_\Sigma \cdot : \cdot \to \Gamma} \qquad \frac{\vdash_\Sigma \gamma : \Gamma' \to \Gamma \quad \Gamma' \vdash_\Sigma A : \texttt{type} \quad \Gamma \vdash_\Sigma t : \gamma(A)}{\vdash_\Sigma \gamma, t \ : \ \Gamma', x : A \to \Gamma}$$

$$\frac{\vdash_\Sigma \Gamma \ \texttt{Ctx} \quad c : E \text{ in } \Sigma}{\Gamma \vdash_\Sigma c : E} \qquad \frac{\vdash_\Sigma \Gamma \ \texttt{Ctx} \quad x : E \text{ in } \Gamma}{\Gamma \vdash_\Sigma x : E}$$

$$\frac{\vdash_\Sigma \Gamma \ \texttt{Ctx}}{\Gamma \vdash_\Sigma \texttt{type Univ}} \qquad \frac{\vdash_\Sigma \Gamma \ \texttt{Ctx}}{\Gamma \vdash_\Sigma \texttt{kind Univ}} \qquad \frac{\vdash_\Sigma \Gamma \ \texttt{Ctx}}{\Gamma \vdash_\Sigma \texttt{type} : \texttt{kind}}$$

Figure 2: Basic Typing Rules

$$\frac{\vdash_\Sigma \Gamma \ \texttt{Ctx}}{\vdash_\Sigma \cdot = \cdot} \qquad \frac{\vdash_\Sigma \Gamma = \Gamma' \quad \Gamma \vdash_\Sigma A = A'}{\vdash_\Sigma \Gamma, x : A = \Gamma, x : A'}$$

$$\frac{\vdash_\Sigma \Gamma \ \texttt{Ctx}}{\vdash_\Sigma \cdot = \cdot : \cdot \to \Gamma} \qquad \frac{\vdash_\Sigma \gamma_1 = \gamma_2 : \Gamma' \to \Gamma \quad \Gamma \vdash_\Sigma t_1 = t_2}{\vdash_\Sigma \gamma_1, t_1 = \gamma_2, t_2 : \Gamma', x : A \to \Gamma}$$

$$\frac{\Gamma \vdash_\Sigma E : E'}{\Gamma \vdash_\Sigma E = E} \qquad \frac{\Gamma \vdash_\Sigma E = E'}{\Gamma \vdash_\Sigma E' = E} \qquad \frac{\Gamma \vdash_\Sigma E = E' \quad \Gamma \vdash_\Sigma E' = E''}{\Gamma \vdash_\Sigma E = E''}$$

$$\frac{\Gamma \vdash_\Sigma s_i = s_i' \quad \Gamma \vdash_\Sigma \Delta = \Delta' \quad \Gamma, \Delta \vdash_\Sigma t_i = t_i'}{\Gamma \vdash_\Sigma C(s_1, \ldots, s_m; \Delta; t_1, \ldots, t_n) = C(s_1', \ldots, s_n'; \Delta'; t_1', \ldots, t_n')} * \qquad \frac{\Gamma \vdash_\Sigma t : A \quad \Gamma \vdash_\Sigma t = t' \quad \Gamma \vdash_\Sigma A = A'}{\Gamma \vdash_\Sigma t' : A'}$$

Figure 3: Basic Equality Rules

Fig. 3 gives the equality rules. Equality of contexts and substitutions is defined component-wise. Equality of expressions is a congruence relation, defined by reflexivity, symmetry, transitivity, and the two congruence rules. Only the rule $*$ is non-standard – it is a generic congruence rule for any expression constructor C. For example, if C is λ, then $*$ subsumes the ξ rule for conversion under a λ.

Our rules only provide minimal information about typing and equality of expressions. Additional rules to govern the typing, equality, and universe-hoods of complex expressions must be added in conjunction with the respective expression constructor C.

2.3 Categories

The structural properties of our generic type theory are well-known, and we will state the ones that we will use frequently later on:

Theorem 4. *The following rules are admissible*

$$\frac{\vdash_\Sigma \Gamma\ \mathtt{Ctx}}{\vdash_\Sigma id_\Gamma : \Gamma \to \Gamma} \qquad \frac{\vdash_\Sigma \gamma : \Gamma \to \Delta \quad \vdash_\Sigma \delta : \Delta \to \Theta}{\vdash_\Sigma \gamma[\delta] : \Gamma \to \Theta}$$

$$\frac{\vdash_\Sigma \gamma : \Gamma \to \Gamma' \quad \Gamma \vdash_\Sigma \Delta\ \mathtt{Ctx}}{\Gamma' \vdash_\Sigma \Delta[\gamma]\ \mathtt{Ctx}} \qquad \frac{\vdash_\Sigma \gamma : \Gamma \to \Gamma' \quad \Gamma \vdash_\Sigma \delta : \Delta \to \Delta'}{\Gamma' \vdash_\Sigma \delta[\gamma] : \Delta[\gamma] \to \Delta'[\gamma]}$$

$$\frac{\vdash_\Sigma \gamma : \Gamma \to \Gamma' \quad \Gamma \vdash_\Sigma t : A}{\Gamma' \vdash_\Sigma t[\gamma] : A[\gamma]}$$

and the contexts and substitutions (up to the equality judgment) form a category with composition $\delta \circ \gamma$ given by substitution application $\gamma[\delta]$.

The analogue to Lem. 2 for signatures and morphisms also holds. We omit the details, which will be special cases of later theorems.

Remark 5. Technically, these results depend on which type theoretical features are added to the type theory because these add inference rules, and the admissibility proofs must be carried out by induction on derivations. However, it is possible to establish these results generically in the presence of arbitrary rules of a certain shape. All rules we will consider here are instances of this generic rule.

3 Individual Type Theoretical Features

We can now extend our basic type theory with various specific type features. Each features usually consists of a few constructors C together with notations and inference rules for them.

We will look at 2 such features in particular, namely dependent product and function types. We will use slightly generalized definitions where both are n-ary in the sense that they bind a whole context. Given $\Gamma = x_1 : A_1, \dots, x_n : A_n$, we will use $\langle\Gamma\rangle$ instead of $\Sigma_{x_1:A_1} \dots \Sigma_{x_{n-1}:A_{n-1}} A_n$ and $\Pi_\Gamma A$ instead of $\Pi_{x_1:A_1} \dots \Pi_{x_n:A_n} A$.

3.1 Dependent Product Types

The feature of dependent product types consists of the following constructors, notations, and rules.

Constructor	Arguments	Notation
formation	$(\cdot;\Gamma;\cdot)$	$\langle\Gamma\rangle$
introduction	$(\gamma;\cdot;\cdot)$	$\langle\gamma\rangle$
i-th elimination for $i \in \mathbb{N}$	$(t;\cdot;\cdot)$	$t.i$

$$\frac{\Gamma \vdash_\Sigma \Gamma'\ \mathtt{Ctx}}{\Gamma \vdash_\Sigma \langle\Gamma'\rangle : \mathtt{type}}$$

$$\frac{\Gamma \vdash_\Sigma \gamma : \Gamma' \to \cdot}{\Gamma \vdash_\Sigma \langle\gamma\rangle : \langle\Gamma'\rangle} \qquad \frac{\Gamma \vdash_\Sigma t : \langle x_1 : A_1, \dots, x_n : A_n\rangle}{\Gamma \vdash_\Sigma t.i : A_i[t.1, \dots, t.(i-1)]}$$

$$\frac{\Gamma \vdash_\Sigma t_1, \dots, t_n : \Gamma' \to \cdot}{\Gamma \vdash_\Sigma \langle t_1, \dots, t_n\rangle.i = t_i} \qquad \frac{\Gamma \vdash_\Sigma t : \langle\Gamma'\rangle}{\Gamma \vdash_\Sigma \langle t.1, \dots, t.n\rangle = t}$$

As the special case $n = 0$, i.e., Γ is empty, we obtain the unit type.

3.2 Dependent Function Types

The feature of dependent function types consists of the following constructors, notations, and rules.

Constructor	Arguments	Notation
formation	$(\cdot; \Gamma; A)$	$\Pi_{\Gamma'} A$
introduction	$(\cdot; \Gamma; t)$	$\lambda_{\Gamma'} t$
elimination	$(t, \gamma; \cdot; \cdot)$	$t\,\gamma$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma A : \mathtt{type}}{\Gamma \vdash_\Sigma \Pi_{\Gamma'} A : \mathtt{type}}$$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma t : A}{\Gamma \vdash_\Sigma \lambda_{\Gamma'} t : \Pi_{\Gamma'} A} \qquad \frac{\Gamma \vdash_\Sigma t : \Pi_{\Gamma'} A \quad \Gamma \vdash_\Sigma \gamma : \Gamma' \to \cdot}{\Gamma \vdash_\Sigma t\,\gamma : A[\gamma]}$$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma t : A \quad \Gamma \vdash_\Sigma \gamma : \Gamma' \to \cdot}{\Gamma \vdash_\Sigma (\lambda_{\Gamma'} t)\,\gamma = t[\gamma]} \qquad \frac{\Gamma \vdash_\Sigma t : \Pi_{\Gamma'} A}{\Gamma \vdash_\Sigma \lambda_{\Gamma'} (t\, id_{\Gamma'}) = t}$$

Example signatures in the presence of dependent function types are given by Ex. 61 and 62. An example morphism between them is given in Ex. 63.

3.3 Higher Dependent Functions

If we replace $\mathtt{type}$ with U for some $\Gamma \vdash_\Sigma U\ \mathtt{Univ}$, we obtain λ-abstraction over typed variables in expressions at the universe U.

We are not particularly interested here in this as a feature of our type theory. This is because the constructions we will consider in the sequel will focus on operations on contexts. And these higher abstractions essentially do not affect the contexts. (If they do occur in contexts, they can be removed by β-reduction.)

Nonetheless, it is important to mention this feature because at least the special case $U = \mathtt{kind}$ is needed to actually declare any dependent types in signatures. Otherwise, the features of dependent products and functions would degenerate to simple products and functions.

4 From Morphisms to E-Morphisms

4.1 Formal Definition

E-Morphisms as Abstract Syntax Transformations Morphisms are syntax transformations that are defined inductively and enjoy various useful properties, including the preservation typing and commuting with substitution. We will now abstract from these properties and call any transformation that satisfies these properties an extensional morphism:

Definition 6. For a signature Σ, we write $\mathtt{Cons}(\Sigma)$ for the category of contexts and substitutions over Σ.

Definition 7. A *extensional morphism*, short *e-morphism* from a signature Σ to a signature Σ' is a functor $\Phi : \mathtt{Cons}(\Sigma) \to \mathtt{Cons}(\Sigma')$, denoted $-^\Phi$, that satisfies

$$\begin{aligned} .^\Phi &= . \\ (\Gamma,\ x : A)^\Phi &= \Gamma^\Phi,\ x : A^{\Phi_\Gamma} \end{aligned}$$

$$\begin{aligned} .^\Phi &= . \\ (\gamma,\ t)^\Phi &= \gamma^\Phi,\ t^{\Phi_\Gamma} \qquad \text{if } \vdash_\Sigma \gamma : \Gamma' \to \Gamma \end{aligned}$$

for arbitrary types A^{Φ_Γ} and terms t^{Φ_Γ}.

Note that the types/terms A^{Φ_Γ} and t^{Φ_Γ} are uniquely determined by the functor Φ. Conversely, the functor Φ can be defined by giving the expressions A^{Φ_Γ} and t^{Φ_Γ}.

To avoid confusion, we stop using the word "morphism" without qualification:

Definition 8. An *intensional morphism*, short *i-morphism* is a morphism in the sense of Sect. 2.

