Proofs for Free in the λΠ-Calculus Modulo Theory

Thomas Traversié

Proofs for Free in the $\lambda\Pi$-Calculus Modulo Theory

Thomas Traversié
Université Paris-Saclay, CentraleSupélec, MICS
Gif-sur-Yvette, France
Université Paris-Saclay, Inria, CNRS, ENS-Paris-Saclay, LMF
Gif-sur-Yvette, France
thomas.traversie@centralesupelec.fr

Parametricity allows the transfer of proofs between different implementations of the same data structure. The $\lambda\Pi$-calculus modulo theory is an extension of the λ-calculus with dependent types and user-defined rewrite rules. It is a logical framework, used to exchange proofs between different proof systems. We define an interpretation of theories of the $\lambda\Pi$-calculus modulo theory, inspired by parametricity. Such an interpretation allows to transfer proofs for free between theories that feature the notions of proposition and proof, when the source theory can be embedded into the target theory.

1 Introduction

Many proof assistants have been developed during the past decades, such as AGDA, COQ, HOL LIGHT, ISABELLE, LEAN or MIZAR. All those systems have their own theoretical foundations and proof language. If a library of proofs has been formalized in some proof assistant, one would ideally like to export it automatically to any other proof assistant. That is why the question of the *interoperability* between proof systems arises. Exchanging formal proofs between different proof systems strengthen reusability, re-checking and preservation of libraries. For this purpose, Cousineau and Dowek developed the $\lambda\Pi$-calculus modulo theory [8], that combines λ-calculus with dependent types and user-defined rewrite rules. It is a logical framework, in which theories are defined by typed constants and rewrite rules, specified by the users. Many theories can be expressed in the $\lambda\Pi$-calculus modulo theory [4], such as Predicate Logic, Simple Type Theory and the Calculus of Constructions. Most of all, theories from various proof assistants can be expressed in this logical framework. As a consequence, it can be used as a common framework for exchanging proofs between proof systems [17]. The $\lambda\Pi$-calculus modulo theory has been implemented in the concrete language DEDUKTI [1, 14] and in the LAMBDAPI proof assistant, which features user-friendly proof tactics.

The problem of the exchange of proofs also emerges when it comes to the different implementations of a same data structure. One would like to share the theorems proved for one implementation to all the other implementations of the same data structure, without additional efforts. One method to derive theorems for free is to use *parametricity*. Reynolds [16] originally introduced an abstraction theorem, stating that the different implementations of a polymorphic function behave similarly. Wadler [18] used this result to derive properties satisfied by polymorphic functions, depending on their types. In other words, all functions of the same abstract type satisfy the same theorems. Bernardy *et al.* [2, 3] later extended parametricity to Pure Type Systems. Keller and Lasson [15] investigated parametricity for the Calculus of Inductive Constructions, the language behind the COQ proof assistant. More recently, Cohen *et al.* [7] developed a parametricity framework and implemented TROCQ, a COQ plugin for proof transfer based on parametricity. The exchange of proofs—the very purpose of the $\lambda\Pi$-calculus modulo theory—is therefore an important application of the parametricity translations.

Rabe, Sacerdoti Coen (Eds): LFMTP 2024
EPTCS 404, 2024, pp. 49–63, doi:10.4204/EPTCS.404.4

Transferring databases of proofs is relevant when working with related mathematical structures. For instance, if we have proved theorems in a theory of natural numbers and we want to use them in a theory of integers, we would like to export the proofs for non-negative integers. The same issue arises concerning various mathematical structures and databases of proofs, as we can *embed* natural numbers into reals, reals into reals extended with infinity elements, or sets into pointed graphs [5]. It would therefore be interesting to exchange proofs between theories of the λΠ-calculus modulo theory, when the source theory can be embedded into the target theory.

Contribution. In this paper, we define an interpretation of theories of the λΠ-calculus modulo theory, when they feature a prelude encoding of the notions of proposition and proof. Such an interpretation, inspired by parametricity, applies when we can embed the source theory $\mathbb{S}$ into the target theory $\mathbb{T}$. The interpretation depends on parameters, given by the user for representing each constant of the source theory by a term in the target theory. We provide the parameters necessary for interpreting the prelude encoding. We show that if $\mathbb{S}$ has an interpretation in $\mathbb{T}$, then the proofs written inside $\mathbb{S}$ can be transformed into proofs written inside $\mathbb{T}$. This interpretation comes with a relative consistency theorem: if $\mathbb{T}$ is consistent, then $\mathbb{S}$ is consistent too.

In order to illustrate this interpretation, we embed a theory of natural numbers into a theory of integers. This example, as well as the parameters for the prelude encoding, are given in DEDUKTI, and are available at `https://github.com/thomastraversie/InterpDK`.

Outline of the paper. In Section 2, we give a formal presentation of the λΠ-calculus modulo theory, and we detail a prelude encoding of the notions of proposition and proof. In Section 3, we define an interpretation of theories of the λΠ-calculus modulo theory. In particular, we specify the parameters required for interpreting the prelude encoding. We prove the interpretation theorem and the relative consistency theorem. At the end, we show how this interpretation can be used to derive theorems for free, taking the running example of natural numbers and integers.

2 Theories in the λΠ-Calculus Modulo Theory

In this section, we give a formal definition of the syntax and type system of the λΠ-calculus modulo theory. We present a standard way of expressing the notions of proposition and proof in it—called prelude encoding—and we emphasize the theories that will be considered in the rest of the paper.

2.1 The λΠ-Calculus Modulo Theory

The Edinburgh Logical Framework [13], also known as λΠ-calculus, is an extension of simply typed λ-calculus with dependent types. The λΠ-calculus modulo theory [8] is an extension of the Edinburgh Logical Framework, in which user-defined rewrite rules [9] have been added. Its syntax is given by:

Sorts	$s ::= \texttt{TYPE} \mid \texttt{KIND}$
Terms	$t,u,A,B ::= c \mid x \mid s \mid \Pi(x:A).\,B \mid \lambda(x:A).\,t \mid t\ u$
Contexts	$\Gamma ::= \langle\rangle \mid \Gamma, x:A$
Signatures	$\Sigma ::= \langle\rangle \mid \Sigma, c:A \mid \Sigma, \ell \hookrightarrow r$

where c is a constant and x is a variable (ranging over disjoint sets), $\Pi(x:A).\,B$ is a dependent product (simply written $A \to B$ if x does not occur in B), $\lambda(x:A).\,t$ is an abstraction, and $t\ u$ is an application. For

convenience, $\lambda(x_1:A_1).\ldots.\lambda(x_n:A_n).\,t$ is written $\lambda(x_1:A_1)\ldots(x_n:A_n).\,t$ and $\Pi(x_1:A_1).\ldots.\Pi(x_n:A_n).\,B$ is written $\Pi(x_1:A_1)\ldots(x_n:A_n).\,B$. Terms of type `TYPE` are called types, and terms of type `KIND` are called kinds. Signatures and contexts are finite sequences, and are written $\langle\rangle$ when empty. The $\lambda\Pi$-calculus modulo theory is a logical framework, in which Σ is fixed by the users depending on the theory they are working in. Signatures are composed of typed constants $c:A$ (such that A is a closed term, that is a term with no free variables) and rewrite rules $\ell \hookrightarrow r$ (such that the head-symbol of ℓ is a constant). The relation $\hookrightarrow_{\beta\Sigma}$ is the smallest relation, closed by context, such that if t rewrites to u for some rule in Σ or by β-reduction, then $t \hookrightarrow_{\beta\Sigma} u$. The conversion $\equiv_{\beta\Sigma}$ is the reflexive, symmetric, and transitive closure of the relation $\hookrightarrow_{\beta\Sigma}$.

$$\frac{}{\vdash\langle\rangle}\ [\textsc{Empty}] \qquad \frac{\vdash\Gamma \qquad \Gamma\vdash A:\texttt{TYPE}}{\vdash\Gamma,x:A}\ [\textsc{Decl}]\ x\notin\Gamma \qquad \frac{\vdash\Gamma}{\Gamma\vdash\texttt{TYPE}:\texttt{KIND}}\ [\textsc{Sort}]$$

$$\frac{\vdash\Gamma \qquad \vdash A:s}{\Gamma\vdash c:A}\ [\textsc{Const}]\ c:A\in\Sigma \qquad \frac{\vdash\Gamma}{\Gamma\vdash x:A}\ [\textsc{Var}]\ x:A\in\Gamma$$

$$\frac{\Gamma\vdash A:\texttt{TYPE} \qquad \Gamma,x:A\vdash B:s}{\Gamma\vdash\Pi(x:A).\,B:s}\ [\textsc{Prod}] \qquad \frac{\Gamma\vdash A:\texttt{TYPE} \qquad \Gamma,x:A\vdash B:s \qquad \Gamma,x:A\vdash t:B}{\Gamma\vdash\lambda(x:A).\,t:\Pi(x:A).\,B}\ [\textsc{Abs}]$$

$$\frac{\Gamma\vdash t:\Pi(x:A).\,B \qquad \Gamma\vdash u:A}{\Gamma\vdash t\ u:B[x\leftarrow u]}\ [\textsc{App}] \qquad \frac{\Gamma\vdash t:A \qquad \vdash B:s}{\Gamma\vdash t:B}\ [\textsc{Conv}]\ A\equiv_{\beta\Sigma}B$$

Figure 1: Typing rules of the $\lambda\Pi$-calculus modulo theory.

