A Type Theory Based on Reflection

Florian Rabe

Jacobs University, Bremen, Germany

Abstract. This paper is a partially finished and completely unpolished work-in-progress. The content is very interesting but still very hard to read.

1 Introduction

2 A Basic Type Theory

	Grammar	Typing judgment	Equality judgment
Signatures	$\Sigma ::= \cdot \mid \Sigma, c : E$	$\vdash \Sigma$ `Sig`	
Morphisms	$\sigma ::= \cdot \mid \sigma, c \mapsto E$	$\Gamma \vdash \sigma : \Sigma' \to \Sigma$	$\Gamma \vdash \sigma = \sigma' : \Sigma' \to \Sigma$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x : E$	$\Gamma \vdash_\Sigma \Gamma'$ `Ctx`	
Substitutions	$\gamma ::= \cdot \mid \gamma, E$	$\Gamma \vdash_\Sigma \gamma : \Gamma_1 \to \Gamma_2$	$\Gamma \vdash_\Sigma \gamma = \gamma' : \Gamma_1 \to \Gamma_2$
Expressions	$E ::= c \mid x \mid \texttt{type}^n$ for $n = 1, 2, \ldots$	$\Gamma \vdash_\Sigma E : E'$	$\Gamma \vdash_\Sigma E = E'$

Additional meta-variables for expressions: t and A for expressions

Fig. 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. Its rules are given in Fig. 2 and 3.

Our type theory is empty in the sense that it does not contain any complex expressions. The only expressions over a signature Σ in a context Γ are the constants c and the variables x declared in Σ and Γ as well as the type universes. We will occasionally use the meta-variables t and A instead of E for expressions on the left (usually terms) or right (usually types) of the :-judgment.

Instead of complex expressions, our language defines the general structure of a type theory. It provides signatures Σ declaring typed constants and relative to such a signature contexts Γ declaring typed variables. Both signatures and contexts are ordered, and once declared, a constant or variable may occur in the types of any later declaration.

Signatures and contexts come with homomorphic mappings: morphisms map between signatures and substitutions between contexts. A morphism $\Gamma \vdash \sigma : \Sigma' \to \Sigma$ assigns a Σ-Γ-expression of the appropriate type to every Σ'-constant.

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

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

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

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

$$\frac{\vdash_\Sigma \Gamma\ \texttt{Ctx} \quad c : E \text{ in } \Sigma}{\Gamma \vdash_\Sigma c : E} \qquad \frac{\vdash_\Sigma \Gamma\ \texttt{Ctx} \quad x : E \text{ in } \Gamma}{\Gamma \vdash_\Sigma x : E} \qquad \frac{\vdash_\Sigma \Gamma\ \texttt{Ctx}}{\Gamma \vdash_\Sigma \texttt{type}^n : \texttt{type}^{n+1}}$$

Fig. 2. Basic Typing Rules

$$\frac{\vdash_\Sigma \Gamma\ \texttt{Ctx}}{\Gamma \vdash \cdot = \cdot : \cdot \to \Sigma} \qquad \frac{\Gamma \vdash \sigma_1 = \sigma_2 : \Sigma' \to \Sigma \quad \Gamma \vdash_\Sigma t_1 = t_2}{\Gamma \vdash \sigma_1, c \mapsto t_1 = \sigma_2, c \mapsto t_2 : \Sigma', c : A \to \Sigma}$$

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

$$\frac{\vdash_\Sigma \Gamma\ \texttt{Ctx}}{\Gamma \vdash_\Sigma c = c} \qquad \frac{\vdash_\Sigma \Gamma\ \texttt{Ctx}}{\Gamma \vdash_\Sigma x = x} \qquad \frac{\vdash_\Sigma \Gamma\ \texttt{Ctx}}{\Gamma \vdash_\Sigma \texttt{type}^n = \texttt{type}^n}$$

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

Fig. 3. Basic Equality Rules

The assigned expressions may make use of the variables in Γ. A substitution $\Gamma \vdash_\Sigma \gamma : \Gamma_1 \to \Gamma_2$ translates from Γ_1 to Γ_2, i.e., it provide a Γ_2-expression of the appropriate type for each variable in Γ_1. Both Γ_1 and Γ_2 may depend on Γ, and γ maps each variables from Γ to itself. In both case, equality is defined component-wise.

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

Remark 1. The judgments for morphisms, contexts, and substitutions are unusual: The parameter Γ is usually not used. For contexts and substitutions, Γ can be dropped without loss of generality: As we will see in Def. 1, the judgments with Γ are indeed defined in terms of their counterparts without Γ just abbreviations.

For morphisms, the situation is similar: A morphism in context Γ is equivalent to a morphism where the declaration in Γ are added to the codomain.

The extra context is introduced here already to smoothen the definitions of the reflection operations later on: There, Γ will be the context in which reflection occurs, and morphisms $\sigma : \Sigma' \to \Sigma$ and substitutions $\gamma : \Gamma_1 \to \Gamma_2$ are objects that are reflected.

As we will see, it is desirable to permit free variables from Γ in reflected entities.

Definition 1. *We define the following 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. *Given $\Gamma = x_1 : A_1, \dots, x_n : A_n$, we define*

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

Given $\Gamma \vdash_\Sigma \gamma : \Gamma_1 \to \Delta$ and $\Gamma \vdash_\Sigma \Gamma_1, \Gamma_2 \ \texttt{Ctx}$, we define

$$\gamma^* := \gamma, id_{\Gamma_2}$$

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:*

$$\begin{array}{lll} \gamma(x_i) & := t_i & \text{if } \gamma = t_1, \dots, t_n \text{ and } \Gamma = x_1 : A_1, \dots, x_n : A_n \\ \gamma(c) & := c & \\ \gamma(\texttt{type}^n) & := \texttt{type}^n & \end{array}$$

- *for a context Γ' such that $\Gamma \vdash_\Sigma \Gamma' \ \texttt{Ctx}$:*

$$\begin{array}{ll} \gamma(\cdot) & := \cdot \\ \gamma(\Gamma', x : A) & := \gamma(\Gamma'), x : \gamma^*(A) \end{array}$$

– *for a substitution* γ'*:*

$$\begin{aligned}\gamma(\cdot) &:= \cdot \\ \gamma(\gamma', t) &:= \gamma(\gamma'), \gamma(t)\end{aligned}$$

Given $\Gamma \vdash_\Sigma \gamma : \Gamma' \to \Delta$*, we define the application* $\gamma(-)$ *as an abbreviation of* $(id_\Gamma, \gamma)(-)$*.*

The basic properties of contexts and substitutions are well-known, and we will state the most important ones that will be used frequently later on:

Lemma 1. *The following rules are admissible*

$$\frac{\vdash_\Sigma \Gamma\ \mathtt{Ctx}}{\vdash_\Sigma id_\Gamma : \Gamma \to \Gamma} \qquad \frac{\vdash_\Sigma \gamma : \Gamma \to \Delta \qquad \vdash_\Sigma \delta : \Delta \to \Theta}{\vdash_\Sigma \delta(\gamma) : \Gamma \to \Theta}$$

$$\frac{\vdash_\Sigma \gamma : \Gamma \to \Delta \qquad \Gamma \vdash_\Sigma \Gamma'\ \mathtt{Ctx}}{\Delta \vdash_\Sigma \gamma(\Gamma')\ \mathtt{Ctx}} \qquad \frac{\vdash_\Sigma \gamma : \Gamma \to \Delta \qquad \Gamma \vdash_\Sigma \gamma' : \Gamma_1 \to \Gamma_2}{\Delta \vdash_\Sigma \gamma(\gamma') : \gamma(\Gamma_1) \to \gamma(\Gamma_2)}$$

$$\frac{\vdash_\Sigma \gamma : \Gamma \to \Delta \qquad \Gamma \vdash_\Sigma t : A}{\Delta \vdash_\Sigma \gamma(t) : \gamma(A)}$$

Remark 2. Readers familiar with category theory will recognize the first four properties from Lem. 1 as the properties of a category of Σ-contexts; in particular, composition is given by substitution application. Contexts $\Gamma \vdash_\Sigma \Gamma_1\ \mathtt{Ctx}$ yield inclusion morphisms $\Gamma \hookrightarrow \Gamma, \Gamma'$, and together with the substitutions $\Gamma \vdash_\Sigma \gamma : \Gamma_1 \to \Gamma_2$, they form a subcategory. Moreover, this category of contexts has pushouts along inclusions: For a substitution $\vdash_\Sigma \gamma : \Gamma \to \Delta$, both the pushout of an inclusion out of Γ along γ to an inclusion out of Δ and the universal morphism out of it are given by $\gamma(-)$. Moreover, these pushouts are functorial, i.e., the pushouts are coherent (if we use de-Bruijn indices to avoid issues regarding variable capture). We omit the statement of the respective equality rules, which are all admissible.

We also have the analogue to Def. 2 for morphisms:

Definition 3. *Given* $\Gamma \vdash \sigma : \Sigma' \to \Sigma$*, 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:*

$$\begin{aligned}\sigma(x) &:= x \\ \sigma(c) &:= t \qquad \text{if } c \mapsto t \text{ in } \sigma\end{aligned}$$

The corresponding analogue to Lem. 2 also holds. We omit the details.

Finally, we make some basic definitions that let us talk about general type theories, i.e., extensions of our basic type theory:

Definition 4. *A formal language based on the grammar from Fig. 1, possibly extended with productions and rules, is called a* type theory*. If the production* $E ::= \mathtt{type}^n$ *is restricted to* $n = 1, \dots, l$*, it is called an* l-leveled *type theory.*

If $\Gamma \vdash_\Sigma E : E'$ *and* $\Gamma \vdash_\Sigma E' : \mathtt{type}^n$*, then* E *is called an* expression at level $n-1$*. Expressions at level* 0*,* 1*, and* 2 *are called* terms*,* type families*, and* kinds*, respectively.*

The level of a context $\Gamma \vdash_\Sigma \Gamma\ \mathtt{Ctx}$ *is the maximum of the levels of its variables. The* level of a substitution $\Gamma \vdash_\Sigma \gamma : \Gamma' \to \Gamma$ *is the level of* Γ'*.*

Most of our work is targeted at 2-levelled type theories in which contexts and substitutions are restricted to level 0. This includes in particular simple and dependent type theory. The most important type theories beyond this restriction are those with parametric polymorphism (contexts and substitutions at level 1) such as System F and the calculus of constructions (expressions at all levels).

3 Reflection based on Contexts

We will now add type formation operators to our basic type theory. The terms of these new types will be introduced in Church-style: We use an introduction form to construct new terms that populate the new type; and given a term of the new type, we use elimination forms to construct new terms that characterize their behavior. In particular, every term has a unique type that can be inferred from the context. The central intuition of our type operators is that the introduction forms arise by reflection: An existing meta-level object is reflected into the expression language.

First we will redevelop the well-known types of dependent functions and dependent products. Contrary to their common treatment, we will develop the equivalent formulation where the two are n-ary operators. This generalization is easily definable in terms of the usual form:

$$\begin{aligned} \Pi_{x_1\ A_1\ \dots\ x_n\ A_n} A &:= \Pi_{x_1:A_1} \dots \Pi_{x_n:A_n} A \\ \langle x_1\ A_1\ \dots\ x_n\ A_n \rangle &:= \Sigma_{x_1:A_1} \dots \Sigma_{x_{n-1}:A_{n-1}} A_n \end{aligned}$$

The type $\Pi_\Gamma A$ arises by reflecting terms of type A with free variables from Γ. And the type $\langle \Gamma \rangle$ arises by reflecting substitutions from Γ into the empty context. (We do not reflect substitutions into an arbitrary context because, as we will see in Ex. 1, these can be defined from this special case.) The syntax and rules for these two type operators are given in Fig. 4 and 5 and will be explained individually in the following.

We also extend the definitions of substitution and morphism application by a straightforward homomorphic extension:

Definition 5. *In the situation of Def. 2 and 3, we add the following cases to the definition of* $\gamma(-)$ *and* $\sigma(-)$*:*

Reflecting ...	... terms	... substitutions
Grammar	$E ::= \Pi_\Gamma A \mid \lambda_\Gamma t \mid f\,\gamma \mid \mathtt{ev}^\Gamma(f)$	$E ::= \langle\Gamma\rangle \mid \langle\gamma : \Gamma\rangle \mid u.t \mid \mathtt{ev}^\Gamma(u)$
Meta-variable for inhabitants	f	u
Type formation	$\dfrac{\Gamma, \Gamma' \vdash_\Sigma A : \mathtt{type}^n}{\Gamma \vdash_\Sigma \Pi_{\Gamma'} A : \mathtt{type}^n}$	$\dfrac{\Gamma \vdash_\Sigma \Gamma'\ \mathtt{Ctx}}{\Gamma \vdash_\Sigma \langle\Gamma'\rangle : \mathtt{type}}$
Introduction (reflection)	$\dfrac{\Gamma, \Gamma' \vdash_\Sigma t : A}{\Gamma \vdash_\Sigma \lambda_{\Gamma'} t : \Pi_{\Gamma'} A}$	$\dfrac{\Gamma \vdash_\Sigma \gamma : \Gamma' \to \cdot}{\Gamma \vdash_\Sigma \langle\gamma : \Gamma'\rangle : \langle\Gamma'\rangle}$
Elimination (behavior) inter-definable Evaluation	$\dfrac{\Gamma \vdash_\Sigma f : \Pi_{\Gamma'} A \quad \Gamma \vdash_\Sigma \gamma : \Gamma' \to \cdot}{\Gamma \vdash_\Sigma f\,\gamma : \gamma(A)}$ $\dfrac{\Gamma \vdash_\Sigma f : \Pi_{\Gamma'} A}{\Gamma, \Gamma' \vdash_\Sigma \mathtt{ev}^{\Gamma'}(f) : A}$	$\dfrac{\Gamma \vdash_\Sigma u : \langle\Gamma'\rangle \quad \Gamma, \Gamma' \vdash_\Sigma t : A}{\Gamma \vdash_\Sigma u.t : \mathtt{ev}^{\Gamma'}(u)(A)}$ $\dfrac{\Gamma \vdash_\Sigma u : \langle\Gamma'\rangle}{\Gamma \vdash_\Sigma \mathtt{ev}^{\Gamma'}(u) : \Gamma' \to \cdot}$