Def. 7 focuses on the abstract categorical flavor. It is mathematically elegant but occasionally impractical to work with. Therefore, we immediately complement it with an equivalent definition that focuses on the concrete syntax-transformation flavor:

Theorem 9 (Characterization of E-Morphisms)**.** *Assume a family of maps* Φ_Γ *that map* Σ*-expressions to* Σ'*-expressions. Then the following are equivalent:*

1. Φ *is an e-morphism* $\Sigma \to \Sigma'$*.*
2. Φ *preserves variables, typing, and substitution, i.e., the following rules are admissible:*

$$\frac{\vdash_\Sigma \Gamma, x : A\ \texttt{Ctx}}{(\Gamma, x : A)^\Phi \vdash_{\Sigma'} x^{\Phi_{\Gamma,x:A}} = x}$$

$$\frac{\Gamma \vdash_\Sigma t : A}{\Gamma^\Phi \vdash_{\Sigma'} t^{\Phi_\Gamma} : A^{\Phi_\Gamma}}$$

$$\frac{\vdash_\Sigma \gamma : \Gamma \to \Gamma' \qquad \Gamma \vdash_\Sigma t : A}{\Gamma'^\Phi \vdash_{\Sigma'} t^{\Phi_\Gamma}[\gamma^\Phi] = t[\gamma]^{\Phi_{\Gamma'}}}$$

Proof. Assume Φ to be an e-morphism. Firstly, the preservation of variables follows from

$$id_{\Gamma^\Phi}, x = id_{\Gamma, x:A^\Phi} = id_{\Gamma,x:A}{}^\Phi = id_\Gamma, x^\Phi = id_{\Gamma^\Phi}, x^{\Phi_{\Gamma,x:A}}$$

Secondly, if $\vdash_\Sigma \gamma : \Gamma \to \Gamma'$ and $\Gamma \vdash_\Sigma t : A$, we know that the left diagram below commutes, and then by the properties of Φ also the right one:

$$\begin{array}{ccc} \Gamma', x : A[\gamma] & \xrightarrow{id_{\Gamma'}, t[\gamma]} & \Gamma' \\ {\scriptstyle \gamma, x}\uparrow & & \uparrow{\scriptstyle \gamma} \\ \Gamma, x : A & \xrightarrow[id_\Gamma, t]{} & \Gamma \end{array} \qquad \begin{array}{ccc} \Gamma'^\Phi, x : A[\gamma]^{\Phi_{\Gamma'}} & \xrightarrow{id_{\Gamma'^\Phi}, t[\gamma]^{\Phi_{\Gamma'}}} & \Gamma'^\Phi \\ {\scriptstyle \gamma^\Phi, x}\uparrow & & \uparrow{\scriptstyle \gamma^\Phi} \\ \Gamma^\Phi, x : A^{\Phi_\Gamma} & \xrightarrow[id_{\Gamma^\Phi}, t^{\Phi_\Gamma}]{} & \Gamma^\Phi \end{array}$$

Applying the two substitutions in the right diagram to x yields $t^{\Phi_\Gamma}[\gamma^\Phi] = t[\gamma]^{\Phi_{\Gamma'}}$, which yields the preservation of substitution and typing.

Conversely, assume the preservation properties. Φ already determines the e-morphism uniquely, and it is well-defined due to the preservation of typing. So we only have to proof the functoriality. Firstly, $\vdash_{\Sigma'} id_\Gamma{}^\Phi = id_{\Gamma^\Phi} : \Gamma^\Phi \to \Gamma^\Phi$ follows from the preservation of variables.

Secondly, consider $\vdash_\Sigma \gamma' : \Gamma'' \to \Gamma'$ and $\vdash_\Sigma \gamma : \Gamma' \to \Gamma$ with $\gamma' = t_1, \ldots, t_n$. Then,

$$\vdash_{\Sigma'} (\gamma \circ \gamma')^\Phi = t_1[\gamma]^{\Phi_\Gamma}, \ldots, t_n[\gamma]^{\Phi_\Gamma} : \Gamma''^\Phi \to \Gamma^\Phi$$

and

$$\vdash_{\Sigma'} \gamma^\Phi \circ \gamma'^\Phi = t_1^{\Phi_{\Gamma'}}[\gamma^\Phi], \ldots, t_n^{\Phi_{\Gamma'}}[\gamma^\Phi] : \Gamma''^\Phi \to \Gamma^\Phi$$

and the two right hand sides are equal due to the preservation of substitution. □

Thus, e-morphisms are exactly the transformations that retain some crucial properties of i-morphisms. While, i-morphisms are defined inductively in such a way that they satisfy these properties by construction, e-morphisms are black boxes and can be defined in any way. From a practical perspective, the main property that we lose when going from i-morphisms to e-morphism is that i-morphisms are determined by a finite set of expressions and thus can be expressed as formal objects over a finite grammar.

Naturality of E-Morphisms An e-morphism Φ maps both contexts and expressions in context. This has the effect that we can think of Φ both as a functor and as a natural transformation "on itself". In particular, the preservation of substitution corresponds to the naturality of the maps $-^{\Phi_\Gamma}$. The following makes this intuition precise:

Definition 10. For a signature Σ, we write $\mathtt{Terms}_\Sigma$ for the functor $\mathtt{Cons}(\Sigma) \to \mathcal{SET}$, which maps Γ to the set of pairs (t, A) satisfying $\Gamma \vdash_\Sigma t : A$ and which maps substitutions γ to the substitution application function $-[\gamma]$.

Theorem 11. *Given an e-morphism $\Phi : \Sigma \to \Sigma'$, the family of maps $\Phi_\Gamma : (t, A) \mapsto (t^{\Phi_\Gamma}, A^{\Phi_\Gamma})$ is a natural transformation $\mathtt{Terms}_\Sigma \Rightarrow \mathtt{Terms}_{\Sigma'} \circ \Phi$.*

$$\begin{array}{ccc} \mathcal{SET} & \xrightarrow{\Phi} & \mathcal{SET} \\ {\scriptstyle \mathtt{Terms}_\Sigma}\uparrow & & \uparrow{\scriptstyle \mathtt{Terms}_{\Sigma'}} \\ \mathtt{Cons}(\Sigma) & \xrightarrow[\Phi]{} & \mathtt{Cons}(\Sigma') \end{array}$$

Proof. The naturality follows immediately from the preservation of substitution. □

In particular, the following diagram commutes for every substitution $\vdash_\Sigma \gamma : \Gamma \to \Gamma'$

$$\begin{array}{ccc} \mathtt{Terms}_\Sigma(\Gamma') & \xrightarrow{-^{\Phi_{\Gamma'}}} & \mathtt{Terms}_{\Sigma'}(\Phi^{\Gamma'}) \\ {\scriptstyle \gamma[-]}\uparrow & & \uparrow{\scriptstyle \gamma^\Phi[-]} \\ \mathtt{Terms}_\Sigma(\Gamma) & \xrightarrow{-^{\Phi_\Gamma}} & \mathtt{Terms}_{\Sigma'}(\Phi^\Gamma) \end{array}$$

Extending E-Morphisms to all Expressions So far, we have defined e-morphism as maps of contexts, substitutions, types, and terms. But we can also extend them in two ways:

- to context and substitution fragments introduced in Not. 1,
- to the kinds and type families which arise when using the feature of higher dependent functions from Sect. 3.3.

Definition 12. Consider an e-morphism $\Phi : \Sigma \to \Sigma'$.

- For $\Gamma \vdash_\Sigma \Gamma'\ \mathtt{Ctx}$, we have $(\Gamma, \Gamma')^\Phi = \Gamma^\Phi, \Delta$, and we define ${\Gamma'}^{\Phi_\Gamma} = \Delta$.
- For $\Gamma \vdash_\Sigma \gamma : \Gamma_1 \to \Gamma_2$, we have $(id_\Gamma, \gamma)^\Phi = id_{\Gamma^\Phi}, \delta$, and we define $\gamma^{\Phi_\Gamma} = \delta$.
- For higher dependent functions at universe $\mathtt{kind}$, we define

$$\begin{array}{lll} \mathtt{type}^{\Phi_\Gamma} & = \mathtt{type} & \\ (\Pi_{\Gamma'} K)^{\Phi_\Gamma} & = \Pi_{{\Gamma'}^{\Phi_\Gamma}} K^{\Phi_{\Gamma,\Gamma'}} & \\ (\lambda_{\Gamma'} A)^{\Phi_\Gamma} & = \lambda_{{\Gamma'}^{\Phi_\Gamma}} A^{\Phi_{\Gamma,\Gamma'}} & \\ (A\,\gamma)^{\Phi_\Gamma} & = \lambda_{\Gamma'}(A\,(\gamma, id_{\Gamma'}))^{\Phi_\Gamma} & \text{if } \Gamma \vdash_\Sigma A\,\gamma : \Pi_{\Gamma'} K \end{array}$$

Note that we perform η-expansion in the case for $A\,\gamma$ so that the definition eventually recurses into an atomic type.

In summary, we can say that e-morphisms are homomorphic on contexts, substitutions, kinds, and (essentially) type families. But they need not be homomorphic on types and terms.

With these extensions, we obtain:

Theorem 13. *An e-morphism $\Sigma \to \Sigma'$ preserves typing and substitution for all Σ-expressions.*

Proof. The proof is straightforward. □

4.2 Concrete E-Morphisms

We will now construct some e-morphisms.

Theorem 14. *Every i-morphism $\Sigma \to \Sigma'$ is an e-morphism.*

Proof. This follows immediately from the definition of the homomorphic extension of i-morphisms. □

It is not the case that all e-morphism are i-morphisms, as we will see from the following example:

Definition 15. Assume a type theory with dependent product types.[1]

Consider a list of e-morphisms $\Phi^i : \Sigma \to \Sigma'$ for $i = 1, \dots, n$. By mutual induction, we define an e-morphism $\Phi : \Sigma \to \Sigma'$ and substitutions $\varphi^i_\Gamma : \Gamma^{\Phi^i} \to \Gamma^\Phi$:

$$A^{\Phi_\Gamma} = \langle \dots, x_i : A^{\Phi^i_\Gamma}[\varphi^i_\Gamma], \dots \rangle$$
$$t^{\Phi_\Gamma} = \langle \dots, t^{\Phi^i_\Gamma}[\varphi^i_\Gamma], \dots \rangle$$
$$\varphi^i_\cdot = \cdot \qquad \text{and} \qquad \varphi^i_{\Gamma,x:A} = \varphi^i_\Gamma, x.i.$$

We denote this e-morphism by $\Phi^1 \times \dots \times \Phi^n$.

This definition is a lot simpler than its formal statement makes it appear. Intuitively, Φ maps every type A to the n-ary product of the A^{Φ^i} and every term t to the n-tuple consisting of the t^{Φ^i}. But if t or A contain free variables, we have to be careful as Γ^Φ and Γ^{Φ^i} are not the same context: The former contains variables of product type. Therefore, the substitutions φ^i_Γ map every variable x declared in Γ^{Φ^i} to the corresponding projection $x.i$, which is valid over Γ^Φ. With this intuition, it is easy to prove that Φ is well-defined:

Theorem 16. *In the situation of Def. 15, Φ is an e-morphism.*

Proof. The preservation of typing is obvious and the preservation of substitution straightforward. The preservation of variables follows after observing that

$$x^{\Phi_\Gamma} = \langle \dots, x^{\Phi^i_\Gamma}[\varphi^i_\Gamma], \dots \rangle$$

and

$$\Gamma^\Phi \vdash_{\Sigma'} x^{\Phi^i_\Gamma}[\varphi^i_\Gamma] = x.i$$

from the conversion rule for products. □

Note that Def. 15 includes the case $n = 0$ if we assume a type theory with an empty product as the unit type. This yields a unit e-morphism that maps every type to the unit type $\langle\cdot\rangle$ and every term to the unit term $\langle\cdot\rangle$.