The judgment $\vdash\Gamma$ means that the context Γ is well-formed, and $\Gamma\vdash t:A$ means that t is of type A in the context Γ. When the context is empty, we simply write $\vdash t:A$. The typing rules for the $\lambda\Pi$-calculus modulo theory are given in Figure 1. The standard weakening rule is admissible.

A signature is a theory when its rewrite rules satisfy certain properties. We write Λ_Σ for the set of terms whose constants belong to Σ.

Definition 1 (Theory)**.** *A theory $\mathbb{T}$ in the $\lambda\Pi$-calculus modulo theory is given by a signature Σ such that:*

1. *for each rule $\ell\hookrightarrow r\in\Sigma$, we have ℓ and r in Λ_Σ,*
2. *$\hookrightarrow_{\beta\Sigma}$ is confluent on Λ_Σ,*
3. *for each rule $\ell\hookrightarrow r\in\Sigma$, for all context Γ, term $A\in\Lambda_\Sigma$ and substitution θ, if $\Gamma\vdash\ell\theta:A$ then $\Gamma\vdash r\theta:A$.*

Lemma 1. *If $\Gamma\vdash t:A$, then either $A=\texttt{KIND}$ or $\Gamma\vdash A:s$ for $s=\texttt{TYPE}$ or $s=\texttt{KIND}$. If $\Gamma\vdash\Pi(x:A).\,B:s$, then $\Gamma\vdash A:\texttt{TYPE}$.*

2.2 A Prelude Encoding

It is possible to formalize the notions of proposition and proof in the $\lambda\Pi$-calculus modulo theory [4]. In particular, this encoding—called prelude encoding—gives the possibility to quantify over certain propo-

sitions through codes, which is not possible inside the standard λΠ-calculus modulo theory. This encoding is defined by the following signature, written Σ_{pre}.

$$
\begin{array}{ll}
Set : \texttt{TYPE} & o : Set \\
El : Set \to \texttt{TYPE} & Prf : El\ o \to \texttt{TYPE} \\
\rightsquigarrow_d : \Pi(x : Set).\ (El\ x \to Set) \to Set & \Rightarrow_d : \Pi(x : El\ o).\ (Prf\ x \to El\ o) \to El\ o \\
El\ (x \rightsquigarrow_d y) \hookrightarrow \Pi(z : El\ x).\ El\ (y\ z) & Prf\ (x \Rightarrow_d y) \hookrightarrow \Pi(z : Prf\ x).\ Prf\ (y\ z) \\
\pi : \Pi(x : El\ o).\ (Prf\ x \to Set) \to Set & \forall : \Pi(x : Set).\ (El\ x \to El\ o) \to El\ o \\
El\ (\pi\ x\ y) \hookrightarrow \Pi(z : Prf\ x).\ El\ (y\ z) & Prf\ (\forall\ x\ y) \hookrightarrow \Pi(z : El\ x).\ Prf\ (y\ z)
\end{array}
$$

We declare the constant *Set*, which represents the universe of sorts, along with the injection *El* that maps terms of type *Set* to the type of its elements. We define a sort *o*, such that *El o* corresponds to the universe of propositions. The injection *Prf* maps propositions to the type of its proof. In other words, a term *P* of type *El o* is a proposition, and a term of type *Prf P* is a proof of *P*. The infix symbol $\rightsquigarrow_d$ (respectively $\Rightarrow_d$) is used to represent dependent function types between terms of type *Set* (respectively *El o*). Remark that the symbols $\rightsquigarrow_d$ and $\Rightarrow_d$ are generalizations of the usual functionality $\rightsquigarrow$ and implication $\Rightarrow$ in the case of dependent types. The symbol π (respectively $\forall$) is used to represent dependent function types between elements of type *El o* and *Set* (respectively *Set* and *El o*).

While it is not possible to quantify over TYPE in the λΠ-calculus modulo theory, this encoding allows to quantify over propositions—objects of type *El o*—and then inject them into TYPE using *Prf*. Similarly, we can quantify over sorts—objects of type *Set*—and then inject them into TYPE using *El*.

2.3 Theories with Prelude Encoding

In this paper, we consider theories that feature those basic notions of proposition and proof. More formally, we take theories of the form $\mathbb{T} = \Sigma_{pre} \cup \Sigma_{\mathbb{T}}$, where the user-defined constants $c : A \in \Sigma_{\mathbb{T}}$ have to be expressed in the prelude encoding.

Definition 2 (Theories with prelude encoding)**.** *We say that a theory $\mathbb{T} = \Sigma_{pre} \cup \Sigma_{\mathbb{T}}$ is a theory with prelude encoding when for every $c : A \in \Sigma_{\mathbb{T}}$, we have $\vdash A : \texttt{TYPE}$.*

The condition guarantees that the user-defined constants of $\Sigma_{\mathbb{T}}$ are indeed encoded in the prelude encoding. For instance, we cannot define nat : TYPE, but are forced to take nat : *Set*. Consequently inside a theory with prelude encoding, the only constants $c : A \in \Sigma$ with *A* a kind are *Set* (of type TYPE), *El* (of type *Set* → TYPE) and *Prf* (of type *El o* → TYPE).

For each rewrite rule $\ell \hookrightarrow r \in \Sigma$, the head-symbol of ℓ is a constant. It follows that, if $\Gamma \vdash \ell : A$, then *A* cannot be KIND. We thus have $\Gamma \vdash A : s$ with $s = \texttt{TYPE}$ or $s = \texttt{KIND}$. In particular, TYPE cannot occur in ℓ and *r*.

Example 1 (Natural numbers)**.** *We define a theory with prelude encoding $\mathbb{T}_n = \Sigma_{pre} \cup \Sigma_n$ for natural numbers.* nat *is the sort of natural numbers. We declare two constructors 0_n and succ_n, a relation $\geq_n$,*

and an induction principle rec_n.

$$
\begin{array}{ll}
\mathsf{nat}: & \mathit{Set} \\
0_n: & \mathit{El}\ \mathsf{nat} \\
\mathsf{succ}_n: & \mathit{El}\ \mathsf{nat} \to \mathit{El}\ \mathsf{nat} \\
\geq_n\ : & \mathit{El}\ \mathsf{nat} \to \mathit{El}\ \mathsf{nat} \to \mathit{El}\ o \\
\mathsf{ax}_n^1: & \Pi(x : \mathit{El}\ \mathsf{nat}).\ \mathit{Prf}\ (x \geq_n x) \\
\mathsf{ax}_n^2: & \Pi(x : \mathit{El}\ \mathsf{nat}).\ \mathit{Prf}\ (\mathsf{succ}_n\ x \geq_n x) \\
\mathsf{ax}_n^3: & \Pi(x,y,z : \mathit{El}\ \mathsf{nat}).\ \mathit{Prf}\ (x \geq_n y) \to \mathit{Prf}\ (y \geq_n z) \to \mathit{Prf}\ (x \geq_n z) \\
\mathsf{rec}_n: & \Pi(P : \mathit{El}\ \mathsf{nat} \to \mathit{El}\ o).\ \mathit{Prf}\ (P\ 0_n) \to \\
& [\Pi(x : \mathit{El}\ \mathsf{nat}).\ \mathit{Prf}\ (P\ x) \to \mathit{Prf}\ (P\ (\mathsf{succ}_n\ x))] \to \\
& \Pi(x : \mathit{El}\ \mathsf{nat}).\ \mathit{Prf}\ (P\ x)
\end{array}
$$

In this theory, we can prove $\Pi(x : \mathit{El}\ \mathsf{nat}).\ \mathit{Prf}\ (x \geq_n 0_n)$ *and* $\Pi(x : \mathit{El}\ \mathsf{nat}).\ \mathit{Prf}\ (\mathsf{succ}_n\ x \geq_n 0_n)$.