Fig. 4. Reflection based on Contexts: Typing Rules

$$
\begin{array}{lll}
\gamma(\Pi_{\Gamma'} A) & := \Pi_{\gamma(\Gamma')} \gamma^*(A) & \sigma(\Pi_{\Gamma'} A) := \Pi_{\sigma(\Gamma')} \sigma(A) \\
\gamma(\lambda_{\Gamma'} t) & := \lambda_{\gamma(\Gamma')} \gamma^*(t) & \sigma(\lambda_{\Gamma'} t) := \lambda_{\sigma(\Gamma')} \sigma(t) \\
\gamma(f\,\gamma') & := \gamma(f)\,\gamma(\gamma') & \sigma(f\,\sigma') := \sigma(f)\,\sigma(\sigma') \\
 & & \\
\gamma(\langle\Gamma'\rangle) & := \langle\gamma(\Gamma')\rangle & \sigma(\langle\Gamma'\rangle) := \langle\sigma(\Gamma')\rangle \\
\gamma(\langle\gamma' : \Gamma'\rangle) & := \langle\gamma(\gamma') : \gamma(\Gamma')\rangle & \sigma(\langle\sigma' : \Gamma'\rangle) := \langle\sigma(\sigma') : \sigma(\Gamma')\rangle \\
\gamma(u.t) & := \gamma(u).\gamma^*(t) & \sigma(u.t) := \sigma(u).\sigma(t)
\end{array}
$$

Grammar As usual, the grammar characterizes each new type by three productions for type formation, introduction, and elimination, respectively. Moreover, we add the unusual production for evaluation, which will be inverse of reflection. As we will see below, elimination and evaluation are inter-definable, and we add both to the grammar to be flexible regarding which one of them is primitive.

For dependent functions, our notation closely corresponds to the usual one.

For dependent products, we use a modified notation. For example, in the binary case, we write $\langle x : A, y : B\rangle$ for the product of A and B. The tuple (a, b) is denoted by $\langle a, b : x : A, y : B\rangle$, i.e, we make the unusual choice to have tuples carry their type with them. This is rather awkward, but we will need it for technical reasons. In a way, this technicality is not so surprising because dependent product types do not in general permit unique type inference. The projections are written $u.x$ and $u.y$, i.e., we refer explicitly to the variable occurring in the type instead of using a number. If we switched to de-Bruijn indices, these projections would be $x.2$ and $x.1$, respectively, and we would recover the usual notation for projections except for the reversed numbering. Moreover, we

Reflecting ...	... terms	... substitutions
Congruence for introduction (ξ)	$\dfrac{\Gamma, \Gamma' \vdash_\Sigma t = t'}{\Gamma \vdash_\Sigma \lambda_{\Gamma'} t = \lambda_{\Gamma'} t'}$	$\dfrac{\Gamma \vdash_\Sigma \gamma = \gamma' : \Gamma' \to \Gamma}{\Gamma \vdash_\Sigma \langle \gamma : \Gamma' \rangle = \langle \gamma' : \Gamma' \rangle}$
Congruence for elimination	$\dfrac{\Gamma \vdash_\Sigma f = f' \quad \Gamma \vdash_\Sigma \gamma = \gamma' : \Gamma' \to \Gamma}{\Gamma \vdash_\Sigma f\,\gamma = f'\,\gamma'}$	$\dfrac{\Gamma \vdash_\Sigma u = u' \quad \Gamma \vdash_\Sigma t = t'}{\Gamma \vdash_\Sigma u.t = u'.t'}$
Computation (β) inter-derivable	$\dfrac{\Gamma, \Gamma' \vdash_\Sigma t : A \quad \Gamma \vdash_\Sigma \gamma : \Gamma' \to \cdot}{\Gamma \vdash_\Sigma (\lambda_{\Gamma'} t)\,\gamma = \gamma(t)}$	$\dfrac{\Gamma \vdash_\Sigma \gamma : \Gamma' \to \cdot \quad \Gamma, \Gamma' \vdash_\Sigma t : A}{\Gamma \vdash_\Sigma \langle \gamma : \Gamma' \rangle.t = \gamma(t)}$
Soundness	$\dfrac{\Gamma, \Gamma' \vdash_\Sigma t : A}{\Gamma \vdash_\Sigma \mathtt{ev}^{\Gamma'}(\lambda_{\Gamma'} t) = t}$	$\dfrac{\Gamma \vdash_\Sigma \gamma : \Gamma' \to \cdot}{\Gamma \vdash_\Sigma \mathtt{ev}^{\Gamma'}(\langle \gamma : \Gamma' \rangle) = \gamma : \Gamma' \to \Gamma}$
Completeness (η) inter-derivable	$\dfrac{\Gamma \vdash_\Sigma f : \Pi_{\Gamma'} A}{\Gamma \vdash_\Sigma \lambda_{\Gamma'} \mathtt{ev}^{\Gamma'}(f) = f}$	$\dfrac{\Gamma \vdash_\Sigma u : \langle \Gamma' \rangle}{\Gamma \vdash_\Sigma \langle \mathtt{ev}^{\Gamma'}(u) : \Gamma' \rangle = u}$
Extensionality	$\dfrac{\Gamma, \Gamma' \vdash_\Sigma \mathtt{ev}^{\Gamma'}(f) = \mathtt{ev}^{\Gamma'}(f')}{\Gamma \vdash_\Sigma f = f'}$	$\dfrac{\Gamma \vdash_\Sigma \mathtt{ev}^{\Gamma'}(u) = \mathtt{ev}^{\Gamma'}(u') : \Gamma' \to \Gamma}{\Gamma \vdash_\Sigma u = u'}$

Fig. 5. Reflection based on Contexts: Equality Rules

generalize the projection to arbitrary terms $u.t$; this more general form is easily definable from the projections for all variables. It has the advantage of leading to a more symmetric treatment of dependent functions and dependent products.

Finally, we introduce meta-variable names to make the notation more intuitive: Terms of the two types (i.e., functions and tuples) are by default written using the names f and u, respectively.

Our contexts permit declarations at arbitrary type universes. But it is common to restrict dependent functions and products to declarations at $\mathtt{type}^1$, i.e., where $x : A : \mathtt{type}^1$. This restriction can be achieved easily, but our treatment below works for the general case.

Type Formation and Introduction The type formation rules are straightforward.

The introduction rules carry out the reflection: If t is a term of type A in context Γ', then its reflection $\lambda_{\Gamma'} t$ populates the type $\Pi_{\Gamma'} A$. In particular, the special cases $n = 1$ and $n = 2$ correspond to the rules $(*, *)$ and $(\Box, \Box)$ in Barendregt's λ-cube [Bar92].