Thus, e-morphisms are closed under finite products. More generally, we expect the same for any limit in the category $\texttt{Cons}(\Sigma')$ that the syntax can express. We do not have exponentials of e-morphisms, but we do have "constant powers":

Definition 17. Assume a type theory with dependent function types.[2] Consider an e-morphism $\Phi : \Sigma \to \Sigma'$ and a context $\vdash_{\Sigma'} U$ `Ctx`. By mutual induction, we define a functor $\Phi^U : \texttt{Cons}(\Sigma) \to \texttt{Cons}(\Sigma')$ and substitutions $\varphi_\Gamma : \Gamma^\Phi \to \Phi^U(\Gamma), U$:

$$A^{\Phi^U_\Gamma} = \Pi_U A^{\Phi_\Gamma}[\varphi_\Gamma]$$
$$t^{\Phi^U_\Gamma} = \lambda_U t^{\Phi_\Gamma}[\varphi_\Gamma]$$
$$\varphi_\cdot = \cdot \qquad \text{and} \qquad \varphi_{\Gamma,x:A} = \varphi_\Gamma, x\ id_U$$

Theorem 18. *In the situation of Def. 17, Φ^U is an e-morphism.*

Proof. The preservation of typing is obvious and the preservation of substitution straightforward. The preservation of variables follows after observing that

$$x^{\Phi^U_\Gamma} = \lambda_U x^{\Phi_\Gamma}[\varphi_\Gamma]$$

and

$$\Gamma^\Phi \vdash_{\Sigma'} x^{\Phi^U_\Gamma}[\varphi_\Gamma] = x\ id_U$$

from the conversion rule for functions. □

4.3 Interaction Between Type Constructors and E-Morphisms

By definition, i-morphisms have the crucial property that they are compositional, i.e., they commute with all type- and term-forming operations, in particular the ones from Sect. 3. Since this commutativity condition is waived for e-morphisms, it is interesting to ask what relationship there is (if any) between e-morphism and additional operators.

Intuitively, we expect a relatively strong condition for dependent products because e-morphisms preserve context formation, and dependent product types are essentially contexts. We also expect some relation for dependent functions and application are essentially just terms in context and substitution, and e-morphisms preserve substitution. These expectations are indeed correct as we will see now.

Product Types E-morphisms commute with dependent products up to a canonical isomorphism, i.e., $\langle\Gamma\rangle^\Phi$ and $\langle\Gamma^\Phi\rangle$ are isomorphic types. The following makes this precise:

Definition 19. In any category, we say that A and B are *canonically isomorphic* and write $A \overset{!}{\cong} B$ iff there is a canonical choice for an isomorphism between A and B.

Theorem 20. *For every e-morphism $\Phi : \Sigma \to \Sigma'$ and every $\Gamma \vdash_\Sigma \Gamma'$ `Ctx`*

$$\langle\Gamma'\rangle^{\Phi_\Gamma} \overset{!}{\cong} \langle\Gamma'^{\Phi_\Gamma}\rangle$$

in the category of types in context Γ^Φ.

Proof. We know that the $\Gamma' \overset{!}{\cong} x : \langle\Gamma'\rangle$ in the category of Σ-contexts that extend Γ. The canonical isomorphism is the substitution that maps each x_i in Γ' to $x.i$. Similarly, we have $\Gamma'^{\Phi_\Gamma} \overset{!}{\cong} x : \langle\Gamma'^{\Phi_\Gamma}\rangle$.

Since e-morphisms are functors, they preserve isomorphisms so that $\Gamma'^{\Phi_\Gamma} \overset{!}{\cong} (x : \langle\Gamma'\rangle)^{\Phi_\Gamma}$. Then the canonical isomorphism is obtained by composition. □

Function Types Contrary to product types, dependent function types do not commute with e-morphisms, i.e., the types $(\Pi_\Gamma A)^\Phi$ and $\Pi_{\Gamma^\Phi} A^{\Phi_\Gamma}$ are not in general isomorphic. However, the type $(\Pi_\Gamma A)^\Phi$ is canonically embedded into the type $\Pi_{\Gamma^\Phi} A^{\Phi_\Gamma}$. The following makes this precise:

Theorem 21. *For every e-morphism $\Phi : \Sigma \to \Sigma'$ and every $\Gamma, \Gamma' \vdash_\Sigma A : \mathtt{type}$, there is a canonical term*

$$\Gamma^\Phi,\ x : (\Pi_{\Gamma'} A)^{\Phi_\Gamma} \vdash_{\Sigma'} x^* : \Pi_{\Gamma'^{\Phi_\Gamma}} A^{\Phi_{\Gamma,\Gamma'}}$$

Proof. We apply Φ to the judgment

$$\Gamma,\ x : \Pi_{\Gamma'} A,\ \Gamma' \vdash_\Sigma x\ id_{\Gamma'} : A$$

This yields

$$\Gamma^\Phi,\ x : (\Pi_{\Gamma'} A)^{\Phi_\Gamma},\ \Gamma'^{\Phi_\Gamma} \vdash_\Sigma (x\ id_{\Gamma'})^{\Phi_{\Gamma, x:(\Pi_{\Gamma'} A)^{\Phi_\Gamma}, \Gamma'}} : A^{\Phi_{\Gamma,\Gamma'}}$$

Here, we have cleaned up by removing irrelevant variable declarations from the Δ in X^{Φ_Δ}. This is possible because Φ preserves substitutions, in particular the inclusion substitutions that add variables to Δ that do not occur in X.

Now x^* is obtained by λ-abstraction, i.e., we obtain x^* as

$$\lambda_{\Gamma' \Phi_\Gamma} (x\ id_{\Gamma'})^{\Phi_{\Gamma, x:(\Pi_{\Gamma'} A)^{\Phi_\Gamma}, \Gamma'}}$$

□

Notation 22. In the situation of Thm. 21, given any $\Gamma \vdash_{\Sigma'} f : (\Pi_{\Gamma'} A)^{\Phi_\Gamma}$, we write f^* for the expression $x^*[id_{\Gamma^\Phi}, f]$.

Technically, f^* depends on the context and type of f, which we omit in the notation. Moreover, f^* depends on Φ: If that is not clear from the context, we write $f^{*\Phi}$.

Remark 23. There is a deep intuition behind the construction of f^*. Given a (for simplicity: closed) Σ-function f of type $\lambda_{x:A} B$, we do not know the type of f^Φ – Φ could map Σ-functions to any Σ'-type.

But Φ is required to map all terms in context, and we know by η-expansion that every f is essentially a term in context. Therefore, we can "trick" Φ to reveal something about f^Φ after all: We first apply f to an arbitrary argument, then apply Φ, and then λ-abstract over the arbitrary argument. This yields f^* as $\lambda_{x:A^\Phi} (f\, x)^{\Phi_{x:A^\Phi}}$.

Note that this trick works precisely because Φ preserves variables, types, and substitutions.

The construction of f^* is enough to obtain a canonical embedding. But it is only useful if there is a meaningful relation between f^* and f^Φ. This relation indeed exists: Via the canonical embedding, e-morphisms commute with function application:

Theorem 24. *Whenever* $\Gamma \vdash_\Sigma f\, \gamma : A$, *then*

$$\Gamma^\Phi \vdash_{\Sigma'} (f\, \gamma)^{\Phi_\Gamma} = f^{\Phi_\Gamma *}\, \gamma^{\Phi_\Gamma}$$

Proof. This follows from $f\, \gamma = (f\ id_{\Gamma'})[\gamma]$ by applying Φ and commuting Φ with the substitutions γ and $id_{\Gamma^\Phi}, f^{\Phi_\Gamma}$. □

Thus, even though e-morphisms are seemingly allowed to map product and function types in any way they want, the preservation properties restrict this freedom sufficiently to recover much of the inductive definition we are used to from i-morphisms: Products just need some rearranging via an isomorphism, and when applying functions we have to insert the * operator.

Example 25. For the e-morphisms from Sect. 4.2, given an appropriate $f : (\Pi_{\Gamma'} A)^{\Phi_\Gamma}$ where $x_1, \ldots, x_n$ are the variables in Γ', the function f^* looks as follows.

- If Φ is an i-morphism, then $f^* = f$.
- If $\Phi = \Phi^1 \times \ldots \times \Phi^m$, then
$$f^{*\Phi} = \lambda_{\Gamma' \Phi_\Gamma} \langle \ldots, f.i^{*\Phi_i}\, (x_1.i, \ldots, x_n.i), \ldots \rangle$$
In particular, in the special case where $\Gamma = \cdot$, $\Gamma' = x : A$, and $m = 2$, this yields
$$f^{*\Phi} = \lambda_{x:A^\Phi} \langle f.1^*\, x.1,\ f.2^*\, x.2 \rangle$$
- If Φ is a power of the form Ψ^U, then
$$f^{*\Phi} = \lambda_{\Gamma' \Phi_\Gamma} \lambda_U \big((f\ id_U)^{*\Psi}\, (x_1\ id_U), \ldots, (x_n\ id_U) \big)$$
In particular, in the special case where $\Gamma = \cdot$, $\Gamma' = x : A$, and $U = p : P$, this yields
$$f^{*\Phi} = \lambda_{x:A^\Phi} \lambda_{p:P} \big((f\, p)^*\, (x\, p) \big)$$

4.4 Combining E-Morphisms

An important property of i-morphisms is that they can be combined modularly. Intuitively, if $\sigma_i : \Sigma_i \to \Sigma$ are i-morphisms that agree on the overlap between Σ_1 and Σ_2 (if any), then $\sigma_1 \cup \sigma_2 : \Sigma_1 \cup \Sigma_2 \to \Sigma$ is an i-morphism. Here $\cup$ is defined as concatenation and removal of overlap.

This property is exploited in module systems like in ASL [SW83], SML [MTHM97] (whose structures correspond to our i-morphisms in this case), or MMT [RK13], which build large signatures out of small reusable modules. By combining i-morphisms, this modularity can be extended to building morphisms out these large signatures accordingly.

It is in the nature of e-morphisms that similar combination results will be more difficult to obtain. E-morphisms Φ_i out of Σ_i can be defined using arbitrary inductions on the Σ_i expressions. Thus, it is not obvious how to define an e-morphism $\Phi : \Sigma_1 \cup \Sigma_2 \to \Sigma$ on expressions that use constants from both Σ_1 and Σ_2.

So far, we can only give one positive result, which however has already proved valuable in practice:

Definition 26 (I-Extension of an E-Morphism)**.** Consider an e-morphism $\Phi : \Sigma \to \Sigma'$ and an extension Σ, Γ of Σ such that Γ declares only typed constants (i.e., Γ is essentially a context). Moreover, consider an i-morphism $\vdash id_{\Sigma'}, \gamma : \Sigma', \Gamma^\Phi \to \Sigma'$.

Then we define an e-morphism $\Phi, \gamma : \Sigma, \Gamma \to \Sigma'$ by

$$\Delta^{\Phi,\gamma} = \Delta^{\Phi_\Gamma}[\gamma] \qquad \text{for } \vdash_{\Sigma,\Gamma} \Delta\ \texttt{Ctx}$$

$$\delta^{\Phi,\gamma} = \delta^{\Phi_\Gamma}[\gamma] \qquad \text{for } \vdash_{\Sigma,\Gamma} \delta : \Delta \to \Delta'$$

Theorem 27. *In the situation of Def. 26, Φ, γ is an e-morphism.*

Proof. It is enough to show the definition is well-typed. Then the functoriality follows immediately because we are only composing two functors.

