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

Thomas Traversié

To cite this version:
Thomas Traversié. Proofs for Free in the $\lambda\Pi$-Calculus Modulo Theory. LFMTP 2024 - International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, Jul 2024, Tallinn, Estonia. pp.49 - 63, 10.4204/eptcs.404.4 . hal-04646198

HAL Id: hal-04646198
https://hal.science/hal-04646198
Submitted on 12 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
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 $\lambda$-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 re-usability, re-checking and preservation of libraries. For this purpose, Cousineau and Dowek developed the $\lambda\Pi$-calculus modulo theory [8], that combines $\lambda$-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.
Proofs for Free in the \(\lambda\Pi\)-Calculus Modulo Theory

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 \(\lambda\Pi\)-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 \(\lambda\Pi\)-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 \(S\) into the target theory \(T\). 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 \(S\) has an interpretation in \(T\), then the proofs written inside \(S\) can be transformed into proofs written inside \(T\). This interpretation comes with a relative consistency theorem: if \(T\) is consistent, then \(S\) 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 DEUKTI, and are available at [https://github.com/thomastraversie/InterpDK](https://github.com/thomastraversie/InterpDK).

**Outline of the paper.** In Section 2 we give a formal presentation of the \(\lambda\Pi\)-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 \(\lambda\Pi\)-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 \(\lambda\Pi\)-Calculus Modulo Theory

In this section, we give a formal definition of the syntax and type system of the \(\lambda\Pi\)-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 \(\lambda\Pi\)-Calculus Modulo Theory

The Edinburgh Logical Framework \([13]\), also known as \(\lambda\Pi\)-calculus, is an extension of simply typed \(\lambda\)-calculus with dependent types. The \(\lambda\Pi\)-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**  
  \[ s ::= \text{TYPE} \mid \text{KIND} \]

- **Terms**  
  \[ t,u,A,B ::= c \mid x \mid s \mid \Pi(x:A).B \mid \lambda(x:A).t \mid tu \]

- **Contexts**  
  \[ \Gamma ::= \langle \rangle \mid \Gamma,x:A \]

- **Signatures**  
  \[ \Sigma ::= \langle \rangle \mid \Sigma,c:A \mid \Sigma,\ell \leftrightarrow r \]

where \(c\) is a constant and \(x\) is a variable (ranging over disjoint sets), \(\Pi(x:A).B\) is a dependent product (simply written \(A \rightarrow B\) if \(x\) does not occur in \(B\)), \(\lambda(x:A).t\) is an abstraction, and \(tu\) is an application. For
The relation \( \to_{\beta} \) is a term with no free variables) and rewrite rules of the relation \( \to \) then \( \Gamma \)

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

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 \(\lambda\Pi\)-calculus modulo theory. This encoding is defined by the following signature, written \(\Sigma_{pre}\).

\[
\begin{align*}
Set &: \text{TYPE} \\
El &: \text{Set} \to \text{TYPE} \\
\rightsquigarrow_d &: \Pi(x : \text{Set}). (El x \to \text{Set}) \to \text{Set} \\
El(x \rightsquigarrow_d y) &: \Pi(z : El x). El(y z) \\
p &: \Pi(x : El o). (Prf x \to \text{Set}) \to \text{Set} \\
El(p x y) &: \Pi(z : Prf x). El(y z) \\
\Rightarrow_d &: \Pi(x : El o). (Prf x \to El o) \to El o \\
\forall &: \Pi(x : \text{Set}). (El x \to El o) \to El o
\end{align*}
\]

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 \(p\) (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 \(\text{TYPE}\) in the \(\lambda\Pi\)-calculus modulo theory, this encoding allows to quantify over propositions—objects of type \(El o\)—and then inject them into \(\text{TYPE}\) using \(Prf\). Similarly, we can quantify over sorts—objects of type \(Set\)—and then inject them into \(\text{TYPE}\) 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 \(T = \Sigma_{pre} \cup \Sigma_T\), where the user-defined constants \(c : A \in \Sigma_T\) have to be expressed in the prelude encoding.