Similarly, for a substitution γ out of Γ', its reflection $\langle \gamma : \Gamma' \rangle$ populates the type $\langle \Gamma' \rangle$.

Elimination and Computation The intended meaning of these new types becomes apparent when considering the elimination and the computation rules together.

The behavior of a term with free variables is substitution. Consequently, a term f of type $\Pi_{\Gamma'} A$ operates on a substitution γ for its free variables. In the simply-typed case, this yields a term of type A; in the dependently-typed case

where the variables of Γ' may also be free in A, the resulting type is $\gamma(A)$. The computation rule defines this behavior: $(\lambda_{\Gamma'} t)\,\gamma$ simplifies to $\gamma(t)$. This is the usual rule of β-equality.

The behavior of a substitution out of Γ' is application to a term with free variables from Γ'. Consequently, a term u of type $\langle\Gamma'\rangle$ operates on a term t of type A in context Γ'. In particular, if x is a variable in Γ', then $u.x$ returns the term provided for x u (qua substitution).

In the simply-typed case, the projection $u.t$ yields a term of type A. In the dependently-typed case, the variables of Γ' that occur in A must be substituted with the corresponding projection of u, i.e., the variable x of Γ' is substituted with $u.x$. Since u is only the reflection of a substitution and not a substitution itself, we have to evaluate u into a substitution first. This is the purpose of $\mathsf{ev}^{\Gamma'}(u)$.

Evaluation and Soundness The natural inverse of reflection is evaluation, which transforms an expression into the meta-level object it reflects. Intuitively, evaluation inverts reflection, and we call the two equality rules expressing this inverseness *soundness* (i.e., evaluation inverts introduction) and *completeness* (i.e., introduction inverts evaluation). While soundness is intuitively necessary, completeness could potentially be omitted.

Theorem 1. *Both for functions and for products, evaluation and soundness can be defined and proved from elimination and computation, and vice versa.*

Proof. Given elimination and computation, we define evaluation as follows:

$$\mathsf{ev}^{\Gamma'}(f) := f\,(x_1, \ldots, x_n) \qquad \mathsf{ev}^{\Gamma'}(u) := u.x_1, \ldots, u.x_n$$

for $\Gamma' = x_1 : A_1, \ldots, x_n : A_n$. The soundness rules follow immediately from the respective computation rule.

Conversely, given evaluation and soundness, we define elimination as follows

$$f\,\gamma := \gamma(\mathsf{ev}^{\Gamma'}(f)) \qquad u.t := \mathsf{ev}^{\Gamma'}(u)(t)$$

where Γ' is as in the respective typing rule. The computation rules follow immediately from the respective soundness rule.

Congruence The congruence rules make sure that the new term formation rules respect equality, i.e., all operators produce equal terms from equal arguments. The congruence rule for the introduction form of the dependent function type is also known as the ξ rule.

Remark 3. Interestingly, not all accounts of typed λ-calculus assume ξ. It is well-known that ξ and η imply functional extensionality. Therefore, in the typical case where η is assumed, either ξ or η must be dropped if one wants to avoid extensionality. Often ξ is dropped, but we consider it the more natural rule due to its flavor of a congruence rule. Therefore, we will first assume ξ and later show (see Thm. 2) that η and extensionality are equivalent.

Completeness/Extensionality When introducing a new Church-style type, its inhabitants do not exist a-priori. Instead, we have to create new terms by introductory forms in order to populate the new type. Often we want to assume that (up to the equality judgment) all inhabitants can be obtained by applying an introductory form. This means that the new type does not have any spurious, unobservable inhabitants. This is not the case in general and two different kinds of rules can be used to specify this.

The first way – which we call "completeness" – has the flavor of inductive type definitions: It says that the new type is freely generated from the introductory forms. Another way to put this is that the introduction forms are surjective. Concretely, we give an axiom that makes introduction the inverse of evaluation so that every element can be written as an introductory form. For dependent functions, this yields the usual η-rule. For dependent products, it yields the equally common rule $(\pi_1(u), \dots, \pi_n(u)) = u$, albeit somewhat disguised by our notation.

The second way – which we call "extensionality" – has the flavor of coinductive type definitions: It says that the new type is cofreely cogenerated in the sense that any two terms with the same behavior are equal. Another way to put this is that the elimination forms are injective. Concretely, we give an axiom that makes two terms equal if their evaluations are. For dependent functions, this yields the usual extensionality rule: two functions are equal if they agree for all arguments. For dependent products, the resulting rule is sometimes called the characteristic property: In standard notation, it becomes $(u_1, \dots, u_n) = (u'_1, \dots, u'_n)$ iff $u_i = u'_i$ for $i = 1, \dots, n$.

These two ways are equivalent, i.e., completeness and extensionality rules are derivable from each other.

Theorem 2. *Both for function and for products, in the presence of the other rules, the freeness and extensionality rules can be derived from each other.*

Proof. The two derivations are as follows:

Symmetry We have systematically developed our rules in a way that maximizes the symmetry between functions and products. In particular, one's introduction form corresponds to the other's elimination form, and computation is substitution application, which combines the two reflected objects. Consequently, in a type theory with both of these types, we have can relate them to each other:

Theorem 3. *Consider the basic type theory with dependent functions and dependent products. Then the following rule is derivable:*

$$\frac{\Gamma \vdash_\Sigma f : \Pi_{\Gamma'} A \quad \Gamma \vdash_\Sigma u : \langle \Gamma' \rangle}{\Gamma \vdash_\Sigma f\, \mathtt{ev}^{\Gamma'}(u) = u.\mathtt{ev}^{\Gamma'}(f)}$$

Proof. Applying first η and then β to each expression, we see that both are equal to $\mathtt{ev}^{\Gamma'}(u)(\mathtt{ev}^{\Gamma'}(f))$.

Definition 6. *In the situation of Thm. 3, we write* $f * u$ *for* $\mathtt{ev}^{\Gamma'}(u)(\mathtt{ev}^{\Gamma'}(f))$.

Informally, an n-ary function is the same as a unary function whose domain is an n-ary product.

Example 1. Above we pointed out that we only have to reflected substitutions into the empty context. Indeed, we can define the type of substitution $\vdash_\Sigma \gamma{:}\Gamma{\to}\Delta$ as follows:

$$Subs(\Gamma, \Delta) := \langle \Delta \rangle \to \langle \Gamma \rangle$$

Here we use the usual notation $A \to B$ instead of $\Pi_{x:A} B$ if x does not occur in B. Note that the reflection is contravariant: Substitutions $\vdash_\Sigma \gamma : \Gamma \to \Delta$ are reflected as terms of type $\langle \Delta \rangle \to \langle \Gamma \rangle$.

Given a substitution $\vdash_\Sigma t_1, \dots, t_n : \Gamma \to \Delta$, we can reflect it as

$$\lambda_{u:\langle \Delta \rangle} \langle u.t_1, \dots, u.t_n : \Gamma \rangle.$$

Conversely, given a term g of type $Subs(\Gamma, \Delta)$, its evaluation is the substitution $t'_1, \dots, t'_n$ where the term t'_i in context Δ is given by $(g\, \langle id_\Delta : \Delta \rangle).x_i$ if $x_1, \dots, x_n$ are the variables of Γ. It is straightforward to show that these are mutually inverse (i.e., to show soundness and completeness).