Because Γ can be regarded as a Σ-context, we obtain

- Γ^Φ can be regarded as a Σ'-context
- γ can be regarded as a substitution and then $\vdash_{\Sigma'} \gamma : \Gamma^\Phi \to \cdot$
- $\vdash_{\Sigma,\Gamma} \Delta\ \texttt{Ctx}$ is equivalent to $\Gamma \vdash_\Sigma \Delta\ \texttt{Ctx}$
- $\vdash_{\Sigma,\Gamma} \delta : \Delta \to \Delta'$ is equivalent to $\Gamma \vdash_\Sigma \delta : \Delta \to \Delta'$

For contexts Δ, we obtain first $\Gamma^\Phi \vdash_{\Sigma'} \Delta^{\Phi_\Gamma}\ \texttt{Ctx}$ and then $\vdash_{\Sigma'} \Delta^{\Phi_\Gamma}[\gamma]\ \texttt{Ctx}$ as needed. Accordingly, for substitutions δ, we obtain first $\Gamma^\Phi \vdash_{\Sigma'} \delta^{\Phi_\Gamma} : \Delta^{\Phi_\Gamma} \to \Delta'^{\Phi_\Gamma}$ and then $\vdash_{\Sigma'} \delta^{\Phi_\Gamma}[\gamma] : \Delta^{\Phi_\Gamma}[\gamma] \to \Delta'^{\Phi_\Gamma}[\gamma]$ as needed. □

Thm. 27 has the effect that we can at least combine a single e-morphisms with i-morphisms if the domain of the e-morphism covers all type declarations. This is a relatively common situation when dependent type theory is used as a logical framework like in LF [HHP93] and was one of our initial motivations to consider e-morphisms at all. We get back to this in Ex. 68.

5 Logical Relations on E-Morphisms

In [RS13], we introduced the notion of a logical relation on an i-morphism. We can now generalize this definition to e-morphisms.

5.1 Formal Definition

Definition 28. Given an e-morphism $\Phi : \Sigma \to \Sigma'$. A logical relation ρ on Φ is a functor $\Phi : \texttt{Cons}(\Sigma) \to \texttt{Cons}(\Sigma')$, denoted $-^\rho$, that satisfies

$$\begin{aligned} \cdot^\rho &= \cdot \\ (\Gamma,\ x : A)^\rho &= \Gamma^\rho,\ x : A^{\Phi_\Gamma},\ x^! : A^{\rho_\Gamma}[x] \end{aligned}$$

$$\begin{aligned} \cdot^\rho &= \cdot \\ (\gamma,\ t)^\rho &= \gamma^\rho,\ t^{\Phi_\Gamma},\ t^{\rho_\Gamma} \qquad \text{if } \vdash_\Sigma \gamma : \Gamma' \to \Gamma \end{aligned}$$

for arbitrary types A^{ρ_Γ} and terms t^{ρ_Γ}.

Notation 29. We assume that for every variable x, we can always find a fresh variable name $x^!$.

Remark 30. A^{ρ_Γ} is a type in a context whose last variable has type A^{Φ_Γ}. We will never give this variable a name. Instead, we will always use that type under a substitution $A^{\rho_\Gamma}[t]$ that substitutes a term for that variable.

Alternatively, we could introduce type-level λ-abstraction and consider A^{ρ_Γ} as a type family of kind $\Pi x : A^{\Phi_\Gamma}.\mathtt{type}$ (in which case the variable name would be transparent due to α-renaming).

In the same way as for e-morphisms, we complement this abstract definition of logical relations with a more concrete one. In particular, Def. 28 and Thm. 31 correspond to Def. 7 and Thm. 9.

Theorem 31 (Characterization of Logical Relations)**.** *Assume an e-morphisms $\Phi : \Sigma \to \Sigma'$ and a family of maps ρ_Γ that map Σ-expressions to Σ'-expressions. Then the following are equivalent:*

1. *ρ is a logical relation on Φ.*
2. *ρ preserves variables, typing, and substitution in the sense that the following rules are admissible:*

$$\frac{\vdash_\Sigma \Gamma, x : A\ \mathtt{Ctx}}{(\Gamma, x : A)^\rho \vdash_{\Sigma'} x^{\rho_{\Gamma,x:A}} = x^!}$$

$$\frac{\Gamma \vdash_\Sigma t : A}{\Gamma^\rho \vdash_{\Sigma'} t^{\rho_\Gamma} : A^{\rho_\Gamma}[t^{\Phi_\Gamma}]} \qquad \frac{\Gamma \vdash_\Sigma A : \mathtt{type}}{\Gamma^\rho, x : A^{\Phi_\Gamma} \vdash_{\Sigma'} A^{\rho_\Gamma}[x] : \mathtt{type}}$$

$$\frac{\vdash_\Sigma \gamma : \Gamma \to \Gamma' \qquad \Gamma \vdash_\Sigma t : A}{\Gamma'^\rho \vdash_{\Sigma'} t^{\rho_\Gamma}[\gamma^\rho] = t[\gamma]^{\rho_{\Gamma'}}} \qquad \frac{\vdash_\Sigma \gamma : \Gamma \to \Gamma' \qquad \Gamma \vdash_\Sigma A : \mathtt{type}}{\Gamma'^\rho, x : A^{\Phi_\Gamma}[\gamma^\rho] \vdash_{\Sigma'} A^{\rho_\Gamma}[\gamma^\rho, x] = A[\gamma]^{\rho_{\Gamma'}}[x]}$$

Proof. These conditions are essentially the same as in Thm. 9. We only spelt out the cases for types explicitly to increase clarity.

The proof proceeds in the same way as for Thm. 9. In particular, the well-definedness, the identity law, and the composition law of the functor are equivalent to the preservation of typing, variables, and substitution, respectively. □

Remark 32. The definition given here is that of a unary logical relation. More generally, our definition extends to n-ary logical relations (between n e-morphisms). However, because e-morphisms are closed under products in the presence of product types, we can restrict attention to the unary case without loss of generality: n-ary logical relations can be recovered as unary ones on the product of n e-morphisms.

5.2 Extensional and Intensional Relations

Just like e- and i-morphisms, we obtain a distinction between e- and i-relations. E-relations are the logical relations as defined in Def. 28. I-relations are the special case where an e-relation is determined inductively by its action on the constants.

Like for i-morphisms, we define i-relations generically for expressions of our basic type theory: If ρ is given as a list $\rho = \ldots, c \mapsto E, \ldots$ of assignments to the constants, then

$$\begin{aligned} x^{\rho_\Gamma} &:= x^! \\ c^{\rho_\Gamma} &:= E \quad \text{if } c \mapsto E \text{ in } \rho \end{aligned}$$

However, contrary to i-morphisms, there is no generic scheme that provides the cases of the inductive definition generically for arbitrary type theoretical features. Instead, we have to provide these cases separately for each type theoretical feature. Correspondingly, the inductive proof that i-relations are indeed e-relations cannot be carried out generically. Instead, the necessary cases have to supplied separately for every type theoretical feature.

We only have a general intuition how to construct these cases: For every type formation operator T, the i-relation at types A formed from T should state that an A-term satisfies the relation if all its elimination forms do.

The following examples apply this intuition to our example features of product and function types. The resulting definitions are well-known from the inductive definition of i-relations on i-morphisms. Here our results generalize them to i-relations on arbitrary e-morphisms.

I-Relations at Product Types The general intuition from above leads to the definition that a tuple satisfies an i-relation iff all its projections do. Concretely, we obtain the following cases for the inductive definition:

Definition 33. In a type theory with dependent product types, the cases for i-relations are

$$\langle \Delta_n \rangle^{\rho_\Gamma}[x] = \langle E_n \rangle$$

$$\langle \ldots, t_i, \ldots \rangle^{\rho_\Gamma} = \langle \ldots, t_i^{\rho_\Gamma}, \ldots \rangle$$

$$(t.i)^{\rho_\Gamma} = t^{\rho_\Gamma}.i$$

where we abbreviate

$$\begin{aligned} \Delta_i &= \ldots, x_i : A_i \\ E_i &= \ldots, x_i^! : A_i^{\rho_{\Gamma,\Delta_{i-1}}}[\delta_{i-1}, (x.i)^{\Phi_{\Gamma,x:\langle\Delta_n\rangle}}] \\ \delta_i &= \ldots, (x.i)^{\Phi_{\Gamma,x:\langle\Delta_n\rangle}}, x_i^! \end{aligned}$$

satisfying

$$\Gamma^\rho \vdash_{\Sigma'} \delta_i : \Delta_i^{\rho_\Gamma} \to x : \langle \Delta_n \rangle^{\Phi_\Gamma}, E_{i-1}$$

for $i = 1, \ldots, n$.

These definitions are more straightforward than they look. For terms, the definition is compositional. For types, we understand it best by specializing to the case where Γ is empty and where the product is simply-typed. Then

$$x : A^\Phi \vdash_{\Sigma'} A^\rho[x] = \ldots \times A_i^\rho[(x.i)^{\Phi_{x:A}}] \times \ldots \qquad \text{where} \qquad A = \ldots \times A_i \times \ldots$$

The rest of the definition is just bureaucracy to keep track of variable contexts. If we specialize further to the case where Φ is an i-morphism, we obtain

$$x : A^\Phi \vdash_{\Sigma'} A^\rho[x] = \ldots \times A_i^\rho[x.i] \times \ldots \qquad \text{where} \qquad A = \ldots \times A_i \times \ldots$$

This is the familiar form of a logical relation at product types: A tuple x is in the relation at $\ldots \times A_i \times \ldots$ if each component $x.i$ is in the relation at A_i.

Theorem 34. *I-relations formed as in Def. 33 are e-relations.*

Proof. We use Thm. 31.

The preservation of variables holds generically.

The preservation of typing and substitution are shown by induction on the syntax. The cases for substitution follow from the induction hypothesis by inspecting the cases in Def. 33. The cases for typing follow by inferring the types of the expressions on the right-hand sides of Def. 33 and comparing them to the types expected for an e-relation. The latter comparison is tedious but straightforward; it already uses the preservation of substitution. □

I-Relations at Function Types The general intuition from above leads to the definition that a function satisfies the relation if all its applications do, i.e., if it maps satisfying arguments to satisfying results. Concretely, we obtain the following cases for the inductive definition:

Definition 35. In a type theory with dependent function types, the cases for i-relations are

$$(\Pi_\Delta A)^{\rho_\Gamma}[x] = \Pi_{\Delta^{\rho_\Gamma}} A^{\rho_{\Gamma,\Delta}}[(x\ id_\Delta)^{\Phi_{\Gamma,x:\Pi_\Delta A,\Delta}}]$$

$$(\lambda_\Delta t)^{\rho_\Gamma} = \lambda_{\Delta^{\rho_\Gamma}} t^{\rho_{\Gamma,\Delta}}$$

$$(t\,\delta)^{\rho_\Gamma} = t^{\rho_\Gamma}\,\delta^{\rho_\Gamma}$$

Def. 35 is structurally very similar to Def. 33. The cases for terms are compositional, and the case for types is almost compositional. We understand it best by specializing to the case where Γ is empty. Then

$$x : (\Pi_\Delta A)^\Phi \vdash_{\Sigma'} (\Pi_\Delta A)^\rho[x] = \Pi_{\Delta^\rho} A^{\rho_\Delta}[(x\ id_\Delta)^{\Phi_{x:\Pi_\Delta A,\Delta}}]$$

If we further specialize to the case where Φ is an i-morphisms, we obtain

$$x : (\Pi_\Delta A)^\Phi \vdash_{\Sigma'} (\Pi_\Delta A)^\rho[x] = \Pi_{\Delta^\rho} A^{\rho_\Delta}[x\ id_\Delta]$$

If we specialize even further to the unary case where $\Delta = y : B$, we obtain

$$x : \Pi_{y:B^\Phi} A^{\Phi_{y:B}} \vdash_{\Sigma'} (\Pi_{y:B} A)^\rho[x] = \Pi_{y:B^\Phi,\, y^!:B^\rho} A^{\rho_{y:B}}[x\ y]$$

This is the familiar definition of logical relations at function types: A function x is in the relation at $\Pi_{y:B}A$ if it maps arguments y that are in the relation at B (assumed by $y^!$) to results $x\,y$ that are in the relation at A.