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

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

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

**Example 1** (Natural numbers). We define a theory with prelude encoding \(\Sigma_n = \Sigma_{pre} \cup \Sigma_n\) for natural numbers. \(\text{nat}\) is the sort of natural numbers. We declare two constructors \(0_n\) and \(\text{succ}_n\), a relation \(\geq_n\).
and an induction principle \( \text{rec}_n \).

\[
\begin{align*}
\text{nat} & : \quad \text{Set} \\
0_n & : \quad \text{El} \text{ nat} \\
\text{succ}_n & : \quad \text{El} \text{ nat} \rightarrow \text{El} \text{ nat} \\
\geq_n & : \quad \text{El} \text{ nat} \rightarrow \text{El} \text{ nat} \rightarrow \text{El} \text{ o} \\
\text{ax}_0^i & : \quad \Pi(x : \text{El} \text{ nat}). \ \text{Prf} (x \geq_{\text{n}} x) \\
\text{ax}_0^\geq & : \quad \Pi(x : \text{El} \text{ nat}). \ \text{Prf} (\text{succ}_n x \geq_{\text{n}} x) \\
\text{ax}_1^i & : \quad \Pi(x,y,z : \text{El} \text{ nat}). \ \text{Prf} (x \geq_{\text{n}} y) \rightarrow \text{Prf} (y \geq_{\text{n}} z) \rightarrow \text{Prf} (x \geq_{\text{n}} z) \\
\text{ax}_1^\geq & : \quad \Pi(P : \text{El} \text{ nat} \rightarrow \text{El} \text{ o}). \ \text{Prf} (P 0_n) \rightarrow \text{Prf} (P (\text{succ}_n x)) \\
\text{rec}_n & : \quad \Pi(P : \text{El} \text{ nat} \rightarrow \text{El} \text{ o}). \ \text{Prf} (P x) \rightarrow \text{Prf} (P (\text{succ}_n x)) \\
\end{align*}
\]

In this theory, we can prove \( \Pi(x : \text{El} \text{ nat}). \ \text{Prf} (x \geq_{\text{n}} 0_n) \) and \( \Pi(x : \text{El} \text{ nat}). \ \text{Prf} (\text{succ}_n x \geq_{\text{n}} 0_n) \).

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

\[
\begin{align*}
\text{int} & : \quad \text{Set} \\
0_i & : \quad \text{El} \text{ int} \\
\text{succ}_i & : \quad \text{El} \text{ int} \rightarrow \text{El} \text{ int} \\
\text{pred}_i & : \quad \text{El} \text{ int} \rightarrow \text{El} \text{ int} \rightarrow \text{El} \text{ o} \\
\geq_i & : \quad \text{El} \text{ int} \rightarrow \text{El} \text{ int} \rightarrow \text{El} \text{ o} \\
\text{ax}_0^i & : \quad \Pi(x : \text{El} \text{ int}). \ \text{Prf} (x \geq_i x) \\
\text{ax}_0^\geq & : \quad \Pi(x : \text{El} \text{ int}). \ \text{Prf} (\text{succ}_i x \geq_i x) \\
\text{ax}_1^i & : \quad \Pi(x,y,z : \text{El} \text{ int}). \ \text{Prf} (x \geq_i y) \rightarrow \text{Prf} (y \geq_i z) \rightarrow \text{Prf} (x \geq_i z) \\
\text{ax}_1^\geq & : \quad \Pi(P : \text{El} \text{ int} \rightarrow \text{El} \text{ o}). \ \text{Prf} (P (\text{succ}_i x)) \rightarrow \text{Prf} (P x) \\
\text{rec}_i & : \quad \Pi(P : \text{El} \text{ int} \rightarrow \text{El} \text{ o}). \ \text{Prf} (P c) \rightarrow \text{Prf} (P x) \\
\end{align*}
\]

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

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

An alternative is to interpret the type of natural numbers $\text{El} \text{ nat}$ by the type of integers $\text{El} \text{ int}$, 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 $0_i$, $\text{succ}_n$ by $\text{succ}_i$, and $\geq_n$ by $\geq_i$. The interpretation of the theorem $\Pi(x : \text{El} \text{ nat}). \text{Prf}(\text{succ}_n x \geq_n 0_n)$ should not be $\Pi(x^* : \text{El} \text{ int}). \text{Prf}(\text{succ}_i x^* \geq_i 0_i)$, 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 $\text{Prf}(x^* \geq_i 0_i)$. Thus the interpretation of the theorem $\Pi(x : \text{El} \text{ nat}). \text{Prf}(\text{succ}_n x \geq_n 0_n)$ should be $\Pi(x^* : \text{El} \text{ int}). \text{Prf}(x^* \geq_i 0_i) \rightarrow \text{Prf}(\text{succ}_i x^* \geq_i 0_i)$.