Identity and composition of substitutions can also be defined easily:

$$\begin{array}{lcl} i & : & Subs(\Gamma, \Gamma) = \lambda_{u:\langle \Gamma \rangle} u \\ ; & : & \Pi_{g:Subs(\Gamma,\Delta), h:Subs(\Delta,\Theta)} Subs(\Gamma, \Theta) = \lambda_{g,h} \lambda_{u:\langle \Theta \rangle} h\, (g\, u) \\ @ & : & \Pi_{g:Subs(\Gamma,\Delta), a:\Pi_\Gamma \mathtt{type}^1, t:\Pi_\Gamma \mathtt{ev}^\Gamma(a)} \Pi_\Delta a * (g\, \langle id_\Delta : \Delta \rangle) \\ & = & \lambda_{g,a,t} \lambda_\Delta t * (g\, \langle id_\Delta : \Delta \rangle) \end{array}$$

Here we have omitted the inferable variable types. Of course, the type of @ is only well-formed in type theories where reflection of expressions in level-1 contexts is permitted.

4 Reflection based on Signatures

4.1 Signatures and Contexts

Our basic type theory treats signatures and contexts in a very analogous fashion. This raises the question why they are distinguished at all. Indeed, neither our grammar nor our rules introduce a fundamental distinction that would preclude conflating them. Therefore, we will briefly discuss the intuitions behind them.

Often signatures are more expressive than contexts. Signatures can contain additional declarations (e.g., definitions, theorems, notations), they can be subject to a module system, or contexts can be restricted to certain levels (e.g., level 0). But these differences are design questions for individual type theories, and do not stand against conflating them in principle.

Another distinction is that signatures are often larger (e.g., signatures often growing linearly and contexts logarithmically in the size of a project). Thus,

signatures are often stored as (unordered) sets of declarations with the declared names visible to the outside. Contexts, on the other hand, are often stored as lists in which declarations are primarily identified by their position. Consequently, shadowing is no problem in contexts but often avoided in signatures. But this difference is mainly an implementation issue that has little bearing on the formal definition.

The difference that we have identified as crucial for reflection is the following: The semantics of a signature is absolute whereas the semantics of a context is relative to a signature. More precisely, once a signature Σ is fixed, the terms over Σ are also fixed. Thus, Σ forms a closed world that is populated by the constants declared in it. Intuitively, the constants declared in Σ are objects that exist by virtue of being declared. But contexts are used within signatures, while the signature is built and thus is still changing. Therefore, the semantics of a context changes as the signature grows, and in particular, its semantics is not fixed yet when it is used.

For example, the type $\Pi_\Gamma A$ contains all expressions of type A that can be formed using the variables in Γ. These expressions may also use all constants of the containing signature no matter whether they are declared before or after this reference to the context Γ. For example, a later declaration $c : \Pi_\Gamma A$ introduces a new term of type A in context Γ. This is not the case for signatures: If $S = \{\Sigma\}$ is declared, the terms over it are fixed. A later declaration $c :\uparrow_S A$ (This is the notation we will use later on for the type of terms of type A over signature S.) can never introduce additional terms.

Thus, contexts are subject to an open-world assumption that implicitly includes all future declarations, and signatures represent closed worlds. When reflecting, this will make a big difference. Most importantly, induction is possible for terms of type $\uparrow_S A$ but not for terms of type $\Pi_\Gamma A$.

4.2 Reflecting Signatures

We will now extend our approach to reflect terms over a signature instead of over a context. Intuitively, we want to write $\uparrow_S A$ for the type reflecting the closed terms over the signature S and $\ulcorner t \urcorner_S$ for the reflection of a term $\vdash_S t : A$. For example, with a signature

$$S = n : \texttt{type},\ z : n,\ s : n \to n$$

the type $\uparrow_S n$ yields the type of natural numbers. More generally, we see that this kind of reflection yields abstract data types.

Consequently, the elimination form of this new type should express induction. It is well-known that abstract data types can be seen as initial in a canonical category, in which case their elimination form can be stated in terms of the universal morphism out of the initial object (then often called a *catamorphism*). Therefore, and in correspondence to context-based reflection, we conjecture that the elimination form of our reflected signatures will be given by morphisms out of S.

Moreover, morphisms σ out of S are also interesting in themselves for two reasons: If S is seen as a record type, σ is a record value of type S. And if S is seen as a logical theory or a software specification, then morphisms out of it become models and implementations, respectively. Therefore, we introduce a second type $\downarrow S$ with elements $\ulcorner \sigma : S \urcorner$ reflecting morphisms out of S.

We can proceed in very much the same way as for context-based reflection. But before doing so, we discuss a few differences between context- and signature-based reflection.

Signature Expressions As discussed above, signatures correspond to closed worlds, and we are interested in reflecting *previously fixed* signatures. This has two consequences.

Firstly, we need a way to fix a signature. Therefore, we introduce signature declarations using the production

$$\Sigma \ ::= \ \Sigma, S = \{\Sigma\}$$

This is useful anyway because signature-based reflection often requires referring to the same (possibly big) signature multiple times. Therefore, it is convenient to introduced named signatures and refer to a signature by name. More generally, signatures may be structured using a module system, which requires a substantially more complex language to build the signatures S.

Secondly, we need of notion of signature expressions, which define the signatures that are available for reflection. In the easiest case, the valid signature expressions are exactly the names S of previously declared signatures. We will leave it at that and not formally define the valid signature expressions at this point.

Union Signatures We write $\Sigma + S$ for the union of Σ and S. Given a module system for signatures, this union is equivalent to $\Sigma, \texttt{include}\, S$.

Then the rules for reflection based on signatures are given in Fig. 6 and Fig. 7.

4.3 Limitations

The reflection on signatures looks very elegant, but upon closer inspection, we find a few weaknesses.

Top-down Inductive Functions Not all inductive functions can be represented as morphisms. Inspecting the definition of the homomorphic extension of a morphism, we see that the resulting function is always bottom-up: The application to a composed expression is computed in terms of the application to the components. Thus, we cannot represent a function that inspects a component. Examples of such top-down functions are selector functions, which return a specific component without a recursive call.