Remark 36. Note the similarity between the occurrences of $(x.i)^{\Phi_{\Gamma,x:\langle\Delta_n\rangle}}$ and $(x\ id_\Delta)^{\Phi_{\Gamma,x:\Pi_\Delta A,\Delta}}$ – in both cases, Φ is applies to the elimination form of the variable x, for which the relation is stated.

This hints at the possibility of a generic definition for arbitrary type theoretical features, of which Def. 33 and 35 would be special cases. However, we have so far not succeeded at finding one.

Theorem 37. *In a type theory with dependent function types, i-relations formed as in Def. 35 are e-relations.*

Proof. The proof proceeds in the same way as for Thm. 34. □

An example of an i-relation in the presence of function types is provided by Ex. 64.

Combining Features We can strengthen the results on feature-wise i-relations. If type theories are formed by combining features modularly, then the cases from Def. 33 and 35 as well as Thm. 34 and 37 combine to an inductive definition and proof, respectively.

More generally, we could now develop an abstract notion of type theoretical feature that permits defining new features and providing exactly the needed cases.

5.3 Concrete Logical Relations

Logical relations have been studied extensively employing various formal definitions. Roughly, there are two major distinctions between the typical definitions:

- syntactic relations, where A^Φ is a type over another signature and A^ρ is a predicate on it,
- semantic relations, where A^Φ is a universe in a model and A^ρ is a (unary) relation on it.

as well as between

- strict relations, which correspond to our i-relations and,
- lax relations [PPST00], which correspond to our e-relations.

Moreover, we can further distinguish based on the type theoretical features that are present.

I-Relations As shown above, i-relations are e-relations. The analogue to the preservation property for semantic relations is often referred to as the Basic Lemma.

[RS13] studies n-ary syntactic logical relations on i-morphisms $\Phi_1, \dots, \Phi_n$ in the presence of dependent function types. This work extends to semantic relations by using a codomain representing the foundation of mathematics such as set theory. It includes an overview of related work and a number of concrete examples.

We can recover these relations as i-relations on the e-morphism $\Phi_1 \times \ldots \times \Phi_n$. Note that [RS13] works with i-morphisms only, which are not closed under products. Therefore, it had to consider the n-ary case explicitly.

Morphisms as Relations Every e-morphism induces a trivial e-relation as follows:

Theorem 38. *Given an e-morphism Φ, we obtain a e-relation $\tilde{\Phi}$ on Φ by putting*

$$A^{\tilde{\Phi}_\Gamma}[x] = A^{\Phi_\Gamma}$$

$$t^{\tilde{\Phi}_\Gamma} = t^{\Phi_\Gamma}$$

Proof. The properties of $\tilde{\Phi}$ follow immediately from the ones of Φ. □

In these e-relations, the predicate $A^{\tilde{\Phi}_\Gamma}[x]$ never depends on x. By using the injection $\Phi \mapsto \tilde{\Phi}$, we can occasionally derive results about e-morphisms as special cases of results about e-relations.

Products In the presence of product types, we obtain the product of e-relations in essentially the same was as the product of e-morphisms:

Definition 39. Assume a type theory with product types. Consider a list of e-relations ρ^i on $\Phi : \Sigma \to \Sigma'$ for $i = 1, \dots, n$. By mutual induction, we define an e-relation ρ on Φ and substitutions $\varphi^i_\Gamma : \Gamma^{\rho^i} \to \Gamma^\rho$:

$$A^{\rho_\Gamma}[x] = \dots \times A^{\rho^i_\Gamma}[\varphi^i_\Gamma, x] \times \dots$$
$$t^{\rho_\Gamma} = \langle \dots, t^{\rho^i_\Gamma}[\varphi^i_\Gamma], \dots \rangle$$
$$\varphi^i_\cdot = \cdot \quad \text{and} \quad \varphi^i_{\Gamma,x:A} = \varphi^i_\Gamma, x, x^*.i.$$

We denote this e-relation by $\rho^1 \wedge \dots \wedge \rho^n$.

If we think of e-relations ρ on Φ as judgments on Φ, then $\rho^1 \wedge \rho^2$ is the conjunction (intersection) of the judgments ρ_1 and ρ_2: It holds if each ρ^i holds. As the special case $n = 0$, this yields a total e-relation $\top_\Phi$, which maps every type to the unit type, i.e., the relation that holds everywhere.

Just like in Def. 15, the substitutions φ^i are just bookkeeping. If Γ is empty, we obtain the very simple

$$A^\rho[x] = \dots \times A^{\rho^i}[x] \times \dots$$

We can also define a product, where every factor is an e-relation on a different e-morphism:

Definition 40. Assume a type theory with product types. Consider a list of e-relations ρ^i on $\Phi^i : \Sigma \to \Sigma'$ for $i = 1, \dots, n$. By mutual induction, we define an e-relation ρ on $\Phi^1 \times \dots \times \Phi^n$ and substitutions $\varphi^i_\Gamma : \Gamma^{\rho^i} \to \Gamma^\rho$:

$$A^{\rho_\Gamma}[x] = \dots \times A^{\rho^i_\Gamma}[\varphi^i_\Gamma, x.i] \times \dots$$
$$t^{\rho_\Gamma} = \langle \dots, t^{\rho^i_\Gamma}[\varphi^i_\Gamma], \dots \rangle$$
$$\varphi^i_\cdot = \cdot \quad \text{and} \quad \varphi^i_{\Gamma,x:A} = \varphi^i_\Gamma, x.i, x^*.i.$$

We denote this e-relation by $\rho^1 \times \dots \times \rho^n$.

This amounts to taking a product in the category category of functors $\mathbf{Con}(\Sigma) \to \mathbf{Con}(\Sigma')$. The special case $n = 0$, yields the e-relation $\top_\top$, which is a terminal element. If we think of e-relation ρ on Φ as elements of Φ, then $\rho_1 \times \rho_2$ on $\Phi_1 \times \Phi_2$ corresponds to cartesian product.

Theorem 41. *In the situation of Def. 39 and Def. 40, the defined objects are indeed e-morphisms.*
Moreover, we have the isomorphism

$$(\rho_1 \times \rho_2) \wedge (\rho'_1 \times \rho'_2) \cong (\rho_1 \wedge \rho'_1) \times (\rho_2 \wedge \rho'_2).$$

Proof. The proofs are straightforward. □

Binary Logical Relations We expect the constructions given for lax logical relations in [PPST00] to carry over to our setting. These include a number of closure properties for e-relations.

The most important one is the closure of binary e-relations (i.e., e-relations on e-morphisms $\Phi \times \Phi'$) under composition. If ρ_i is an e-relation on $\Phi_i \times \Phi_{i+1}$ for $i = 1, 2$, then $\rho_2 \circ \rho_1$ is an e-relation on $\Phi_1 \times \Phi_3$ defined by

$$A^{\rho_2 \circ \rho_{1\Gamma}}[\langle x, z \rangle] = \langle y : A^{\Phi_{2\Gamma}}, \_ : A^{\rho_{1\Gamma}}[\langle x, y \rangle] \times A^{\rho_{2\Gamma}}[\langle y, z \rangle] \rangle$$
$$t^{\rho_2 \circ \rho_{1\Gamma}} = \langle t^{\Phi_{2\Gamma}}, \langle t^{\rho_{1\Gamma}}, t^{\rho_{2\Gamma}} \rangle \rangle$$

5.4 Logical Relations as Sub-E-Morphisms

A logical relation ρ on Φ can be seen as a dependent product of Φ with ρ, i.e., we can think of ρ as an e-morphism with a free variable representing Φ. The following construction makes this idea precise:

Definition 42. Consider a type theory with dependent products. Consider an e-morphism Φ and a logical relation ρ on it. By mutual induction we define an e-morphism Ψ and two families of substitutions $\varphi_\Gamma : \Gamma^\Phi \to \Gamma^\Psi$ and $\varphi^!_\Gamma : \Gamma^\rho \to \Gamma^\Psi$:

$$A^{\Psi_\Gamma} = \langle \_ : A^{\Phi_\Gamma}[\varphi_\Gamma],\ ! : A^{\rho_\Gamma}[\varphi^!_\Gamma,\ \_] \rangle$$
$$t^{\Psi_\Gamma} = \langle t^{\Phi_\Gamma}[\varphi_\Gamma], t^{\rho_\Gamma}[\varphi^!_\Gamma] \rangle$$
$$\varphi_\cdot = \cdot \quad \text{and} \quad \varphi_{\Gamma,x:A} = \varphi_\Gamma,\ x.\_$$
$$\varphi^!_\cdot = \cdot \quad \text{and} \quad \varphi^!_{\Gamma,x:A} = \varphi^!_\Gamma,\ x.\_,\ x.!$$

where we use $\_$ and $!$ as convenient variable names.

We denote this e-morphism by $\Phi|\rho$.

We can understand Def. 42 by comparing it to Def. 15 for the case $n = 2$: Φ and ρ correspond to Φ^1 and Φ^2, and φ and $\varphi^!$ correspond to φ^1 and φ^2. The remaining complexity of Def. 42 over Def. 15 corresponds to the complexity of a dependent product type over a simple product type.

Another way to understand Def. 42 is that $A^{(\Phi|\rho)_\Gamma}$ is the subtype of A^{Φ_Γ} containing exactly those x for which the relation ρ holds, i.e., for which the type $A^{\rho_\Gamma}[x]$ is inhabited.

More precisely, if we have any type A and any judgment $B[x]$ for $x : A$, we can regard the type $\langle x : A, x^! : B[x] \rangle$ as the subtype of A containing exactly the $x : A$ for which $B[x]$ holds, i.e., for which $B[x]$ is inhabited. This works particularly well if we use a proposition-as-types representation with proof irrelevance, i.e., if the types $B[x]$ represents a proposition inhabited by proofs and if proofs of the same proposition are considered equal. In that case, we might opt to write the type $\langle x : A, x^! : B[x] \rangle$ as $\{x : A \,|\, B[x]\}$.

Applying this notation in to our setting, we obtain $A^{(\Phi|\rho)_\Gamma} = \{x : A^{\Phi_\Gamma} \,|\, A^{\rho_\Gamma}[x]\}$, which inspired the notation $\Phi|\rho$. Thus, the e-morphism $\Phi|\rho$ interprets every closed type A as the subtype of A^Φ defined by A^ρ. These subtypes are taken coherently: The properties of e-relations ensure that all subtypes are closed under all operations that are definable in Σ.

Theorem 43. *In the situation of Def. 42, $\Phi|\rho$ is an e-morphism.*

Proof. The argument proceeds in the same way as for Thm. 16. □

An example is provided by Ex. 65.

6 Connections between E-Morphisms

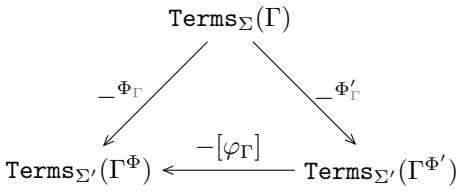
While signatures and signature morphisms form a category, it has proved very difficult to extend this to a 2-category. We will now see that a 2-categorical structure emerges very naturally at the level of e-morphisms.