Formal definition. Following this intuition, when interpreting a term $t$ of type $A$ in $S$ by a term $t^*$ of type $A^*$ in $T$, 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 $\lambda(x^* : A^*)(x^* : A^+ x^*). t^* \rightarrow (\lambda(x : A). t)^+$ 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 TYPE, we take ($\Pi(x : A). B)^+ := \lambda(f : (\Pi(x : A). B)^*). \Pi(x^* : A^*)(x^* : A^+ x^*). B^+ (f x^* x^*)$. However, we cannot do the same when $B$ is of type KIND, because this term would be ill-typed. Indeed, ($\Pi(x : A). B)^+$ has type KIND, while the type of the bound variable $f$ must have type 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 KIND, we take ($\Pi(x : A). B)^+\{X\} := \Pi(x^* : A^*)(x^* : A^+ x^*). B^+\{X x^* x^+\}$.

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

- $(x)^* := x^*$ (variable)
- $(c)^* := c^*$ (parameter)
- TYPE$^* := $TYPE
- KIND$^* := $KIND
- $(t u)^* := t^* u^* u^+$
- $(\lambda(x : A). t)^* := \lambda(x^* : A^*)(x^* : A^+ x^*). t^*$
- $(\Pi(x : A). B)^* := \Pi(x^* : A^*)(x^* : A^+ x^*). B^*$
Using parametricity, the translation $(\Prf p)^+ := x^+$ (variable) 
$(c)^+ := c^+$ (parameter) 
$\text{TYPE}^+\{X\} := X \rightarrow \text{TYPE}$ 
$\text{KIND}^+ := \text{KIND}$ 
$(t u)^+ := t^+ u^+ x^+$ 
$(\lambda (x : A). t)^+ := \lambda (x^+ : A^+)(x^+ : A^+ x^+). t^+$ 
$(\Pi (x : A). B)^+ := \lambda (f : (\Pi (x : A). B)^+). (\Pi (x^+ : A^+ x^+). B^+ (f x^+ x^+))$ if $B : \text{TYPE}$ 
$(\Pi (x : A). B)^+ \{X\} := (\Pi (x^+ : A^+ x^+). B^+ \{X x^+\})$ if $B : \text{KIND}.$

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

$$\langle \rangle^{*+} := \langle \rangle$$  
$$\langle \Gamma, x : A \rangle^{*+} := \Gamma^{*+}, x^+ : A^+ x^+.$$

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 $\Gamma^*$ and $\Gamma^+$, 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 $t^+ u^+ x^+$, 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 $\Sigma$-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 $\Sigma$-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 $t^+ u^+ x^+$. 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 $S$ in a target theory $T$, when $S$ and $T$ are theories with prelude encoding. Such an interpretation is parametrized by the terms of $T$ that correspond to the constants of $S$. In particular, we have to provide the parameters for the constants of the prelude encoding.

When $\vdash t : A$ in $S$, we want to have $\vdash t^* : A^*$ in $T$. Moreover, we want $\vdash A^+ : A^+ \rightarrow \text{TYPE}$ in $T$ when $A = \text{TYPE}$. These conditions lead to the definition of $\text{Set}^+$, $\text{Set}^+$, $\text{El}^+$, $\text{El}^+$, $\Prf^+$, $\Prf^+$ 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 $\text{El} o$ is indeed a proposition, so we choose $o^+$ to be $\lambda (z : \text{El} o). z \Rightarrow (\lambda (x : \Prf z). z)$. Consequently, it is is always possible to find a witness $p^+$ of type
$\text{Prf} \ (a^+ \ p^*)$, that is $\text{Prf} \ p^* \rightarrow \text{Prf} \ p^*$.

$\text{Set}^* := \text{Set}$

$\text{Set}^+ := \lambda(x : \text{Set}). \text{El} \ x \rightarrow \text{El} \ o$

$a^* := \lambda(z : \text{El} \ o). \ z \Rightarrow_d (\lambda(x : \text{Prf} \ z). \ z)$

$\text{El}^* := \lambda(x^* : \text{Set})(x^* : \text{El} \ x^* \rightarrow \text{El} \ o). \text{El} \ x^*$

$\text{El}^+ := \lambda(u^* : \text{Set})(u^* : \text{El} \ u^* \rightarrow \text{El} \ o)(x : \text{El} \ u^*). \text{Prf} \ (u^+ x)$

$\text{Prf}^* := \lambda(x^* : \text{El} \ o)(x^* : \text{Prf} \ (a^+ x^*)). \text{Prf} \ x^*$

$\text{Prf}^+ := \lambda(u^* : \text{El} \ o)(u^* : \text{Prf} \ (o^+ u^*))(x : \text{Prf} \ u^*). \text{Prf} \ u^*$

Parameters $\sim^*_d$ and $\sim^+_d$ are defined so that $(\text{El} \ (a \sim_d b))^{\oplus} \equiv_{\beta \Sigma} (\Pi(x : \text{El} \ a). \text{El} \ (b \ x))^{\oplus}$ for @ ∈ {*, +}.