Reflecting ...	... terms	... morphisms
Grammar	$E ::= \uparrow_S A \mid \ulcorner t \urcorner_S \mid q\,\sigma \mid \llcorner q \lrcorner_S$	$E ::= \downarrow S \mid \ulcorner \sigma : S \urcorner \mid m.t \mid \llcorner m \lrcorner_S$
Meta-variable for inhabitants	q	m
Type formation	$\dfrac{\vdash_S A : \mathtt{type}}{\vdash_\Sigma \uparrow_S A : \mathtt{type}}$	$\dfrac{\vdash_\Sigma S\ \mathtt{Sig}}{\vdash_\Sigma \downarrow S : \mathtt{type}}$
Introduction (reflection)	$\dfrac{\vdash_\Sigma \Gamma\ \mathtt{Ctx} \qquad \vdash_S A : \mathtt{type} \qquad \Gamma \vdash_{\Sigma+S} t : A}{\Gamma \vdash_\Sigma \ulcorner t \urcorner_S : \uparrow_S A}$	$\dfrac{\Gamma \vdash \sigma : S \to \Sigma}{\Gamma \vdash_\Sigma \ulcorner \sigma : S \urcorner : \downarrow S}$
Elimination (induction) inter-definable	$\dfrac{\Gamma \vdash_\Sigma q : \uparrow_S A \qquad \Gamma \vdash \sigma : S \to \Sigma}{\Gamma \vdash_\Sigma q\,\sigma : \sigma(A)}$	$\dfrac{\Gamma \vdash_\Sigma m : \downarrow S \qquad \vdash_S t : A}{\Gamma \vdash_\Sigma m.t : \llcorner m \lrcorner_S(A)}$
Evaluation	$\dfrac{\Gamma \vdash_\Sigma q : \uparrow_S A}{\Gamma \vdash_{\Sigma+S} \llcorner q \lrcorner_S : A}$	$\dfrac{\Gamma \vdash_\Sigma m : \downarrow S}{\Gamma \vdash \llcorner m \lrcorner_S : S \to \Sigma}$

Fig. 6. Reflection based on Signatures

Dependent Inductive Functions The inductive functions that we can represent using morphisms are always simply-typed. Even though dependent functions are less important than simple ones, it is disconcerting to exclude a priori.

Inductive Proof Principle While morphisms yield an inductive definition principle, the theory lacks an inductive proof principle. This is not surprising because in a Curry-Howard-style representation, such a proof corresponds to a dependent function from the inductive type to the respective type of proofs.

Henkin Models A well-known result in higher-order logic states that standard models, which interpret the type of functions compositionally, are not complete for signatures containing higher-order functions. Cast in our terms, this means that morphism are standard models, and higher-order term models cannot be represented as morphisms.

In order to remedy these short-comings, we introduce a generalized notion of a morphism.

5 Extensional Morphisms

5.1 Extensional Morphisms

Notation 2. We will write substitution application as $t[\gamma]$ and composition $\gamma \circ \gamma'$ as $\gamma'[\gamma]$ from now on. In particular, $t[\gamma'[\gamma]] = t[\gamma'][\gamma]$.

Definition 7. *For a signature Σ, we write $\mathtt{Cons}(\Sigma)$ for the category of contexts and substitutions over Σ.*

Reflecting ...	... terms	... morphisms
Congruence for introduction	$\dfrac{\Gamma \vdash_{\Sigma+S} t = t'}{\Gamma \vdash_{\Sigma} \ulcorner t \urcorner_S = \ulcorner t' \urcorner_S}$	$\dfrac{\Gamma \vdash \sigma = \sigma' : S \to \Sigma}{\Gamma \vdash_{\Sigma} \ulcorner \sigma : S \urcorner = \ulcorner \sigma' : S \urcorner}$
Congruence for elimination	$\dfrac{\Gamma \vdash_{\Sigma} q = q' \qquad \Gamma \vdash \sigma = \sigma' : S \to \Sigma}{\Gamma \vdash_{\Sigma} q\,\sigma = q'\,\sigma'}$	$\dfrac{\Gamma \vdash_{\Sigma} m = m' \qquad \vdash_S t = t'}{\Gamma \vdash_{\Sigma} m.t = m'.t'}$
Computation inter-derivable	$\dfrac{* \quad \Gamma \vdash_{\Sigma+S} t : A \qquad \Gamma \vdash \sigma : S \to \Sigma}{\Gamma \vdash_{\Sigma} (\ulcorner t \urcorner_S)\,\sigma = \sigma(t)}$	$\dfrac{\Gamma \vdash \sigma : S \to \Sigma \qquad \vdash_S t : A}{\vdash_{\Sigma} \ulcorner \sigma : S \urcorner.t = \sigma(t)}$
Soundness	$\dfrac{* \quad \Gamma \vdash_{\Sigma+S} t : A}{\Gamma \vdash_{\Sigma+S} \llcorner \ulcorner t \urcorner_S \lrcorner_S = t}$	$\dfrac{\Gamma \vdash \sigma : S \to \Sigma}{\Gamma \vdash \llcorner \ulcorner \sigma : S \urcorner \lrcorner_S = \sigma : S \to \Sigma}$
Completeness inter-derivable	$\dfrac{\Gamma \vdash_{\Sigma} q \colon \uparrow_S A}{\Gamma \vdash_{\Sigma} \ulcorner \llcorner q \lrcorner_S \urcorner_S = q}$	$\dfrac{\Gamma \vdash_{\Sigma} m \colon \downarrow S}{\Gamma \vdash_{\Sigma} \ulcorner \llcorner m \lrcorner_S : S' \urcorner = m}$
Extensionality	$\dfrac{\Gamma \vdash_{\Sigma+S} \llcorner q \lrcorner_S = \llcorner q' \lrcorner_S}{\Gamma \vdash_{\Sigma} q = q'}$	$\dfrac{\Gamma \vdash \llcorner m \lrcorner_S = \llcorner m' \lrcorner_S : S \to \Sigma}{\Gamma \vdash_{\Sigma} m = m'}$

Fig. 7. Reflection based on Signatures: Equality Rules

* abbreviates $\vdash_{\Sigma} \Gamma$ `Ctx` and $\vdash_S A$: `type`

Definition 8. *A* extensional morphism*, short* e-morphism *from a signature Σ to a signature Σ' is a functor Φ : $\mathtt{Cons}(\Sigma) \to \mathtt{Cons}(\Sigma')$, 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 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 9. *An* intensional morphism*, short* i-morphism *is a morphism in the sense of the previous sections.*

The intuition is that intensional morphisms satisfy the required properties by construction whereas extensional morphisms are not necessarily finitary objects.

Theorem 4 (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:

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

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 so that we only have to proof its 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.

So far, we have defined e-morphism as maps contexts, substitutions, types, and terms. But we can also extend them to context and substitution fragments, kinds, and type families:

Definition 10. *Consider an e-morphism $\Phi : \Sigma \to \Sigma'$.*

– *For $\Gamma \vdash_\Sigma \Gamma'\ \texttt{Ctx}$, 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 kinds, we define*

$$\begin{aligned} \mathtt{type}^{\Phi_\Gamma} &= \mathtt{type} \\ (\Pi_{\Gamma'} K)^{\Phi_\Gamma} &= \Pi_{\Gamma'^{\Phi_\Gamma}} K^{\Phi_{\Gamma,\Gamma'}} \end{aligned}$$

– *For (proper) type families* $\Gamma \vdash_\Sigma C : \Pi_{\Gamma'} K$*, we define*

$$\begin{aligned} (\lambda_{\Gamma'} A)^{\Phi_\Gamma} &= \lambda_{\Gamma'^{\Phi_\Gamma}} A^{\Phi_{\Gamma,\Gamma'}} \\ (A\,\gamma)^{\Phi_\Gamma} &= \lambda_{\Gamma'} A\,(\gamma, id_{\Gamma'}^{\Phi_\Gamma}) \end{aligned}$$

Note that we perform η-expansion in the case for $A\,\gamma$ so that the definition eventually recurses into a base 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 can sum up the properties of e-morphisms as:

Theorem 5. *An e-morphism* $\Sigma \to \Sigma'$ *preserves all* Σ*-judgments.*

5.2 Concrete E-Morphisms

We will now construct some e-morphisms.