Example 2 (Integers)**.** *We define a theory with prelude encoding* $\mathbb{T}_i = \Sigma_{pre} \cup \Sigma_i$ *for integers.* int *is the sort of integers. We declare three constructors* 0_i, succ_i *and* pred_i*, a relation* $\geq_i$ *and a generalized induction principle* rec_i.

$$
\begin{array}{ll}
\mathsf{int}: & \mathit{Set} \\
0_i: & \mathit{El}\ \mathsf{int} \\
\mathsf{succ}_i: & \mathit{El}\ \mathsf{int} \to \mathit{El}\ \mathsf{int} \\
\mathsf{pred}_i: & \mathit{El}\ \mathsf{int} \to \mathit{El}\ \mathsf{int} \\
\geq_i\ : & \mathit{El}\ \mathsf{int} \to \mathit{El}\ \mathsf{int} \to \mathit{El}\ o \\
\mathsf{ax}_i^1: & \Pi(x : \mathit{El}\ \mathsf{int}).\ \mathit{Prf}\ (x \geq_i x) \\
\mathsf{ax}_i^2: & \Pi(x : \mathit{El}\ \mathsf{int}).\ \mathit{Prf}\ (\mathsf{succ}_i\ x \geq_i x) \\
\mathsf{ax}_i^3: & \Pi(x,y,z : \mathit{El}\ \mathsf{int}).\ \mathit{Prf}\ (x \geq_i y) \to \mathit{Prf}\ (y \geq_i z) \to \mathit{Prf}\ (x \geq_i z) \\
\mathsf{ax}_i^4: & \Pi(x : \mathit{El}\ \mathsf{int}).\ \Pi(P : \mathit{El}\ \mathsf{int} \to \mathit{El}\ o).\ \mathit{Prf}\ (P\ (\mathsf{succ}_i\ (\mathsf{pred}_i\ x))) \to \mathit{Prf}\ (P\ x) \\
\mathsf{ax}_i^5: & \Pi(x : \mathit{El}\ \mathsf{int}).\ \Pi(P : \mathit{El}\ \mathsf{int} \to \mathit{El}\ o).\ \mathit{Prf}\ (P\ (\mathsf{pred}_i\ (\mathsf{succ}_i\ x))) \to \mathit{Prf}\ (P\ x) \\
\mathsf{rec}_i: & \Pi(c : \mathit{El}\ \mathsf{int})(P : \mathit{El}\ \mathsf{int} \to \mathit{El}\ o).\ \mathit{Prf}\ (P\ c) \to \\
& [\Pi(x : \mathit{El}\ \mathsf{int}).\ \mathit{Prf}\ (x \geq_i c) \to \mathit{Prf}\ (P\ x) \to \mathit{Prf}\ (P\ (\mathsf{succ}_i\ x))] \to \\
& \Pi(x : \mathit{El}\ \mathsf{int}).\ \mathit{Prf}\ (x \geq_i c) \to \mathit{Prf}\ (P\ x)
\end{array}
$$

In this theory, we cannot prove $\Pi(x : \mathit{El}\ \mathsf{int}).\ \mathit{Prf}\ (x \geq_i 0_i)$ *and* $\Pi(x : \mathit{El}\ \mathsf{int}).\ \mathit{Prf}\ (\mathsf{succ}_i\ x \geq_i 0_i)$*, but we can prove* $\Pi(x : \mathit{El}\ \mathsf{int}).\ \mathit{Prf}\ (x \geq_i 0_i) \to \mathit{Prf}\ (\mathsf{succ}_i\ x \geq_i 0_i)$.

3 Interpretation in the $\lambda\Pi$-Calculus Modulo Theory

In this section, we define an interpretation of theories with prelude encoding. To do so, we first define the interpretation for the terms of the $\lambda\Pi$-calculus modulo theory, and then we extend it to theories with prelude encoding. Such an interpretation requires external parameters. In particular, we provide the parameters necessary for interpreting the prelude encoding. We show how the interpretation of a source theory $\mathbb{S}$ in a target theory $\mathbb{T}$ can be used to derive in $\mathbb{T}$ the theorems proved in $\mathbb{S}$. We conclude with an example: we provide the formal parameters for interpreting the theory of natural numbers $\mathbb{T}_n$ in the theory of integers $\mathbb{T}_i$.

3.1 Interpretation of Terms

Intuition. When we interpret the source theory $\mathbb{S}$ in the target theory $\mathbb{T}$, we want to represent every term t of $\mathbb{S}$ by a term t^* in $\mathbb{T}$, such that if t is of type A in $\mathbb{S}$ then t^* is of type A^* in $\mathbb{T}$. For instance, when interpreting the theory of natural numbers $\mathbb{T}_n$ in the theory of integers $\mathbb{T}_i$, we have to represent $El\ \mathsf{nat}$ by $(El\ \mathsf{nat})^*$. We would like to take $(El\ \mathsf{nat})^* \coloneqq \Sigma(z : El\ \mathsf{int}).\ Prf\ (z \geq_i 0_i)$. However, the λΠ-calculus modulo theory does not feature Σ-types, and it is therefore difficult to express $(El\ \mathsf{nat})^*$ in $\mathbb{T}_i$.

An alternative is to interpret the type of natural numbers $El\ \mathsf{nat}$ by the type of integers $El\ \mathsf{int}$, but we must guarantee that every integer representing a natural number is indeed non-negative. We naturally interpret the sort nat by int, 0_n by 0_i, succ_n by succ_i, and $\geq_n$ by $\geq_i$. The interpretation of the theorem $\Pi(x : El\ \mathsf{nat}).\ Prf\ (\mathsf{succ}_n\ x \geq_n 0_n)$ should not be $\Pi(x^* : El\ \mathsf{int}).\ Prf\ (\mathsf{succ}_i\ x^* \geq_i 0_i)$, which is generally false for integers. Instead, we must ensure that x^* is an integer corresponding to a natural number, meaning that we suppose a proof of $Prf\ (x^* \geq_i 0_i)$. Thus the interpretation of the theorem $\Pi(x : El\ \mathsf{nat}).\ Prf\ (\mathsf{succ}_n\ x \geq_n 0_n)$ should be $\Pi(x^* : El\ \mathsf{int}).\ Prf\ (x^* \geq_i 0_i) \to Prf\ (\mathsf{succ}_i\ x^* \geq_i 0_i)$.

Formal definition. Following this intuition, when interpreting a term t of type A in $\mathbb{S}$ by a term t^* of type A^* in $\mathbb{T}$, we must take into account that A^* is a type that encompasses A, but may be larger than A. In that respect, we introduce another term t^+ of type $A^+\ t^*$, where A^+ is a predicate asserting that a given object of type A^* satisfies the semantic of type A.

The interpretation of every constant c is given by two parameters c^* and c^+. The translation of an application $(t\ u)^*$ is $t^*\ u^*\ u^+$, since t^* takes as arguments u^* but also the witness u^+. Similarly, $(t\ u)^+$ is given by $t^+\ u^*\ u^+$. If the variable x occurs in t, then x^* and x^+ may occur in t^* and t^+. Hence $(\lambda(x : A).\ t)^*$ is given by $\lambda(x^* : A^*)(x^+ : A^+\ x^*).\ t^*$ and $(\lambda(x : A).\ t)^+$ is given by $\lambda(x^* : A^*)(x^+ : A^+\ x^*).\ t^+$.

The same intuition holds for dependent types $(\Pi(x : A).\ B)^*$. The predicate $(\Pi(x : A).\ B)^+$ asserts that an object f of type $(\Pi(x : A).\ B)^*$ corresponds to the semantic of $\Pi(x : A).\ B$. In other words, for every x^* of type A^* and x^+ of type $A^+\ x^*$, the term $f\ x^*\ x^+$ should satisfy the predicate B^+. When B is of type $\mathtt{TYPE}$, we take $(\Pi(x : A).\ B)^+ \coloneqq \lambda(f : (\Pi(x : A).\ B)^*).\ \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^+\ (f\ x^*\ x^+)$. However, we cannot do the same when B is of type $\mathtt{KIND}$, because this term would be ill-typed. Indeed, $(\Pi(x : A).\ B)^*$ has type $\mathtt{KIND}$, while the type of the bound variable f must have type $\mathtt{TYPE}$. To get around this issue, we introduce metavariables. We write $T\{X\}$ when the metavariable X occurs in T, and we write $T\{t\}$ for the term obtained when substituting X by t in T. When B has type $\mathtt{KIND}$, we take $(\Pi(x : A).\ B)^+\{X\} \coloneqq \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^+\{X\ x^*\ x^+\}$. Metavariables are only used for this purpose. In particular, they are always substituted and they never appear in typed terms.

Definition 3 (Interpretation of terms)**.** *The interpretation of terms of the λΠ-calculus modulo theory is given by the function $t \mapsto t^*$ defined inductively by*

$(x)^* \coloneqq x^*$ *(variable)*
$(c)^* \coloneqq c^*$ *(parameter)*
$\mathtt{TYPE}^* \coloneqq \mathtt{TYPE}$
$\mathtt{KIND}^* \coloneqq \mathtt{KIND}$
$(t\ u)^* \coloneqq t^*\ u^*\ u^+$
$(\lambda(x : A).\ t)^* \coloneqq \lambda(x^* : A^*)(x^+ : A^+\ x^*).\ t^*$
$(\Pi(x : A).\ B)^* \coloneqq \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^*$

and by the function $t \mapsto t^+$ defined inductively by

$(x)^+ \coloneqq x^+$ *(variable)*
$(c)^+ \coloneqq c^+$ *(parameter)*
$\mathtt{TYPE}^+\{X\} \coloneqq X \to \mathtt{TYPE}$
$\mathtt{KIND}^+ \coloneqq \mathtt{KIND}$
$(t\ u)^+ \coloneqq t^+\ u^*\ u^+$
$(\lambda(x:A).\ t)^+ \coloneqq \lambda(x^*:A^*)(x^+:A^+\ x^*).\ t^+$
$(\Pi(x:A).\ B)^+ \coloneqq \lambda(f:(\Pi(x:A).\ B)^*).\ \Pi(x^*:A^*)(x^+:A^+\ x^*).\ B^+\ (f\ x^*\ x^+)$ *if* $B:\mathtt{TYPE}$
$(\Pi(x:A).\ B)^+\{X\} \coloneqq \Pi(x^*:A^*)(x^+:A^+\ x^*).\ B^+\{X\ x^*\ x^+\}$ *if* $B:\mathtt{KIND}$.

where the X is a metavariable. The interpretation is extended to contexts with

$\langle\rangle^{*,+} \coloneqq \langle\rangle$
$(\Gamma, x:A)^{*,+} \coloneqq \Gamma^{*,+}, x^*:A^*, x^+:A^+\ x^*$.