6.1 Formal Definition

Since e-morphisms are functors, it is straightforward to define morphisms between them as natural transformations, which we will call *connections*:

Definition 44. Given two e-morphisms $\Phi, \Phi' : \Sigma \to \Sigma'$, a *connection* $\varphi : \Phi \Rightarrow \Phi'$ is a natural transformation from Φ' to Φ.

Thus, φ provides for every Σ-context Γ, a substitution $\vdash_{\Sigma'} \varphi_\Gamma : \Gamma^{\Phi'} \to \Gamma^\Phi$. As syntax transformations, we obtain the following (not in general commutative) diagram:



Remark 45. One may wonder why the above diagram should not commute. In fact, that would not make sense: Because $-^{\Phi_\Gamma}$ and $-^{\Phi'_\Gamma}$ preserve variables, the diagram can only commute if $-[\varphi_\Gamma]$ preserves variables, too. But that is not even well-typed except in the degenerate case where Φ and Φ' agree on types.

Remark 46. One may wonder why we use natural transformations from Φ' to Φ instead of the more natural opposite direction. Substitutions $\gamma : \Gamma' \to \Gamma'$ can be seen as mappings $\mathtt{Terms}_\Sigma(\Gamma') \to \mathtt{Terms}_\Sigma(\Gamma)$. But if we think of contexts Γ as types (e.g., via $\langle\Gamma\rangle$), then γ maps values from Γ to Γ'. It is this second intuition that we will exploit in Sect. 6.4 (which in turn motivated our investigation of connections between e-morphisms in the first place).

In the same way as for e-morphisms, we complement this abstract definition of connections with a more concrete one. In particular, Def. 44 and Thm. 47 correspond to Def. 7 and Thm. 9.

Theorem 47 (Characterization of Connections)**.** *Assume two e-morphisms Φ, Φ' and a family of substitutions φ_Γ from $\Gamma^{\Phi'}$ to Γ^Φ. Then the following are equivalent:*

1. *φ is a connection $\Phi \Rightarrow \Phi'$.*
2. *φ satisfies the following preservation properties:*
   (a) *φ_Γ is of the form*
   $$\varphi_\cdot = \cdot \qquad \text{and} \qquad \varphi_{\Gamma,x:A} = \varphi_\Gamma, \varphi^A_\Gamma$$
   for some terms φ^A_Γ.
   (b) *The following rule is admissible*
   $$\frac{\Gamma \vdash_\Sigma t : A}{\Gamma^\Phi \vdash_{\Sigma'} \varphi^A_\Gamma[id_{\Gamma^\Phi}, t^{\Phi_\Gamma}] = t^{\Phi'_\Gamma}[\varphi_\Gamma]}$$
   (c) *The following rule is admissible*
   $$\frac{\vdash_\Sigma \gamma : \Gamma \to \Gamma' \qquad \Gamma \vdash_\Sigma A : \mathtt{type}}{(\Gamma, x : A[\gamma])^\Phi \vdash_{\Sigma'} \varphi^A_\Gamma[\gamma^\Phi, x] = \varphi^{A[\gamma]}_{\Gamma'}}$$

Note that the three properties are analogous to those in Thm. 9. Their intuitions are as follows:

(a) φ_Γ maps variables in such a way that φ^x_Γ only depends on the type A of x and on the variables occurring in A. Note that φ is uniquely determined by the terms φ^A_Γ and vice versa.

(b) The typing judgment for the φ^A_Γ is
$$\Phi'^\Gamma, x : A^{\Phi_\Gamma} \vdash_{\Sigma'} \varphi^A_\Gamma : A^{\Phi'_\Gamma}[\varphi_\Gamma]$$
This is best understood by considering the special case where A is closed, i.e., Γ is empty, which yields
$$x : A^\Phi \vdash_{\Sigma'} \varphi^A_\cdot : A^{\Phi'}$$
Now we can think of $\varphi^A_\cdot$ as a function from A^Φ to $A^{\Phi'}$, and the property states that via this function all Σ-terms commute with e-morphism application:
$$\frac{\vdash_\Sigma t : A}{\vdash_{\Sigma'} \varphi^A_\cdot[t^\Phi] = t^{\Phi'}}$$

(c) This property states that φ preserves substitution: $\varphi^{A[\gamma]}_{\Gamma'}$ is already determined by φ^A_Γ and γ. We can also think of it as a commutativity property: applying first φ, then γ is the same as applying first γ, then φ.

Proof. Assume φ to be a connection.
The condition (a) follows by applying naturality to the inclusion substitution $\Gamma \to \Gamma, x : A$:

$$\begin{array}{ccc} \Gamma^{\Phi} & \xleftarrow{\varphi_\Gamma} & \Gamma^{\Phi'} \\ \downarrow & & \downarrow \\ (\Gamma, x:A)^{\Phi} & \xleftarrow[\varphi_{\Gamma,x:A}]{} & (\Gamma, x:A)^{\Phi'} \end{array}$$

The condition (b) follows by applying naturality to the substitution id_Γ, t from $\Gamma, x : A$ to Γ:

$$\begin{array}{ccc} \Gamma^{\Phi} & \xleftarrow{\varphi_\Gamma} & \Gamma^{\Phi'} \\ {\scriptstyle id_{\Gamma^\Phi}, t^{\Phi_\Gamma}}\uparrow & & \uparrow{\scriptstyle id_{\Gamma^{\Phi'}}, t^{\Phi'_\Gamma}} \\ (\Gamma, x:A)^{\Phi} & \xleftarrow[\varphi_\Gamma, \varphi^A_\Gamma]{} & (\Gamma, x:A)^{\Phi'} \end{array}$$

The condition (c) follows by applying naturality to the substitution γ, x from $\Gamma, x : A$ to $\Gamma', A[\gamma]$:

$$\begin{array}{ccc} (\Gamma', x:A[\gamma])^{\Phi} & \xleftarrow{\varphi_{\Gamma'}, \varphi^{A[\gamma]}_{\Gamma'}} & (\Gamma', x:A[\gamma])^{\Phi'} \\ {\scriptstyle \gamma^\Phi, x}\uparrow & & \uparrow{\scriptstyle \gamma^{\Phi'}, x} \\ (\Gamma, x:A)^{\Phi} & \xleftarrow[\varphi_\Gamma, \varphi^A_\Gamma]{} & (\Gamma, x:A)^{\Phi'} \end{array}$$

Conversely, assume the preservation properties. To show the naturality condition for a substitution γ from Γ to Γ', we proceed by induction on the structure of γ and Γ. This is possible because of (a).

If $\vdash_{\Sigma'} \cdot : \cdot \to \Gamma'$, then also $\varphi_\cdot = \cdot$, and the naturality condition becomes trivial.

For the case $\vdash_{\Sigma'} \gamma, t : \Gamma, x : A \to \Gamma'$, the induction hypothesis is the left diagram below and the needed result the right one:

$$\begin{array}{ccc} \Gamma'^{\Phi} & \xleftarrow{\varphi_{\Gamma'}} & \Gamma'^{\Phi'} \\ {\scriptstyle \gamma^\Phi}\uparrow & & \uparrow{\scriptstyle \gamma^{\Phi'}} \\ \Gamma^{\Phi} & \xleftarrow[\varphi_\Gamma]{} & \Gamma^{\Phi'} \end{array} \qquad \begin{array}{ccc} \Gamma'^{\Phi} & \xleftarrow{\varphi_{\Gamma'}} & \Gamma'^{\Phi'} \\ {\scriptstyle \gamma^\Phi, t^{\Phi_{\Gamma'}}}\uparrow & & \uparrow{\scriptstyle \gamma^{\Phi'}, t^{\Phi'_{\Gamma'}}} \\ (\Gamma, x:A)^{\Phi} & \xleftarrow[\varphi_\Gamma, \varphi^A_\Gamma]{} & (\Gamma, x:A)^{\Phi'} \end{array}$$

We only have to show

$$\Gamma'^{\Phi} \vdash_{\Sigma'} \varphi^A_\Gamma[\gamma^\Phi, t^{\Phi_{\Gamma'}}] = t^{\Phi'_{\Gamma'}}[\varphi_{\Gamma'}]$$

And that follows using (b) and (c) after observing that $\gamma^\Phi, t^{\Phi_{\Gamma'}} = id_{\Gamma'^\Phi}, t^{\Phi_{\Gamma'}} \circ \gamma^\Phi, x$ (where the intermediate context in the composition of substitutions is $(\Gamma', x : A[\gamma])^\Phi$). □

Remark 48. Our connections could be called e-connections because they must be defined individually at every context. But contrary to e/i-morphisms, we do not have a notion of i-connection that is defined inductively.

This is related to a well-known negative result in higher-order type theories. Connections have function-flavor in that they induce mappings between sets of terms. But when trying to define an i-connection between i-morphisms inductively at a function type, the function-flavor requires defining a function from $A^\Phi \to B^\Phi$ to $A^{\Phi'} \to B^{\Phi'}$, given functions $A^\Phi \to A^{\Phi'}$ and $B^\Phi \to B'^\Phi$. This is not in general possible.

This problem was one of the original motivations [Rey74] to study (binary) logical relations. These have relation-flavor in that they induce relations between sets of terms. And defining a relation between $A^\Phi \to B^\Phi$ and $A^{\Phi'} \to B^{\Phi'}$, given relations on $A^\Phi \times A^{\Phi'}$ and $B^\Phi \times B'^\Phi$ is possible. Working this out leads to the notion

of i-relations on the e-morphism $\Phi \times \Phi'$. However, even such binary i-relations do not compose, which can be remedied by considering binary e-relations akin to [PPST00].

6.2 Concrete Connections

Theorem 49. *In the situation of Def. 15, every φ^i is a connection $\Phi^1 \times \ldots \times \Phi^n \Rightarrow \Phi^i$.*

Theorem 50. *In the situation of Def. 42, φ is a connection $\Phi|\rho \Rightarrow \Phi$.*

All proofs are straightforward.

Because the e-morphisms from Sect. 4.2 are defined point-wise, we expect that isomorphisms in the category $\texttt{Cons}(\Sigma')$ induce isomorphisms in the category of e-morphisms from Σ to Σ'. This is indeed the case:

Theorem 51. *The e-morphisms $\Sigma \to \Sigma'$ are subject to the following isomorphic relations:*

- *The product $\Phi \times \Phi'$ is a commutative monoid (up to $\cong$) whose neutral element is the unit e-morphism.*
- *The monoid of contexts under reverse concatenation acts on e-morphisms via exponentiation:*

$$\Phi^{\cdot} \cong \Phi$$

$$\Phi^{\Gamma,\Gamma'} \cong (\Phi^{\Gamma'})^{\Gamma}$$

- *Exponentiation distributes over product:*

$$(\Phi \times \Phi')^{\Gamma} \cong \Phi^{\Gamma} \times \Phi'^{\Gamma}$$

Proof. Giving the connections that mediate the isomorphisms is straightforward. □

We sketch one additional result without formalizing it:

Remark 52. Consider a type theory with a type T_n that contains exactly n unrelated elements and with a case distinction operator that turns n terms of type A into a function $T_n \to A$. (For example, T_n can be defined as the disjoint union of n unit types.) Independently, let $\Phi^n = \Phi \times \ldots \times \Phi$ be the n-th power of Φ.
Then $\Phi^{x:T_n} \cong \Phi^n$.

6.3 Connections as 2-Cells

We can wrap up many of the abstract properties of signatures, e-morphisms, and connections by showing that they form a 2-category.