Theorem 6. *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 11. *Consider a list of e-morphisms* $\Phi^i : \Sigma \to \Sigma'$ *for* $i = 1, \ldots, n$*. By mutual induction, we define an e-morphism* $\Phi : \mathtt{Cons}(\Sigma) \to \mathtt{Cons}(\Sigma')$ *and substitutions* $\varphi^i_\Gamma : \Gamma^{\Phi^i} \to \Gamma^\Phi$*:*

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

where we used the numbers i *as fresh variables names for simplicity.*

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

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 7. *In the situation of Def. 11, 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

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

and

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

from the completeness rule for products.

Note that Def.11 includes the case $n = 0$, which yield a unit e-morphism: It maps every type to $\langle\cdot\rangle$ and every term to $\langle\cdot : \cdot\rangle$.

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

Definition 12. *Consider an e-morphism $\Phi : \Sigma \to \Sigma'$ and a context $\vdash_{\Sigma'} U$ $\mathtt{Ctx}$. By mutual induction, we define a functor $\Phi^U : \mathtt{Cons}(\Sigma) \to \mathtt{Cons}(\Sigma')$ 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 8. *In the situation of Def. 12, 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 completeness rule for functions.

Finally, we show that result that motivated the very definition, namely that quotation is an e-morphism (but not an i-morphism):

Definition 13. *Given a signature S that can be reflected in Σ, we define an e-morphism $Q : S \to \Sigma$. For every term t, t^Q is the term arising from $\ulcorner t \urcorner_S$ by replacing every free variable x of t with $\llcorner x \lrcorner_S$. Similarly for every type A, A^Q is the type arising from $\uparrow_S A$ by replacing every free variable x of A with $\llcorner x \lrcorner_S$.*

Theorem 9. *In the situation of Def. 13, Q is an e-morphism.*

Proof. The preservation of typing variables is obvious and the preservation of substitution straightforward. The preservation of variables follows from $x^{\Phi_\Gamma} = \ulcorner\llcorner x \lrcorner_S \urcorner_S$ and the completeness rule for reflection.

Note that the Thms. 6, 7, 8, and 9 follow a pattern: Each reflection operation is connected to one construction of e-morphisms. Moreover, in the latter three cases, the respective completeness rule is needed to prove the preservation of variables.

5.3 Context-Based Reflection under an E-Morphism

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.

For our additional operators, which arise by reflecting objects that interact with e-morphisms, we have reason to expect some interesting relationship. Moreover, based on general intuition, we expect this relationship to be relatively strong for dependent products and more subtle for dependent functions. This is indeed the case.

Dependent Products We obtain a canonical isomorphism between $\langle\Gamma\rangle^\Phi$ and $\langle\Gamma^\Phi\rangle$. Precisely:

Definition 14. *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 10. *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. Since e-morphisms are functors, they preserve isomorphisms so that $\Gamma'^{\Phi_\Gamma} \overset{!}{\cong} (x : \langle\Gamma'\rangle)^{\Phi_\Gamma}$. Similarly, we have $\Gamma'^{\Phi_\Gamma} \overset{!}{\cong} x : \langle\Gamma'^{\Phi_\Gamma}\rangle$. Then the canonical isomorphism is obtained by composition.

Dependent Functions We do not have an isomorphism as for products. Instead, the type $(\Pi_\Gamma A)^\Phi$ is canonically embedded into the type $\Pi_{\Gamma^\Phi} A^{\Phi_\Gamma}$. Precisely:

Theorem 11. *For every e-morphism $\Phi : \Sigma \to \Sigma'$ and every $\Gamma, \Gamma' \vdash_\Sigma A$: `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. x^* is given by

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

which is obtained by applying Φ to the judgment

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

Note that here, when forming X^{Φ_Δ}, we have dropped those declaration from Δ that do not occur in X; this is possible because Φ commutes with substitutions.

Notation 3. In the situation of Thm. 11, 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 $f^{*\Phi}$.

With this notation, we can show the second part of the properties of x^*:

Theorem 12. *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 $id_{\Gamma^\Phi}, f^{\Phi_\Gamma}$.

Example 2. For the e-morphisms from Sect. 5.2, 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

$$f^{*\Phi} = \lambda_{\Gamma'^{\Phi_\Gamma}}\langle \ldots, f.i^{*\Phi_i}\,(x_1.i, \ldots, x_n.i), \ldots : \Gamma'^{\Phi_\Gamma}\rangle$$

In particular, in the special case where $\Gamma = \cdot$, $\Gamma' = x : A$, and $m = 2$, this yields

$$f^{*\Phi} = \lambda_{x:A^\Phi}\langle f.1^*\,x.1,\ f.2^*\,x.2 : 1 : A^{\Phi^1}, 2 : A^{\Phi^2}\rangle$$

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

$$f^{*\Phi} = \lambda_{x:A^\Phi}\lambda_{p:P}\big((f\,p)^*\,(x\,p)\big)$$

If Φ is quotation, then $f^* = \lambda_{\Gamma'^{\Phi_\Gamma}}\ulcorner\llcorner f \lrcorner_S(\llcorner x_1 \lrcorner_S, \ldots, \llcorner x_n \lrcorner_S)\urcorner_S$.

5.4 Logical Relations on E-Morphisms

In [RS12], we introduced the notion of a logical relation on an i-morphism. We can now generalize this definition to e-morphisms.

Definition 15. *Given an e-morphism $\Phi : \Sigma \to \Sigma'$. A logical relation ρ on Φ is*

Remark 4. The definition given here is that of a unary logical relation. The result in [RS12] is stated for the n-ary case, and our definition also easily extends to n-ary logical relations (between n e-morphisms). However, because e-morphisms are closed under products, our definition presents no loss of generality: n-ary logical relations can be recovered as unary ones on the product of n e-morhisms.

Logical relations yield another source of e-morphisms:

Definition 16. *Consider an e-morphism Ψ and a logical relation ρ on it. By mutual induction we define an e-morphism Φ and substitutions $\varphi_\Gamma : \Gamma^\Phi \to \Gamma^\Psi$*

$$t^{\Phi_\Gamma} = \langle t^{\Psi_\Gamma}, \rho(t) : \rangle$$

$$A^{\Phi_\Gamma} = \langle \_ : A^{\Psi_\Gamma},\ ! : \rho(A)\ \_ \rangle$$

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

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

Theorem 13. *In the situation of Def. 16, $\Phi|\rho$ is an e-morphism.*

Proof.