When the free variable x occurs in t, then x^* and x^+ may both occur in t^* and t^+. As such, we do not define distinct translations Γ^* and Γ^+, but a single translation $\Gamma^{*,+}$, such that if $(x:A) \in \Gamma$ then $(x^*:A^*) \in \Gamma^{*,+}$ and $(x^+:A^+\ x^*) \in \Gamma^{*,+}$.

Parametricity. Remark that our interpretation is intuitively related to the parametricity translation [2]. Using parametricity, the translation $(t\ u)^*$ is given by $t^*\ u^*$, the translation $(\lambda(x:A).\ t)^*$ is given by $\lambda(x^*:A^*).\ t^*$, and the translation $(\Pi(x:A).\ B)^*$ is given by $\Pi(x^*:A^*).\ B^*$. In our interpretation, we focus on embeddings and we want to represent every type A of the source theory by a type A^* of the target theory. While Σ-types are well-suited for expressing such A^*, they are not defined in the $\lambda\Pi$-calculus modulo theory. That is why we have applied a *currying* operation on Σ-types. We therefore represent type A using a more general type A^*, and we guarantee that each term of type A^* representing a term of type A enjoys the predicate A^+. Consequently, the translation $(\Pi(x:A).\ B)^*$ is given by $\Pi(x^*:A^*)(x^+:A^+\ x^*).\ B^*$, the translation $(\lambda(x:A).\ t)^*$ is given by $\lambda(x^*:A^*)(x^+:A^+\ x^*).\ t^*$, and the translation $(t\ u)^*$ is given by $t^*\ u^*\ u^+$. The formal relation between the parametricity translation and our interpretation remains to be investigated.

3.2 Parameters for the Prelude Encoding

We aim at interpreting a source theory $\mathbb{S}$ in a target theory $\mathbb{T}$, when $\mathbb{S}$ and $\mathbb{T}$ are theories with prelude encoding. Such an interpretation is parametrized by the terms of $\mathbb{T}$ that correspond to the constants of $\mathbb{S}$. In particular, we have to provide the parameters for the constants of the prelude encoding.

When $\vdash t:A$ in $\mathbb{S}$, we want to have $\vdash t^*:A^*$ in $\mathbb{T}$. Moreover, we want $\vdash A^+:A^* \to \mathtt{TYPE}$ in $\mathbb{T}$ when $A = \mathtt{TYPE}$. These conditions lead to the definition of Set^*, Set^+, El^*, El^+, Prf^*, Prf^+ and o^*. When t is of type $Prf\ p$, we need a witness t^+ of type $(Prf\ p)^+\ t^*$ asserting that t^* is indeed a proof of p^*. Since t^* is of type $Prf\ p^*$, it is necessarily a proof of p^*, and we define Prf^+ so that we can always choose t^+ to be t^*. The predicate o^+ asserts that an object p^* of type $El\ o$ is indeed a proposition, so we choose o^+ to be $\lambda(z:El\ o).\ z \Rightarrow_d (\lambda(x:Prf\ z).\ z)$. Consequently, it is is always possible to find a witness p^+ of type

$Prf\ (o^+\ p^*)$, that is $Prf\ p^* \to Prf\ p^*$.

$$
\begin{array}{l}
Set^* \coloneqq Set \\
Set^+ \coloneqq \lambda(x : Set).\ El\ x \to El\ o \\
o^* \coloneqq o \\
o^+ \coloneqq \lambda(z : El\ o).\ z \Rightarrow_d (\lambda(x : Prf\ z).\ z) \\
El^* \coloneqq \lambda(x^* : Set)(x^+ : El\ x^* \to El\ o).\ El\ x^* \\
El^+ \coloneqq \lambda(u^* : Set)(u^+ : El\ u^* \to El\ o)(x : El\ u^*).\ Prf\ (u^+\ x) \\
Prf^* \coloneqq \lambda(x^* : El\ o)(x^+ : Prf\ (o^+\ x^*)).\ Prf\ x^* \\
Prf^+ \coloneqq \lambda(u^* : El\ o)(u^+ : Prf\ (o^+\ u^*))(x : Prf\ u^*).\ Prf\ u^*
\end{array}
$$

Parameters $\rightsquigarrow_d{}^*$ and $\rightsquigarrow_d{}^+$ are defined so that $(El\ (a \rightsquigarrow_d b))^@ \equiv_{\beta\Sigma} (\Pi(x : El\ a).\ El\ (b\ x))^@$ for $@ \in \{*, +\}$.

$$
\begin{array}{rl}
\rightsquigarrow_d{}^* \coloneqq & \lambda(a^* : Set)(a^+ : El\ a^* \to El\ o)(b^* : \Pi(x^* : El\ a^*).\ Prf\ (a^+\ x^*) \to Set). \\
& \lambda(b^+ : \Pi(x^* : El\ a^*)(x^+ : Prf\ (a^+\ x^*)).\ El\ (b^*\ x^*\ x^+) \to El\ o). \\
& a^* \rightsquigarrow_d (\lambda(x^* : El\ a^*).\ \pi\ (a^+\ x^*)\ (b^*\ x^*)) \\
\rightsquigarrow_d{}^+ \coloneqq & \lambda(a^* : Set)(a^+ : El\ a^* \to El\ o)(b^* : \Pi(x^* : El\ a^*).\ Prf\ (a^+\ x^*) \to Set). \\
& \lambda(b^+ : \Pi(x^* : El\ a^*)(x^+ : Prf\ (a^+\ x^*)).\ El\ (b^*\ x^*\ x^+) \to El\ o). \\
& \lambda(f : El\ (a \rightsquigarrow_d b)^*). \\
& \forall\ a^*\ (\lambda(x^* : El\ a^*).\ (a^+\ x^*) \Rightarrow_d (\lambda(x^+ : Prf\ (a^+\ x^*)).\ b^+\ x^*\ x^+\ (f\ x^*\ x^+)))
\end{array}
$$

Parameter $\Rightarrow_d{}^*$ is defined so that $(Prf\ (a \Rightarrow_d b))^* \equiv_{\beta\Sigma} (\Pi(x : Prf\ a).\ Prf\ (b\ x))^*$. Because the condition $(Prf\ (a \Rightarrow_d b))^+ \equiv_{\beta\Sigma} (\Pi(x : Prf\ a).\ Prf\ (b\ x))^+$ holds regardless of the definition of $\Rightarrow_d{}^+$, we choose $\Rightarrow_d{}^+$ so that $\vdash \Rightarrow_d{}^+ : (\Pi(a : El\ o).\ (Prf\ a \to El\ o) \to El\ o)^+ \Rightarrow_d{}^*$.

$$
\begin{array}{rl}
\Rightarrow_d{}^* \coloneqq & \lambda(a^* : El\ o)(a^+ : Prf\ (o^+\ a^*))(b^* : \Pi(x^* : Prf\ a^*).\ Prf\ a^* \to El\ o). \\
& \lambda(b^+ : \Pi(x^* : Prf\ a^*)(x^+ : Prf\ a^*).\ Prf\ (o^+\ (b^*\ x^*\ x^+))). \\
& a^* \Rightarrow_d (\lambda(x^* : Prf\ a^*).\ a^* \Rightarrow_d\ (b^*\ x^*)) \\
\Rightarrow_d{}^+ \coloneqq & \lambda(a^* : El\ o)(a^+ : Prf\ (o^+\ a^*))(b^* : \Pi(x^* : Prf\ a^*).\ Prf\ a^* \to El\ o). \\
& \lambda(b^+ : \Pi(x^* : Prf\ a^*)(x^+ : Prf\ a^*).\ Prf\ (o^+\ (b^*\ x^*\ x^+))). \\
& \lambda(p : Prf\ (a \Rightarrow_d b)^*).\ p
\end{array}
$$

Parameters π^* and π^+ are defined so that $(El\ (\pi\ a\ b))^@ \equiv_{\beta\Sigma} (\Pi(x : Prf\ a).\ El\ (b\ x))^@$ for $@ \in \{*, +\}$.