Theorem 53. *The signatures and the e-morphisms between them form a category.*

Proof. We only have to observe that the identity functor and the composition of e-morphisms are e-morphisms. □

Theorem 54. *The e-morphisms from Σ to Σ' and the connections between them form a category.*

Proof. This follows immediately from the properties of functors and natural transformations. □

Theorem 55. *In the presence of product types, the category of e-morphisms and connections has finite products (including a terminal object as the empty product).*

Proof. The product and its projections are the ones constructed in Def. 15. □

Remark 56. The signatures and i-morphisms form a broad subcategory of the category of signatures and e-morphisms. But the product of i-morphisms is usually not an i-morphism.

Theorem 57. *The signatures, e-morphisms, and connections form a 2-category.*

Proof. This follows immediately from the properties of categories, functors, and natural transformations. □

6.4 Connections as Model Morphisms

Motivation We think of e-morphisms $\Sigma \to F$ as models (or interpretation functions) of Σ. For every e-morphism Φ, we consider

- the F-type A^Φ as the universe that Φ assigns to A,
- the closed F-terms $u : A^\Phi$ as the values of that universe,
- the F-term $t^\Phi : A^\Phi$ as the value that the interpretation Φ assigns to $t : A$,
- the F-substitutions $\vec{u} : \Gamma^\Phi \to \cdot$ as tuples of values,
- if A has free variables from Γ, the F-type $A^{\Phi_\Gamma}[\vec{u}]$ as the family of universes indexed by tuples $\vec{u} : \Gamma^\Phi \to \cdot$ that Φ assigns to A.
- if t has free variables from Γ, the F-type $t^{\Phi_\Gamma}[\vec{u}] : A^{\Phi_\Gamma}[\vec{u}]$ as the family of values indexed by tuples $\vec{u} : \Gamma^\Phi \to \cdot$ that Φ assigns to $t : A$.

Accordingly, we think of a connection $\varphi : \Phi \Rightarrow \Phi'$ as a model morphism from Φ to Φ'. Thus, we expect

- for a Σ-type A, a map from the Φ-universe A^Φ to the Φ'-universe $A^{\Phi'}$.
  This map will be provided by applying the substitution $\varphi^A_\cdot : (x : A)^{\Phi'} \to (x : A)^\Phi$, i.e., mapping u to $\varphi^A_\cdot[u]$.
- a component-wise map of Φ-tuples $\vec{u} : \Gamma^\Phi \to \cdot$ to Φ'-tuples of the corresponding universes.
  This map will be provided by the composition $\varphi_\Gamma[\vec{u}]$ of substitutions.
- if A has free variables from Γ, a $\vec{u}$-indexed family of maps from the Φ-universe $A^{\Phi_\Gamma}[\vec{u}]$ to the corresponding Φ'-universe $A^{\Phi'_\Gamma}[\varphi_\Gamma[\vec{u}]]$.
  This map will be provided by applying the substitution $\varphi^A_\Gamma : \Gamma^{\Phi'} \to \Gamma^\Phi$.

Formal Definition The following definitions and theorem makes these ideas precise:

Definition 58. Given a signature Σ and a signature F. An *F-model* M of Σ is an e-morphism $\Sigma \to F$.
For every $\Gamma \vdash_F A : \mathtt{type}$ and $\vdash_F \vec{u} : \Gamma^M \to \cdot$, we call $A^M[\vec{u}]$ the *universe* of A in M at $\vec{u}$.

Definition 59. Given two F-models M, N, an *F-model morphism* φ from M to N is a connection $M \Rightarrow N$.
For every $\Gamma \vdash_\Sigma A : \mathtt{type}$ and $\vdash_F \vec{u} : \Gamma^M \to \cdot$, we call

$$v : A^{M_\Gamma}[\gamma] \vdash_F \varphi^A_\Gamma[\vec{u}, v] : A^{N_\Gamma}[\varphi_\Gamma[\vec{u}]]$$

the *map* of A in φ at $\vec{u}$.

$$\begin{array}{ccccc}
\cdot & \xleftarrow{\vec{u}} & \Gamma^M & \xleftarrow{\varphi_\Gamma} & \Gamma^N \\
\downarrow & & \downarrow & & \downarrow \\
u : A^{M_\Gamma}[\vec{u}] & \xleftarrow{\vec{u}, v} & (\Gamma, x : A)^M & \xleftarrow{\varphi_{\Gamma, x:A}} & (\Gamma, x : A)^N
\end{array}$$

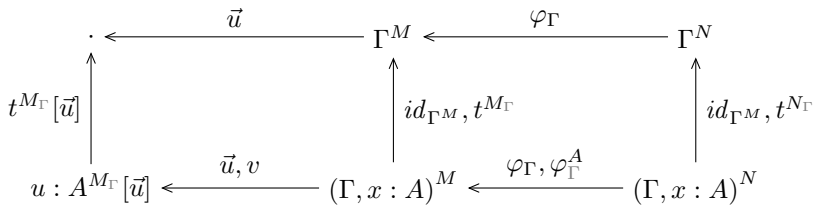
It remains to establish the typical homomorphism property known from models of first-order logic:

Theorem 60. *In the situation of Def. 59, for every term* $\Gamma \vdash_\Sigma t : A$ *and every* $\vdash_F \vec{u} : \Gamma^\Phi \to \cdot$

$$\vdash_F \varphi_\Gamma^A[\vec{u}, t^{M_\Gamma}[\vec{u}]] = t^{N_\Gamma}[\varphi_\Gamma[\vec{u}]]$$

which is an equality between terms of type $A^{N_\Gamma}[\varphi_\Gamma[\vec{u}]]$.

Proof. The result follows by applying the substitution $\vec{u}$ to the condition (b) in Thm. 47.



□

If we specialize Thm. 60 to the case of closed types, i.e., empty Γ, we indeed obtain the well-known condition that model morphisms commute with interpretation in the models:

$$\vdash_F \varphi^A[t^M[\vec{u}]] = t^N[\varphi[\vec{u}]]$$

The Category of Models Thus, we obtain a category of F-models and F-model morphisms for every signature Σ. Moreover, we can generalize some model constructions from universal algebra to the logic-independent level:

- If we have product types, the category of F-models is closed under products.
- If we have dependent product types, the category of F-models is closed under sub-models: A logical relation is used to characterize the sub-universes making up the sub-model. The associated connection $\varphi : \Phi|\rho \Rightarrow \Phi$ yields the inclusion model morphism. The product of e-relations yields the intersection of sub-models.
- Binary e-relations on an e-morphism Φ correspond to congruence relations of Φ seen as a model. The product of e-relations corresponds to the intersection of congruences. We conjecture that in the presence of (a reasonable formulation of) quotient types, we can obtain the quotient model from a congruence.

The value of these constructions is they work for a wide variety of type theories and work coherently at arbitrary higher types. We can think of this as the beginnings of a *universal universal algebra*, which works not only with an arbitrary theory but also with an arbitrary logic.

Henkin Models Henkin models of type theories do not require a compositional definition of the universe at higher types. Thus, the universe $(A \to B)^M$ does not have to be equal to $(B^M)^{A^M}$. Instead, the condition is relaxed to $(A \to B)^M \subseteq (B^M)^{A^M}$.

This yields completeness because one can form an initial model in which $(A \to B)^M$ is the set of all functions that can be expressed as λ-terms, which is usually smaller (e.g., necessarily countable) than the whole function set. This comes at the price of dropping compositionality so that the universe has to be defined individually for every (out of infinitely many) types.

This leads to the analogy

models are to Henkin-models as i-morphisms are to e-morphisms

At first sight, one may think that the analogy is not quite correct: In a Henkin model, $(A \to B)^M$ must be a subset of $(B^M)^{A^M}$, but in an e-morphism, $(A \to B)^\Phi$ does not have to be a subtype of $A^\Phi \to B^\Phi$. However, using the canonical embedding $f \mapsto f^*$ from Sect. 4.3, we see that $(A \to B)^\Phi$ is effectively a subtype of $A^\Phi \to B^\Phi$.

Thus, we can say that e-morphisms permit type theoretical representations of Henkin models. It remains to ask whether this includes a type theoretical representation of the initial Henkin model. We come back to this question in Sect. 8.

7 Example

In this section, we apply our result to represent and verify a logic translation in a logical framework. We will work with a type theory containing all features from Sect. 3.

We will use implication and universal quantifier as the only connectives of first-order logic because they exhibit all the critical behaviors already.

Example 61 (First-Order Logic). The signature FOL contains the following declarations:

$$\begin{array}{lcl} i & : & \texttt{type} \\ o & : & \texttt{type} \\ ded & : & o \to \texttt{type} \\ imp & : & o \to o \to o \\ forall & : & (i \to o) \to o \end{array}$$

Example 62 (Sorted First-Order Logic). The signature $SFOL$ contains the following declarations:

$$\begin{array}{lcl} sort & : & \texttt{type} \\ tm & : & sort \to \texttt{type} \\ o & : & \texttt{type} \\ ded & : & o \to \texttt{type} \\ imp & : & o \to o \to o \\ forall & : & \Pi_{s:sort}\,(tm\,s \to o) \to o \end{array}$$

Example 63 (Type-Erasure and Relativization as an I-Morphism). The i-morphism ER (erase and relativize) contains the following declarations:

$$\begin{array}{lcl} sort & := & i \to o \\ tm & := & \lambda_{s:i\to o}\, i \\ o & := & o \\ ded & := & ded \\ imp & := & imp \\ forall & := & \lambda_{s:i\to o}\,\lambda_{F:i\to o}\, forall\,\lambda_{x:i}\,(s\,x)\,imp\,(F\,x) \end{array}$$

Example 64 (Type Preservation as an I-Relation). The i-relation TP (type preservation) on ER contains the following declarations:

$$\begin{array}{lcl} sort & := & \lambda_{s:i\to o}\,\langle\cdot\rangle \\ tm & := & \lambda_{s:i\to o}\,\lambda_{s^*:\langle\cdot\rangle}\,\lambda_{x:i}\, ded\,(s\,x) \\ o & := & \lambda_{x:o}\,\langle\cdot\rangle \\ ded & := & \lambda_{F:o}\,\lambda_{F^*:\langle\cdot\rangle}\,\lambda_{p:ded\,F}\,\langle\cdot\rangle \\ imp & := & \lambda_{F:o}\,\lambda_{F^*:\langle\cdot\rangle}\,\lambda_{G:o}\,\lambda_{G^*:\langle\cdot\rangle}\,\langle\cdot\rangle \\ forall & := & \lambda_{s:i\to o}\,\lambda_{s^*:\langle\cdot\rangle}\,\lambda_{F:i\to o}\,\lambda_{F^*:\Pi_{x:i}\,\Pi_{x^*:ded\,(s\,x)}\,o}\,\langle\cdot\rangle \end{array}$$

Note how all cases take arguments in pairs (x, x^*) except for the type constructors, which take one additional argument. Our of our 4 type constructors, all but tm are mapped trivially to the unit type, i.e., the i-relation makes no assumptions on the translation of terms of these types. Consequently, all cases for term constructors that return sorts, formulas, or proofs are trivial and just return the unit term.

Only tm is mapped non-trivially: This case represents the assertion that every $\vdash_{SFOL} t : tm\,S$ is translated to a proof $\vdash_{FOL} t^{TP} : ded\,(S^{ER}\,x)$, i.e., that TP maps $SFOL$-terms of sort S to FOL-terms for which S^{ER} holds.

Example 65 (Type Preservation as an E-Morphism). We can now form the e-morphism $ER|TP$ which combines ER and TP. It captures both aspects of the translation: types are erased but preserved.