5.5 Connections

Since e-morphisms are functors, it is straightforward to define morphisms between them, which we will call *connections*. We will not use the obvious candidate definition and instead go with its dual (see also Sect. 5.7):

Definition 17. *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 $\vdash_{\Sigma'} \varphi_\Gamma : \Gamma^{\Phi'} \to \Gamma^\Phi$.

Theorem 14 (Characterization of Connections). *Assume two e-morphisms Φ, Φ' and a family of substitutions φ_Γ from $\Phi'(\Gamma)$ to $\Phi(\Gamma')$. Then the following are equivalent:*

1. *φ is a connection $\Phi \Rightarrow \Phi'$.*
2. *φ preserves variables, typing, and substitution in the following sense:*
   (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 φ is uniquely determined by the terms φ^A_Γ and vice versa. Moreover, note that the three conditions are analogous to those in Thm. 4.

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 $(\Gamma', x : A[\gamma])^{\Phi}$).

5.6 Type Theory as a 2-Category

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

Theorem 15. *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.

Definition 18. *For a signature Σ and a Σ-context Γ, 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 16. *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.

Theorem 17. *The e-morphisms and connections form a category.*

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

Theorem 18. *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. 11.

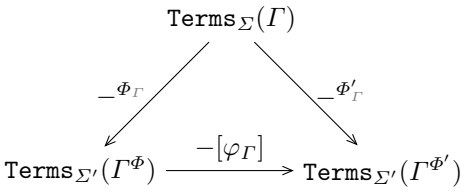
Note that the signatures and i-morphisms form a broad subcategory, but the product of i-morphisms is usually not an i-morphism.

Theorem 19. *The signatures, e-morphisms, and connections form a 2-category.*

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

5.7 Connections as Model Morphisms

If we thought of e-morphisms and connections simply as term-translation operations, it would be intuitive to define connections from Φ to Φ' as natural transformations $\varphi : \Phi \Rightarrow \Phi'$. We would obtain a triangle



that we would expect to commute. But in fact, that is not interesting. Since Φ_Γ and Φ'_Γ preserve variables, φ_Γ would have to, too. But that is only possible if Φ and Φ' agree on types, which is unreasonably restrictive.

Instead, we think of e-morphisms as models or interpretation functions. Then for every e-morphism Φ, we consider the type A^{Φ} as the universe that Φ assigns to A. We consider closed terms $u : A^{\Phi}$ as values of that universe and a substitution $\vec{u}$ from Γ^{Φ} into the closed context as a tuple of values. Moreover, if A has free variables, then $A^{\Phi_\Gamma}[\vec{u}]$ is a family of universes indexed by tuples $\vec{u} : \Gamma^{\Phi} \to \cdot$.

Consequently, we should think of a connection φ as a model morphism. Then, for every closed type Σ-type A, we should expect a Σ'-function $A^{\Phi} \to A^{\Phi'}$ – but that is exactly a substitution $(x : A)^{\Phi'} \to (x : A)^{\Phi}$, and that motivated our definition of connection. More generally, if A has free variables, we expect a $\vec{u}$-indexed family of maps from $A^{\Phi_\Gamma}[\vec{u}]$ to the corresponding Φ'-universe $A^{\Phi'_\Gamma}[\varphi_\Gamma[\vec{u}]]$.

The following definitions and theorem confirm this perspective:

Definition 19. *Given a signature Σ and a signature F. An F-*model *M of Σ is an e-morphism $\Sigma \to F$.*

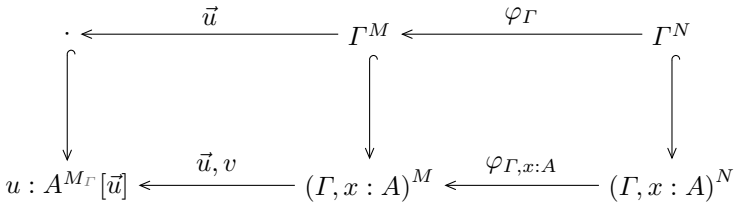
For every $\Gamma \vdash_F A : \mathtt{type}$ and $\vdash_F \vec{u} : \Gamma^M \to \cdot$, we call $A^M[\vec{u}]$ the universe *of A in M at $\vec{u}$.*

Definition 20. *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 $\vec{u}$.*

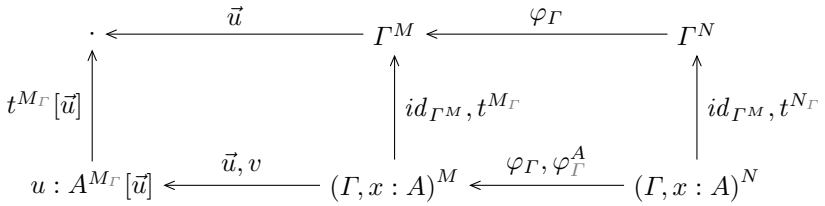


Theorem 20. *In the situation of Def. 20, for every term $\Gamma \vdash_\Sigma t : A$ and every $\vdash_F \vec{u} : \Gamma^\Phi \to \cdot$*

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

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. 14.



In particular, if we restrict attention to the case where A is closed, this specializes to the well-known condition on model morphisms:

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

5.8 Initial E-Morphisms

Based on the intuition of e-morphisms and connections as models and model morphisms, an important question is whether there are initial models. The obvious candidate for the initial e-morphism is Q. However, this is not quite so easy. We will first look at the special case of a first-order signature.

Definition 21. *A type is called* first-order *if it does not take functions as arguments. A signature or context is called* first-order *if no symbol or variable declared in it takes functions as arguments.*

Definition 22. *We define $I : S \to \Sigma$ as the i-morphism, which maps every S-symbol to $Q(a)$.*

Q and I do not agree, but we have:

Theorem 21. *Q and I agree on all first-order types and terms.*

Proof. Clear because Q and morphism application only differ in the cases for higher-order constructors.

Theorem 22. *If S is first-order, the category of i-morphisms from S to Σ has an initial object.*

Proof. The initial object is the i-morphism I. Given an i-morphism $\sigma : S \to \Sigma$, the universal connection $i : I \Rightarrow \sigma$ is given by $i_\Gamma = \ldots, x_i\,\sigma, \ldots$ for every context $\Gamma = \ldots, x_i : A_i, \ldots$. We have $\vdash_\Sigma i_\Gamma : \Gamma^\sigma \to \Gamma^I$ because $\Gamma^I = \ldots, x_i :\uparrow_S A_i, \ldots$. The naturality of i can be verified by direct computation.

E-Morphisms can be reflected in the same way as i-morphisms. Let us assume that the type theory from Sect. 4 is modified in such a way that all judgments about i-morphisms are replaced with judgments about e-morphisms. Then all results stay valid

Theorem 23. *The category of e-morphisms and connections has an initial object.*

Proof. The initial object is given by quotation as constructed in Def. 13. The universal connection φ into an e-morphism Φ is given by

6 Conclusion

References

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

RS12. F. Rabe and K. Sojakova. Logical Relations in Twelf. see `http://kwarc.info/frabe/Research/RS_logrels_12.pdf`, 2012.