$$
\begin{array}{rl}
\pi^* \coloneqq & \lambda(a^* : El\ o)(a^+ : Prf\ (o^+\ a^*))(b^* : \Pi(x^* : Prf\ a^*).\ Prf\ a^* \to Set). \\
& \lambda(b^+ : \Pi(x^* : Prf\ a^*)(x^+ : Prf\ a^*).\ El\ (b^*\ x^*\ x^+) \to El\ o). \\
& \pi\ a^*\ (\lambda(x^* : Prf\ a^*).\ \pi\ a^*\ (b^*\ x^*)) \\
\pi^+ \coloneqq & \lambda(a^* : El\ o)(a^+ : Prf\ (o^+\ a^*))(b^* : \Pi(x^* : Prf\ a^*).\ Prf\ a^* \to Set). \\
& \lambda(b^+ : \Pi(x^* : Prf\ a^*)(x^+ : Prf\ a^*).\ El\ (b^*\ x^*\ x^+) \to El\ o). \\
& \lambda(f : El\ (\pi\ a\ b)^*). \\
& a^* \Rightarrow_d\ (\lambda(x^* : Prf\ a^*).\ a^* \Rightarrow_d (\lambda(x^+ : Prf\ a^*).\ b^+\ x^*\ x^+\ (f\ x^*\ x^+)))
\end{array}
$$

Parameter $\forall^*$ is defined so that $(Prf\ (\forall\ a\ b))^* \equiv_{\beta\Sigma} (\Pi(x : El\ a).\ Prf\ (b\ x))^*$. Because the condition $(Prf\ (\forall\ a\ b))^+ \equiv_{\beta\Sigma} (\Pi(x : El\ a).\ Prf\ (b\ x))^+$ holds regardless of the definition of $\forall^+$, we choose $\forall^+$ so that $\vdash \forall^+ : (\Pi(a : Set).\ (El\ a \to El\ o) \to El\ o)^+\ \forall^*$.

$$
\begin{array}{rl}
\forall^* \coloneqq & \lambda(a^* : Set)(a^+ : El\ a^* \to El\ o)(b^* : \Pi(x^* : El\ a^*).\ Prf\ (a^+\ x^*) \to El\ o). \\
& \lambda(b^+ : \Pi(x^* : El\ a^*)(x^+ : Prf\ (a^+\ x^*)).\ Prf\ (o^+\ (b^*\ x^*\ x^+))). \\
& \forall\ a^*\ (\lambda(x^* : El\ a^*).\ (a^+\ x^*) \Rightarrow_d (b^*\ x^*))
\end{array}
$$

$$\begin{array}{ll}\forall^+ := & \lambda(a^* : Set)(a^+ : El\ a^* \to El\ o)(b^* : \Pi(x^* : El\ a^*).\ Prf\ (a^+\ x^*) \to El\ o).\\ & \lambda(b^+ : \Pi(x^* : El\ a^*)(x^+ : Prf\ (a^+\ x^*)).\ Prf\ (o^+\ (b^*\ x^*\ x^+))).\\ & \lambda(p : Prf\ (\forall\ a\ b)^*).\ p\end{array}$$

The parameters chosen for the constants of the prelude encoding satisfy the expected properties. For any $c : A \in \Sigma_{pre}$, we have $\vdash c^* : A^*$ and $\vdash c^+ : A^+\ c^*$. Moreover, the interpretation respects the conversion relation, meaning that for each rewrite rule $\ell \hookrightarrow r$ of Σ_{pre}, we have both $\ell^* \equiv_{\beta\Sigma} r^*$ and $\ell^+ \equiv_{\beta\Sigma} r^+$.

Proposition 1. *Let $c : A \in \Sigma_{pre}$.*

1. *We have $\vdash c^* : A^*$.*
2. *(a) If $\vdash A : \texttt{TYPE}$ then $\vdash c^+ : A^+\ c^*$.*
   (b) If $\vdash A : \texttt{KIND}$ then $\vdash c^+ : A^+\{c^*\}$.

Proof. By simple verification. The result has been checked in DEDUKTI, see the definitions of the parameters in the file `lo_sp.dk`[1]. □

Proposition 2. *For every $\ell \hookrightarrow r \in \Sigma_{pre}$, we have $\ell^* \equiv_{\beta\Sigma} r^*$ and $\ell^+ \equiv_{\beta\Sigma} r^+$.*

Proof. We only show the case $El\ (a \rightsquigarrow_d b) \hookrightarrow \Pi(x : El\ a).\ El\ (b\ x)$.

$$\begin{array}{lll}\text{We have } (El\ (a \rightsquigarrow_d b))^* & \equiv_{\beta\Sigma} & El\ (a \rightsquigarrow_d b)^*\\ & \equiv_{\beta\Sigma} & El\ (a^* \rightsquigarrow_d (\lambda x^*.\ \pi\ (a^+\ x^*)\ (b^*\ x^*)))\\ & \equiv_{\beta\Sigma} & \Pi(x^* : El\ a^*).\ El\ (\pi\ (a^+\ x^*)\ (b^*\ x^*))\\ & \equiv_{\beta\Sigma} & \Pi(x^* : El\ a^*)(x^+ : Prf\ (a^+\ x^*)).\ El\ (b^*\ x^*\ x^+)\\ & \equiv_{\beta\Sigma} & \Pi(x^* : (El\ a)^*)(x^+ : (El\ a)^+\ x^*).\ (El\ (b\ x))^*\\ & \equiv_{\beta\Sigma} & (\Pi(x : El\ a).\ El\ (b\ x))^*\\ \text{and } (El\ (a \rightsquigarrow_d b))^+ & \equiv_{\beta\Sigma} & \lambda(f : El\ (a \rightsquigarrow_d b)^*).\ Prf\ ((a \rightsquigarrow_d b)^+\ f)\\ & \equiv_{\beta\Sigma} & \lambda(f : El\ (a \rightsquigarrow_d b)^*).\\ & & Prf\ (\forall\ a^*\ (\lambda x^*.\ (a^+\ x^*) \Rightarrow_d (\lambda x^+.\ b^+\ x^*\ x^+\ (f\ x^*\ x^+))))\\ & \equiv_{\beta\Sigma} & \lambda(f : El\ (a \rightsquigarrow_d b)^*).\ \Pi(x^* : El\ a^*).\\ & & Prf\ ((a^+\ x^*) \Rightarrow_d (\lambda x^+.\ b^+\ x^*\ x^+\ (f\ x^*\ x^+)))\\ & \equiv_{\beta\Sigma} & \lambda(f : (El\ (a \rightsquigarrow_d b))^*).\ \Pi(x^* : El\ a^*)(x^+ : Prf\ (a^+\ x^*)).\\ & & Prf\ (b^+\ x^*\ x^+\ (f\ x^*\ x^+))\\ & \equiv_{\beta\Sigma} & \lambda(f : (\Pi(x : El\ a).\ El\ (b\ x))^*).\ \Pi(x^* : (El\ a)^*)(x^+ : (El\ a)^+\ x^*).\\ & & (El\ (b\ x))^+\ (f\ x^*\ x^+)\\ & \equiv_{\beta\Sigma} & (\Pi(x : El\ a).\ El\ (b\ x))^+.\end{array}$$

The result has been checked in DEDUKTI for the four rewrite rules, see the `#ASSERT` commands in the file `lo_sp.dk`. □

3.3 Interpretation of Theories

The interpretation of a source theory $\mathbb{S}$ in a target theory $\mathbb{T}$ is given by the parameters c^* and c^+, for each constant c of Σ. We have provided the parameters for the constants of Σ_{pre}, but the parameters for the constants of $\Sigma_{\mathbb{S}}$ remain to be given by the user.

[1] All the DEDUKTI files are available at `https://github.com/thomastraversie/InterpDK`.

Definition 4 (Interpretation of theories)**.** *Let $\mathbb{S}$ and $\mathbb{T}$ be two theories with prelude encoding. We say that $\mathbb{S}$ has an interpretation in $\mathbb{T}$ when:*

1. *for each constant $c : A \in \Sigma_{\mathbb{S}}$, we have a term c^* such that $\vdash c^* : A^*$ in $\mathbb{T}$,*
2. *for each constant $c : A \in \Sigma_{\mathbb{S}}$, we have a term c^+ such that $\vdash c^+ : A^+\ c^*$ in $\mathbb{T}$,*
3. *for each rewrite rule $\ell \hookrightarrow r \in \Sigma_{\mathbb{S}}$, we have $\ell^* \equiv_{\beta\Sigma} r^*$ and $\ell^+ \equiv_{\beta\Sigma} r^+$ in $\mathbb{T}$.*

Remark that, in the third item, ℓ^+ and r^+ do not contain metavariables, as we have seen that `TYPE` cannot occur in ℓ and r.