Every term

$$\vdash_{SFOL} t : tm\,S$$

is translated to a pair

$$\vdash_{FOL} \langle t^{TP}, t^{ER}\rangle : \langle \_ : i, ! : ded\,(S^{ER}\,\_)\rangle$$

where the first component is the type erasure translation and the second the preservation proof.

More generally, every function

$$\vdash_{SFOL} f : tm\, S \to tm\, T$$

is translated to a pair

$$\vdash_{FOL} \langle f^{TP}, f^{ER} \rangle : \langle \_ : i \to i,\ ! : \Pi_{x:i}\, ded\,(S^{ER}\, x) \to ded\,(T^{ER}\,(\_\, x)) \rangle$$

Here the first component is the function $i \to i$ that result from applying type erasure to the function f, and the second component is the proof that this function preserves typing.

We will now extend FOL and $SFOL$ with declarations their proof theory. We represent the natural deduction introduction and elimination rules for implication and universal quantification. These representations are standard (see, e.g., [HHP93]).

Example 66 (First-Order Logic Proof Theory). The signature $FOLPF$ extends FOL with the following declarations:

$$\begin{array}{lcl} impI & : & \Pi_{F:o}\, \Pi_{G:o}\, (ded\, F \to ded\, G) \to ded\,(F\, imp\, G) \\ impE & : & \Pi_{F:o}\, \Pi_{G:o}\, ded\,(F\, imp, G) \to ded\, F \to ded\, G \\ forallI & : & \Pi_{F:i\to o}\, (\Pi_{x:i}\, ded\,(F\, x)) \to ded\,(forall\, \lambda_{x:i}\,(F\, x)) \\ forallE & : & \Pi_{F:i\to o}\, ded\,(forall\, \lambda_{x:i}\,(F\, x)) \to \Pi_{x:i}\, ded\,(F\, x) \end{array}$$

Example 67 (Sorted First-Order Logic). The signature $SFOLPF$ contains the following declarations:

$$\begin{array}{lcl} impI & : & \Pi_{F:o}\, \Pi_{G:o}\, (ded\, F \to ded\, G) \to ded\,(F\, imp\, G) \\ impE & : & \Pi_{F:o}\, \Pi_{G:o}\, ded\,(F\, imp, G) \to ded\, F \to ded\, G \\ forallI & : & \Pi_{s:sort}\, \Pi_{F:tm\, s\to o}\, (\Pi_{x:tm\, s}\, ded\,(F\, x)) \to ded\,(forall \lambda_{x:tm\, s}\,(F\, x)) \\ forallE & : & \Pi_{s:sort}\, \Pi_{F:tm\, s\to o}\, ded\,(forall \lambda_{x:tm\, s}\,(F\, x)) \to \Pi_{x:tm\, s}\, ded\,(F\, x) \end{array}$$

It is important to realize that we cannot extend ER to an i-morphism $SFOLPF \to FOLPF$. The problem is the case for $forallE$: The required type would be

$$\Pi_{s:i\to o}\, \Pi_{F:i\to o}\, ded\,(forall\, \lambda_{x:i}\,(s\, x)\, imp\,(F\, x)) \to \Pi_{x:i}\, ded\,(F\, x)$$

Thus, we have to prove $F\, x$ for an arbitrary x, but our assumption about F only states $(s\, x)\, imp\,(F\, x)$ – we are missing the assumption $s\, x$. That is exactly the type information that has been erased by ER. We know from TP that the type is preserved, but we cannot ER at this point. A similar problem would occur with the existential introduction rule.

Neither is it possible to change ER such that it maps terms to pairs by using the declarations

$$\begin{array}{lcl} sort & := & i \to o \\ tm & := & \lambda_{s:i\to o}\, \langle \_ : i,\ ! : ded\,(s\, \_) \rangle \\ o & := & o \\ ded & := & ded \end{array}$$

This approach already fails at the case for $forall$: The required type would be

$$\Pi_{s:i\to o}\, \Pi_{F:\langle \_:i,\ !:ded\,(s\, \_)\rangle \to o}\, o$$

Thus, we have to build a formula out of F but we can only apply F to pairs of a $t : i$ and a proof of $s\, t$. But we have no way to come by such a proof. (And even if we had, the resulting translation function would not be the same as the one we set out to formalize.)

We can now solve this problem by using the e-morphism $ER|TP$, which first defines the morphism and then bundles every term with its typing information. This yields:

Example 68 (Truth Preservation as an I-Extension of an E-Morphism). $SFOLPF$ extends $SFOL$ only with typed declarations. Therefore, we can form an e-morphism $(ER|TP), PP$ where PP contains the following declarations

$$
\begin{array}{lll}
forallI & : & \lambda_{s:\langle\_:i\to o,\,!:-\rangle}\,\lambda_{F:\langle\_:i\to o,\,!:-\rangle}\,\lambda_{p:\Pi_{x:\langle\_:i,\,!:ded\,(s.\_\_)\rangle}\,\langle\_:ded\,(F.\_\,x.\_),\,!:-\rangle} \\
 & & \quad forallI\ -\ \lambda_{x:\langle\_:i,\,!:-\rangle}\ impI\ -\ -\lambda_{q:\langle\_:ded\,(s.\_\,x.\_),\,!:-\rangle}\,p.\_\langle x.\_,\,q.\_\rangle \\
forallE & : & \lambda_{s:\langle\_:i\to o,\,!:-\rangle}\,\lambda_{F:\langle\_:i\to o,\,!:-\rangle}\,\lambda_{p:\langle\_:ded\,(forall\lambda_{x:i}\,(s.\_\,x.\_)\,imp\,(F.\_\,x.\_)),\,!:-\rangle}\,\lambda_{x:\langle\_:i,\,!:ded\,(s.\_\_)\rangle} \\
 & & \quad impE\ -\ -\,(forallE\ -\ p.\_\ x.\_)\,x.!
\end{array}
$$

Here we have written $-$ for all assumptions that are not needed and for the inferable arguments of proof rules. The two cases for implication are trivial.

8 Towards Reflection as an Initial E-Morphism

The original motivation of our work was to study type theories with reflection. Our basic idea is that a signature Σ' should be able to reflect the syntax of another signature Σ. For example, Σ' could define inductive data types containing exactly the Σ-expressions.

However, such a reflection operation proved tricky in the presence of function types. With the terminology that is now at our disposal, we can capture our difficulties succinctly: Reflection is an e-morphism but not an i-morphism. Therefore, we have developed the theory of e-morphisms first, leading to the present results.

The development of reflection on top of this work requires a substantial independent effort, and therefore we refrain from spelling it out here. Nonetheless, we briefly sketch the main idea.

Definition 69. We say that Σ' reflects Σ if Σ' has

- for every Σ-type A, a type $\ulcorner A \urcorner$,
- for every Σ-term $t : A$, a term $\ulcorner t \urcorner : \ulcorner A \urcorner$,
- an induction principle that makes the $\ulcorner t \urcorner$ exactly the terms terms of type $\ulcorner A \urcorner$.

We expect the following two theorems to become provable:

Theorem 70. *If Σ' reflects Σ, then the reflection operator induces an e-morphism $Q : \Sigma \to \Sigma'$.*

Theorem 71. *Q is initial in the category of e-morphism $\Sigma \to \Sigma'$.*

Of course, their truth depends on the exact way reflection is realized in the type theory. A particularly appealing choice is to use e-morphisms $\Phi : \Sigma \to \Sigma'$ as the induction operators on the type $\ulcorner A \urcorner$. This guarantees initiality in the sense that the induction principle becomes by definition strong enough to define the universal connection $Q \Rightarrow \Phi$ whenever the type theory is strong enough to define Φ.

9 Practical Considerations

The major drawback of e-morphisms and e-relations over their i-counterparts is the lack of effective representation in computer system. The i-concepts can be represented easily as finite lists of pairs $c \mapsto E$. But the e-concepts can be arbitrary maps between infinite sets of expressions, and the application function E^{Φ_Γ} can be implemented by induction on the representation of expressions.

Technically, this is a conjecture that the author has not investigated. But it is plausible that e-morphisms are uncountable or at least undecidable and that their application may be uncomputable.

Therefore, we can only implement approximations. But here the constructions obtained in the preceding sections already provide a rich language of e-morphisms.

Concretely, we can extend our grammar with the following productions:

	Grammar	Typing judgment
E-morphisms	$\Phi\ ::=\ \{\sigma\} \mid \top \mid \Phi \times \Phi \mid \Phi \vert P \mid \Phi^\Gamma$	$\vdash \Phi :^e \Sigma \to \Sigma'$
I-morphisms	$\sigma\ ::=\ \cdot \mid \sigma,\ c \mapsto E$	$\vdash \sigma : \Sigma \to \Sigma'$
E-relations	$P\ ::=\ \{\rho\} \mid \top \mid P \times P \mid P \wedge P$	$\vdash \rho \leq^e \Phi : \Sigma \to \Sigma'$
I-relations	$\rho\ ::=\ \cdot \mid \rho,\ c \mapsto E$	$\vdash \rho \leq^i \Phi : \Sigma \to \Sigma'$

It is then straightforward to add typing rules (and congruence rules) for the new constructors.
Moreover, we can use e-morphism and e-relation application as additional (admissible) expression constructors:

$$E ::= E^{\Phi} \mid E^{P}$$

10 Conclusion

References

[BJP12] J. Bernardy, P. Jansson, and R. Paterson. Proofs for Free - Parametricity for Dependent Types. *Journal of Functional Programming*, 22(2):107–152, 2012.

[Bou64] N. Bourbaki. Univers. In *Séminaire de Géométrie Algébrique du Bois Marie - Théorie des topos et cohomologie étale des schémas*, pages 185–217. Springer, 1964.

[BP13] J. Blanchette and A. Popescu. Mechanizing the metatheory of sledgehammer. In *FroCos*. Springer, 2013. to appear.

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

[CHK+11] 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, editors, *Intelligent Computer Mathematics*, pages 289–291. Springer, 2011.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

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

[FGT93] W. Farmer, J. Guttman, and F. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11(2):213–248, 1993.

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

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

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

[HR11] F. Horozal and F. Rabe. Representing Model Theory in a Type-Theoretical Logical Framework. *Theoretical Computer Science*, 412(37):4919–4945, 2011.

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

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

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

[MP08] J. Meng and L. Paulson. Translating Higher-Order Clauses to First-Order Clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008.

[MTHM97] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML,* Revised edition. MIT Press, 1997.

[OS06] S. Obua and S. Skalberg. Importing HOL into Isabelle/HOL. In N. Shankar and U. Furbach, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.

[PPST00] G. Plotkin, J. Power, D. Sannella, and R. Tennent. Lax Logical Relations. In *Colloquium on Automata, Languages and Programming*, volume 1853 of *LNCS*, pages 85–102. Springer, 2000.

[Rab13] F. Rabe. A Logical Framework Combining Model and Proof Theory. *Mathematical Structures in Computer Science*, 2013. to appear; see http://kwarc.info/frabe/Research/rabe_combining_10.pdf.

[Rey74] J. Reynolds. On the relation between direct and continuation semantics. In *Second Colloq. Automata, Languages and Programming*, LNCS, pages 141–156. Springer, 1974.

[RK13] F. Rabe and M. Kohlhase. A Scalable Module System. *Information and Computation*, 2013. to appear; see http://kwarc.info/frabe/Research/RK_mmt_10.pdf.

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

[RS13] F. Rabe and K. Sojakova. Logical Relations for a Logical Framework. *ACM Transactions on Computational Logic*, 2013. to appear; see http://kwarc.info/frabe/Research/RS_logrels_12.pdf.

[SS04] 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*, 2004.

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