If we cannot interpret the rewrite rules of $\mathbb{S}$ into conversions in $\mathbb{T}$, we can nonetheless replace the rewrite rules of $\mathbb{S}$ by equational axioms—that is by typed constants—and then interpret such constants in $\mathbb{T}$. So as to replace user-defined rewrite rules by equational axioms [6], we add an equality in our signature, and we use functional extensionality, uniqueness of identity proofs, and the congruence of equality on applications.

The λΠ-calculus modulo theory features substitutions in the type of an application—in the case of dependent types—and features user-defined rewrite rules. So that the translation of a provable judgment remains provable, it is important to maintain substitution and conversion through the translations $t \mapsto t^*$ and $t \mapsto t^+$. For each variable z occurring in a term t, the two variables z^* and z^+ occur in the translated terms t^* and t^+. The translation $(t[z \leftarrow w])^*$ is thus given by $t^*[z^* \leftarrow w^*][z^+ \leftarrow w^+]$.

Proposition 3 (Substitution)**.** *Let t and w be two terms and z be a variable. We have:*

- $(t[z \leftarrow w])^* = t^*[z^* \leftarrow w^*][z^+ \leftarrow w^+]$.
- $(t[z \leftarrow w])^+ = t^+[z^* \leftarrow w^*][z^+ \leftarrow w^+]$.

Proof. By induction on the term t. □

Proposition 4 (Conversion)**.** *If $A \equiv_{\beta\Sigma} B$ in $\mathbb{S}$, then $A^* \equiv_{\beta\Sigma} B^*$ and $A^+ \equiv_{\beta\Sigma} B^+$ in $\mathbb{T}$.*

Proof. We prove the result by induction on the formation of $A \equiv_{\beta\Sigma} B$.

- We have $(\lambda(x:A).\,t)\ u)^* = (\lambda(x^*:A^*)(x^+:A^+\ x^*).\,t^*)\ u^*\ u^+$, which β-reduces to $t^*[x^* \leftarrow u^*][x^+ \leftarrow u^+]$, that is $(t[x \leftarrow u])^*$ following Proposition 3. Similarly, $((\lambda(x:A).\,t)\ u)^+ \equiv_{\beta\Sigma} (t[x \leftarrow u])^+$.
- For each $\ell \hookrightarrow r \in \Sigma$ and any substitution θ, we have $\ell^* \equiv_{\beta\Sigma} r^*$ by definition and Proposition 2. Using Proposition 3, we have $(\ell\theta)^* = \ell^*\theta^{*,+}$ and $(r\theta)^* = r^*\theta^{*,+}$, where $\theta^{*,+}$ is defined so that if θ substitutes z by w, then $\theta^{*,+}$ substitutes z^* by w^* and z^+ by w^+. Therefore $(\ell\theta)^* = \ell^*\theta^{*,+} \equiv_{\beta\Sigma} r^*\theta^{*,+} = (r\theta)^*$. Similarly, we have $(\ell\theta)^+ = \ell^+\theta^{*,+} \equiv_{\beta\Sigma} r^+\theta^{*,+} = (r\theta)^+$.
- For closure by context, we only show the λ-abstraction case. Suppose that $\lambda(x:A).\,t \equiv_{\beta\Sigma} \lambda(x:B).\,u$ derives from $A \equiv_{\beta\Sigma} B$ and $t \equiv_{\beta\Sigma} u$. By induction, we have $A^* \equiv_{\beta\Sigma} B^*$, and $A^+ \equiv_{\beta\Sigma} B^+$, and $t^* \equiv_{\beta\Sigma} u^*$, and $t^+ \equiv_{\beta\Sigma} u^+$. We derive that $\lambda(x^*:A^*)(x^+:A^+\ x^*).\,t^* \equiv_{\beta\Sigma} \lambda(x^*:B^*)(x^+:B^+\ x^*).\,u^*$, that is $(\lambda(x:A).\,t)^* \equiv_{\beta\Sigma} (\lambda(x:B).\,u)^*$. Similarly, $(\lambda(x:A).\,t)^+ \equiv_{\beta\Sigma} (\lambda(x:B).\,u)^+$.
- Reflexivity, symmetry and transitivity are immediate.

□

We have at hand all the tools allowing us to prove that, when $\mathbb{S}$ has an interpretation in $\mathbb{T}$, any provable judgment in $\mathbb{S}$ is interpreted as a provable judgment in $\mathbb{T}$. The first item of the theorem concerns well-formedness judgments. The second item concerns typing judgments with respect to the translation $t \mapsto t^*$, and the third item concerns typing judgments with respect to the translation $t \mapsto t^+$.

Theorem 1 (Interpretation)**.** *Let $\mathbb{S}$ and $\mathbb{T}$ be two theories with prelude encoding, such that $\mathbb{S}$ has an interpretation in $\mathbb{T}$.*

1. *If $\vdash \Gamma$ in $\mathbb{S}$, then $\vdash \Gamma^{*,+}$ in $\mathbb{T}$.*
2. *If $\Gamma \vdash t : A$ in $\mathbb{S}$ then $\Gamma^{*,+} \vdash t^* : A^*$ in $\mathbb{T}$.*
3. *(a) If $\Gamma \vdash t : A$ and $\Gamma \vdash A : \mathtt{TYPE}$ in $\mathbb{S}$, then $\Gamma^{*,+} \vdash t^+ : A^+\ t^*$ in $\mathbb{T}$.*
   (b) If $\Gamma \vdash t : A$ and $\Gamma \vdash A : \mathtt{KIND}$ in $\mathbb{S}$, then $\Gamma^{*,+} \vdash t^+ : A^+\{t^*\}$ in $\mathbb{T}$.
   (c) If $\Gamma \vdash A : \mathtt{KIND}$ in $\mathbb{S}$, then for every t such that $\Gamma^{*,+} \vdash t : A^*$ in $\mathbb{T}$, we have $\Gamma^{*,+} \vdash A^+\{t\} : \mathtt{KIND}$.

Proof. We proceed by induction on the derivation. We only show the most interesting cases.

- CONST: By induction, we have $\vdash \Gamma^{*,+}$ and $\vdash A^* : s^*$. Since $c : A \in \Sigma$, we have $\vdash c^* : A^*$. We derive $\Gamma^{*,+} \vdash c^* : A^*$ by weakening. If $s = \mathtt{TYPE}$, then $\vdash c^+ : A^+\ c^*$ and we derive $\Gamma^{*,+} \vdash c^+ : A^+\ c^*$ by weakening. If $s = \mathtt{KIND}$, then $\vdash c^+ : A^+\{c^*\}$ and we derive $\Gamma^{*,+} \vdash c^+ : A^+\{c^*\}$ by weakening.
- PROD: By induction, we have $\Gamma^{*,+} \vdash A^* : \mathtt{TYPE}$, and $\Gamma^{*,+} \vdash A^+ : A^* \to \mathtt{TYPE}$, and $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash B^* : s^*$. Using PROD, we get $\Gamma^{*,+} \vdash \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^* : s^*$.

  Suppose that $s = \mathtt{TYPE}$. By induction, $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash B^+ : B^* \to \mathtt{TYPE}$. By weakening, we have $\Gamma^{*,+}, f : (\Pi(x : A).\ B)^*, x^* : A^*, x^+ : A^+\ x^* \vdash B^+ : B^* \to \mathtt{TYPE}$. Since $\Gamma^{*,+}, f : (\Pi(x : A).\ B)^*, x^* : A^*, x^+ : A^+\ x^* \vdash B^+\ (f\ x^*\ x^+) : \mathtt{TYPE}$, we derive $\Gamma^{*,+} \vdash \lambda(f : (\Pi(x : A).\ B)^*).\ \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^+\ (f\ x^*\ x^+) : (\Pi(x : A).\ B)^* \to \mathtt{TYPE}$, which corresponds to $\Gamma^{*,+} \vdash (\Pi(x : A).\ B)^+ : \mathtt{TYPE}^+\{(\Pi(x : A).\ B)^*\}$.

  Suppose that $s = \mathtt{KIND}$ and that we have $\Gamma^{*,+} \vdash t : (\Pi(x : A).\ B)^*$. Since $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash t\ x^*\ x^+ : B^*$, by induction we get $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash B^+\{t\ x^*\ x^+\} : \mathtt{KIND}$. We derive $\Gamma^{*,+} \vdash \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^+\{t\ x^*\ x^+\} : \mathtt{KIND}$, that is $\Gamma^{*,+} \vdash (\Pi(x : A).\ B)^+\{t\} : \mathtt{KIND}$.
- ABS: By induction, we have $\Gamma^{*,+} \vdash A^* : \mathtt{TYPE}$, and $\Gamma^{*,+} \vdash A^+ : A^* \to \mathtt{TYPE}$, and $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash B^* : s^*$, and $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash t^* : B^*$, . We derive $\Gamma^{*,+} \vdash \lambda(x^* : A^*)(x^+ : A^+\ x^*).\ t^* : \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^*$, that is $\Gamma^{*,+} \vdash (\lambda(x : A).\ t)^* : (\Pi(x : A).\ B)^*$.

  Suppose that $s = \mathtt{TYPE}$. By induction, we have $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash B^+ : B^* \to \mathtt{TYPE}$ and $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash t^+ : B^+\ t^*$. We derive $\Gamma^{*,+} \vdash \lambda(x^* : A^*)(x^+ : A^+\ x^*).\ t^+ : \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^+\ t^*$. Using CONV, we conclude that $\Gamma^{*,+} \vdash (\lambda(x : A).\ t)^+ : (\Pi(x : A).\ B)^+\ (\lambda(x : A).\ t)^*$.

  Suppose that $s = \mathtt{KIND}$. By induction, we have $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash B^+\{t^*\} : \mathtt{KIND}$ and $\Gamma^{*,+}, x^* : A^*, x^+ : A^+\ x^* \vdash t^+ : B^+\{t^*\}$. We derive $\Gamma^{*,+} \vdash \lambda(x^* : A^*)(x^+ : (A^+\ x^*)).\ t^+ : \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^+\{t^*\}$, that is $\Gamma^{*,+} \vdash (\lambda(x : A).\ t)^+ : (\Pi(x : A).\ B)^+\{(\lambda(x : A).\ t)^*\}$ using CONV.
- APP: By induction, we have $\Gamma^{*,+} \vdash t^* : \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^*$, and $\Gamma^{*,+} \vdash u^* : A^*$, and $\Gamma^{*,+} \vdash u^+ : A^+\ u^*$. We derive $\Gamma^{*,+} \vdash t^*\ u^*\ u^+ : B^*[x^* \leftarrow u^*][x^+ \leftarrow u^+]$. Using Proposition 3, we conclude that $\Gamma^{*,+} \vdash (t\ u)^* : (B[x \leftarrow u])^*$.

  Suppose that $\Gamma \vdash \Pi(x : A).\ B : \mathtt{TYPE}$ (and thus $\Gamma \vdash B : \mathtt{TYPE}$). By induction, we have $\Gamma^{*,+} \vdash t^+ : \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^+\ (t^*\ x^*\ x^+)$. It follows that $\Gamma^{*,+} \vdash t^+\ u^*\ u^+ : B^+[x^* \leftarrow u^*][x^+ \leftarrow u^+]\ (t^*\ u^*\ u^+)$. Using Proposition 3, we conclude that $\Gamma^{*,+} \vdash (t\ u)^+ : (B[x \leftarrow u])^+\ (t\ u)^*$.

  Suppose that $\Gamma \vdash \Pi(x : A).\ B : \mathtt{KIND}$ (and thus $\Gamma \vdash B : \mathtt{KIND}$). By induction, we have $\Gamma^{*,+} \vdash t^+ : \Pi(x^* : A^*)(x^+ : A^+\ x^*).\ B^+\{t^*\ x^*\ x^+\}$. It follows that $\Gamma^{*,+} \vdash t^+\ u^*\ u^+ : (B^+\{t^*\ x^*\ x^+\})[x^* \leftarrow u^*][x^+ \leftarrow u^+]$. Using Proposition 3, we conclude that $\Gamma^{*,+} \vdash (t\ u)^+ : (B[x \leftarrow u])^+\{(t\ u)^*\}$.
- CONV: We conclude using the induction hypotheses and Proposition 4.

□

Given an interpretation of a source theory $\mathbb{S}$ in a target theory $\mathbb{T}$, the results proved in $\mathbb{S}$ are automatically transported to $\mathbb{T}$. The interpretation of $\mathbb{S}$ in $\mathbb{T}$ only requires the parameters c^* and c^+ in $\mathbb{T}$ for each user-defined constant c of $\mathbb{S}$. Once we have an interpretation of $\mathbb{S}$ in $\mathbb{T}$, it is possible to prove that $\mathbb{S}$ is consistent provided that $\mathbb{T}$ is so. In the λΠ-calculus modulo theory, we say that a theory is inconsistent when we can build a term that takes a proposition and returns one of its proofs, that is when there exists a term t such that $\vdash t : \Pi(P : El\ o).\ Prf\ P$.

Theorem 2 (Relative consistency)**.** *Let $\mathbb{S}$ and $\mathbb{T}$ be two theories with prelude encoding, such that $\mathbb{S}$ has an interpretation in $\mathbb{T}$. If $\mathbb{T}$ is consistent, then $\mathbb{S}$ is consistent too.*

Proof. Assume that $\mathbb{S}$ is inconsistent, meaning that we have a term $\vdash t : \Pi(P : El\ o).\ Prf\ P$. By applying Theorem 1, we get $\vdash t : \Pi(P^* : El\ o)(P^+ : Prf\ P^* \to Prf\ P^*).\ Prf\ P^*$. We take the term $t' \coloneqq \lambda(P^* : El\ o).\ t\ P^*\ (\lambda(x : Prf\ P^*).\ x)$ and we have $\vdash t' : \Pi(P^* : El\ o).\ Prf\ P^*$. It follows that $\mathbb{T}$ is inconsistent. □

3.4 Examples of Interpretation

We illustrate the interpretation with two examples. First, we detail the embedding of the theory of natural numbers into the theory of integers. This example has been implemented in DEDUKTI. Second, we give an informal presentation of the embedding of Zermelo set theory into a theory where sets are represented by graphs. These two examples exemplify the practicality and limitations of this interpretation.

3.4.1 Natural Numbers and Integers

We aim at interpreting the theory of natural numbers $\mathbb{T}_n$ in the theory of integers $\mathbb{T}_i$. We intuitively take $\mathsf{nat}^* \coloneqq \mathsf{int}$. An integer is a non-negative natural number, so the predicate asserting that an integer is a natural number is defined by $\mathsf{nat}^+ \coloneqq \lambda z.\ z \geq_i 0_i$. The interpretation of 0_n is given by $0_n^* \coloneqq 0_i$, and we choose $0_n^+ \coloneqq \mathsf{ax}_i^1\ 0_i$ for the proof of $0_n^* \geq_i 0_i$. We take $\mathsf{succ}_n^* \coloneqq \lambda x^*.\ \lambda x^+.\ \mathsf{succ}_i\ x^*$ and $\mathsf{succ}_n^+ \coloneqq \lambda x^*.\ \lambda x^+.\ \mathsf{ax}_i^3\ (\mathsf{succ}_i\ x^*)\ x^*\ 0_i\ (\mathsf{ax}_i^2\ x^*)\ x^+$. For the interpretation of $\geq_n$, we choose $\geq_n^* \coloneqq \lambda x^*.\ \lambda x^+.\ \lambda y^*.\ \lambda y^+.\ x^* \geq_i y^*$. Given that $\geq_n$ returns a proposition, the parameter $\geq_n^+$ must have type $\Pi x^*.\ \Pi x^+.\ \Pi y^*.\ \Pi y^+.\ Prf\ (x^* \geq_i y^*) \to Prf\ (x^* \geq_i y^*)$, which has an immediate inhabitant. The interpretation of ax_i^1 is given by $(\mathsf{ax}_i^1)^* \coloneqq \lambda x^*.\ \lambda x^+.\ \mathsf{ax}_i^1\ x^*$. Since ax_i^1 returns a proof, and by definition of Prf^+, both $(\mathsf{ax}_i^1)^*$ and $(\mathsf{ax}_i^1)^+$ have the same type, so we can take $(\mathsf{ax}_i^1)^+ \coloneqq (\mathsf{ax}_i^1)^*$. The parameters for ax_i^2 and ax_i^3 are chosen correspondingly.

When defining the parameter rec_n^*, we assume P^* of type $\Pi(x^* : El\ \mathsf{nat}^*).\ Prf\ (x^* \geq_i 0_i) \to El\ o$. We must apply rec_i to a predicate of type $El\ \mathsf{nat}^* \to El\ o$, which asserts that an integer z is non-negative and that, given a proof h_z of its non-negativity, it holds $P^*\ z\ h_z$. Such a predicate can be encoded using $\forall$ and $\Rightarrow_d$. At some point in the proof, we want to show $P^*\ z\ h_z$, but we can only derive $P^*\ z\ h'_z$, where h_z and h'_z are two proofs of $z \geq_i 0_i$. To overcome this problem, we suppose *proof irrelevance*

$$\mathsf{proof\_irr} : \Pi(p : El\ o)(h\ h' : Prf\ p)(Q : Prf\ p \to El\ o).\ Prf\ (Q\ h) \to Prf\ (Q\ h')$$

which states that two proofs of the same proposition are equal.

Using this interpretation of natural numbers into integers, we can derive for free the theorems of $\mathbb{T}_n$ in $\mathbb{T}_i$. For instance, we can show in $\mathbb{T}_n$ that $\vdash \mathsf{thm} : \Pi(x : El\ \mathsf{nat}).\ Prf\ (\mathsf{succ}_n\ x \geq_n 0_n)$, where thm is a proof that uses rec_n, ax_n^1, ax_n^2 and ax_n^3. The interpretation of $\mathbb{T}_n$ in $\mathbb{T}_i$ allows us to directly derive $\vdash \mathsf{thm}^* : \Pi(x^* : El\ \mathsf{int}).\ Prf\ (x^* \geq_i 0_i) \to Prf\ (\mathsf{succ}_i\ x^* \geq_i 0_i)$ in $\mathbb{T}_i$.

The complete interpretation of natural numbers into integers has been formalized in DEDUKTI, and is available in the file `nat_sp.dk`.

3.4.2 Sets and Pointed Graphs

Sets can be represented by a more primitive notion of pointed graphs, such that this encoding satisfies Zermelo set theory [11]. Pointed graphs are directed graphs with a distinguished node—the root. In the $\lambda\Pi$-calculus modulo theory, pointed graphs are implemented [5] thanks to sorts graph and node of type *Set*. The predicate $\mathsf{eta} : \mathit{El}\ \mathsf{graph} \to \mathit{El}\ \mathsf{node} \to \mathit{El}\ \mathsf{node} \to \mathit{El}\ o$ is such that $\mathsf{eta}\ a\ x\ y$ is the proposition asserting that there is an edge in pointed graph a from node y to node x. The operator $\mathsf{root} : \mathit{El}\ \mathsf{graph} \to \mathit{El}\ \mathsf{node}$ returns the root of a pointed graph, and $\mathsf{cr} : \mathit{El}\ \mathsf{graph} \to \mathit{El}\ \mathsf{node} \to \mathit{El}\ \mathsf{graph}$ is such that $\mathsf{cr}\ a\ x$ corresponds to the pointed graph a in which the root is now at node x.

The different constructors on sets—unions, pairs, powersets and comprehension—are defined via rewrite rules using the structure of pointed graphs. At the end, every axiom of Zermelo set theory is a theorem in the theory of pointed graphs. Hence we can naturally interpret Zermelo set theory in the theory of pointed graphs. Remark that every pointed graph represents a set. It follows that the predicates asserting that an object of type $\mathit{El}\ \mathsf{graph}$ is indeed a set are not necessary.

The theory of pointed graphs is more computational than the usual Zermelo set theory. In particular, it satisfies a normalization theorem in deduction modulo theory [11]. Using such an interpretation, the theorems proved in Zermelo set theory can be transferred to the theory of pointed graphs.

4 Conclusion

In this paper, we have defined an interpretation of theories of the $\lambda\Pi$-calculus modulo theory with prelude encoding, given well-suited parameters for interpreting the constants of the source theory. If a source theory $\mathbb{S}$ has an interpretation in a target theory $\mathbb{T}$, then the theorems proved in $\mathbb{S}$ come for free in $\mathbb{T}$. At the end, we obtain a relative consistency result, establishing that the consistency of the theory $\mathbb{T}$ entails the consistency of the theory $\mathbb{S}$.

This interpretation applies when $\mathbb{S}$ can be embedded into $\mathbb{T}$. In particular, we allow the interpretation of a type A of $\mathbb{S}$ by a more general type A^* of $\mathbb{T}$. As a consequence, we ensure that, for every term t of type A in $\mathbb{S}$, its interpretation t^* of type A^* in $\mathbb{T}$ indeed satisfies the predicate A^+. Such an interpretation is well-suited when we embed a source theory into a more general target theory, as we have seen with natural numbers and integers. However, if the target theory encompasses exactly the source theory, then the translation introduces unnecessary predicates, as we have seen with sets and pointed graphs.

Practical application. The $\lambda\Pi$-calculus modulo theory has been implemented in the DEDUKTI proof language and in the LAMBDAPI proof assistant. Future work would be to implement this interpretation in DEDUKTI. It would allow *effective* proof transfers between different DEDUKTI theories, and would therefore strengthen the interoperability between proof assistants via DEDUKTI.

Theoretical application. Dowek and Miquel [12] developed a method for interpreting theories of first-order logic. They showed that this interpretation can be used to prove a relative normalization result for theories in deduction modulo theory [10], that is first-order logic extended with user-defined rewrite rules. An application of this paper would be to prove a relative normalization result for the $\lambda\Pi$-calculus modulo theory. We would therefore be able to show that the encoding of set theory via pointed graphs in the $\lambda\Pi$-calculus modulo theory [5] satisfies a relative normalization result, just like this encoding in deduction modulo theory [11] does.

Acknowledgments

The author is grateful to Valentin Blot, Gilles Dowek and Théo Winterhalter for their insightful feedback on this work, and thanks the reviewers for their relevant comments.

References

[1] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant & Ronan Saillard (2016): *Dedukti: a Logical Framework based on the λΠ-Calculus Modulo Theory*. Manuscript.

[2] Jean-Philippe Bernardy, Patrik Jansson & Ross Paterson (2010): *Parametricity and dependent types*. In: *ICFP 2010 - 15th ACM SIGPLAN International Conference on Functional Programming*, Association for Computing Machinery, Baltimore, USA, p. 345–356, doi:10.1145/1863543.1863592.

[3] Jean-Philippe Bernardy, Patrik Jansson & Ross Paterson (2012): *Proofs for free: Parametricity for dependent types*. *Journal of Functional Programming* 22(2), p. 107–152, doi:10.1017/S0956796812000056.

[4] Frédéric Blanqui, Gilles Dowek, Emilie Grienenberger, Gabriel Hondet & François Thiré (2023): *A modular construction of type theories*. *Logical Methods in Computer Science* Volume 19, Issue 1, doi:10.46298/lmcs-19(1:12)2023. Available at https://lmcs.episciences.org/10959.

[5] Valentin Blot, Gilles Dowek & Thomas Traversié (2022): *An Implementation of Set Theory with Pointed Graphs in Dedukti*. In: *LFMTP 2022 - International Workshop on Logical Frameworks and Meta-Languages : Theory and Practice*, Haïfa, Israel. Available at https://inria.hal.science/hal-03740004.

[6] Valentin Blot, Gilles Dowek, Thomas Traversié & Théo Winterhalter (2024): *From Rewrite Rules to Axioms in the λΠ-Calculus Modulo Theory*. In: *FoSSaCS 2024 - 27th International Conference on Foundations of Software Science and Computation Structures*, Springer Nature Switzerland, Luxembourg, Luxembourg, pp. 3–23, doi:10.1007/978-3-031-57231-9_1.

[7] Cyril Cohen, Enzo Crance & Assia Mahboubi (2024): *Trocq: Proof Transfer for Free, With or Without Univalence*. In: *ESOP 2024 - 33rd European Symposium on Programming*, Springer Nature Switzerland, Luxembourg, Luxembourg, pp. 239–268, doi:10.1007/978-3-031-57262-3_10.

[8] Denis Cousineau & Gilles Dowek (2007): *Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo*. In: *TLCA 2007 - 8th International Conference on Typed Lambda Calculi and Applications*, Springer Berlin Heidelberg, Paris, France, pp. 102–117, doi:10.1007/978-3-540-73228-0_9.

[9] Nachum Dershowitz & Jean-Pierre Jouannaud (1991): *Rewrite Systems*. In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, doi:10.1016/B978-0-444-88074-1.50011-1.

[10] Gilles Dowek, Thérèse Hardin & Claude Kirchner (2003): *Theorem Proving Modulo*. *Journal of Automated Reasoning* 31, pp. 33–72, doi:10.1023/A:1027357912519.

[11] Gilles Dowek & Alexandre Miquel (2007): *Cut elimination for Zermelo set theory*. Manuscript.

[12] Gilles Dowek & Alexandre Miquel (2007): *Relative normalization*. Available at https://arxiv.org/abs/2310.20248. Manuscript.

[13] Robert Harper, Furio Honsell & Gordon Plotkin (1993): *A Framework for Defining Logics*. *Journal of the ACM* 40(1), p. 143–184, doi:10.1145/138027.138060.

[14] Gabriel Hondet & Frédéric Blanqui (2020): *The New Rewriting Engine of Dedukti*. In: *FSCD 2020 - 5th International Conference on Formal Structures for Computation and Deduction*, 167, Paris, France, p. 16, doi:10.4230/LIPIcs.FSCD.2020.35. Available at https://inria.hal.science/hal-02981561.

[15] Chantal Keller & Marc Lasson (2012): *Parametricity in an Impredicative Sort*. In: *CSL 2012 - 26th EACSL Annual Conference on Computer Science Logic*, 16, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Fontainebleau, France, pp. 381–395, doi:10.4230/LIPIcs.CSL.2012.381. Available at https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2012.381.

[16] John C. Reynolds (1983): *Types, Abstraction and Parametric Polymorphism*. In: *Information Processing 83 - IFIP 9th World Computer Congress*, North-Holland/IFIP, Paris, France, pp. 513–523.

[17] François Thiré (2020): *Interoperability between proof systems using the logical framework Dedukti*. Ph.D. thesis, Université Paris-Saclay. Available at `https://hal.science/tel-03224039`.

[18] Philip Wadler (1989): *Theorems for free!* In: *FPCA 1989 - 4th International Conference on Functional Programming Languages and Computer Architecture*, Association for Computing Machinery, New York, USA, p. 347–359, doi:`10.1145/99370.99404`.