$\sim^*_d := \lambda(a^* : \text{Set})(a^* : \text{El} \ a^* \rightarrow \text{El} \ o)(b^* : \Pi(x^* : \text{El} \ a^*). \text{Prf} \ (a^+ x^*) \rightarrow \text{Set}).$

$\sim^+_d := \lambda(a^* : \text{Set})(a^* : \text{El} \ a^* \rightarrow \text{El} \ o)(b^* : \Pi(x^* : \text{El} \ a^*). \text{Prf} \ (a^+ x^*) \rightarrow \text{Set}).$

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

$\Rightarrow^*_d := \lambda(a^* : \text{El} \ o)(a^* : \text{Prf} \ (o^+ a^*))(b^* : \Pi(x^* : \text{Prf} \ a^*). \text{Prf} \ a^* \rightarrow \text{El} \ o)$

$\Rightarrow^+_d := \lambda(a^* : \text{El} \ o)(a^* : \text{Prf} \ (o^+ a^*))(b^* : \Pi(x^* : \text{Prf} \ a^*). \text{Prf} \ a^* \rightarrow \text{El} \ o)$

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

$\pi^* := \lambda(a^* : \text{El} \ o)(a^* : \text{Prf} \ (o^+ a^*))(b^* : \Pi(x^* : \text{Prf} \ a^*). \text{Prf} \ a^* \rightarrow \text{Set}).$

$\pi^+ := \lambda(a^* : \text{El} \ o)(a^* : \text{Prf} \ (o^+ a^*))(b^* : \Pi(x^* : \text{Prf} \ a^*). \text{Prf} \ a^* \rightarrow \text{Set}).$

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

$\forall^* := \lambda(a^* : \text{Set})(a^* : \text{El} \ a^* \rightarrow \text{El} \ o)(b^* : \Pi(x^* : \text{El} \ a^*). \text{Prf} \ (a^+ x^*) \rightarrow \text{El} \ o)$

$\forall^+ := \lambda(a^* : \text{Set})(a^* : \text{El} \ a^* \rightarrow \text{El} \ o)(b^* : \Pi(x^* : \text{El} \ a^*). \text{Prf} \ (a^+ x^*) \rightarrow \text{El} \ o)$.

$\forall^+ := \lambda(a^* : \text{Set})(a^* : \text{El} \ a^* \rightarrow \text{El} \ o)(b^* : \Pi(x^* : \text{El} \ a^*). \text{Prf} \ (a^+ x^*) \rightarrow \text{El} \ o)$.

$\forall^+ := \lambda(a^* : \text{Set})(a^* : \text{El} \ a^* \rightarrow \text{El} \ o)(b^* : \Pi(x^* : \text{El} \ a^*). \text{Prf} \ (a^+ x^*) \rightarrow \text{El} \ o)$.

$\forall^+ := \lambda(a^* : \text{Set})(a^* : \text{El} \ a^* \rightarrow \text{El} \ o)(b^* : \Pi(x^* : \text{El} \ a^*). \text{Prf} \ (a^+ x^*) \rightarrow \text{El} \ o)$.
∀⁺ := \( \lambda (a^* : \text{Set})(a^+ : \text{El } a^*) \to \text{El } o)(b^* : \Pi(x^* : \text{El } a^*) \to \text{Prf } (a^+ x^*) \to \text{El } o).\)
\( \lambda (b^* : \Pi(x^* : \text{El } a^*)\langle x^* : \text{Prf } (a^* x^*)\rangle) \to \text{Prf } (a^+ (b^* x^* x^+)\langle x^* : \text{Prf } (a^* x^*)\rangle)).\)
\( \lambda (p : \text{Prf } (\forall a b^*)\langle x^* : \text{Prf } (a^* x^*)\rangle, p) \)

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

**Proposition 1.** Let \( c : A \in \Sigma_{\text{pre}} \).

1. We have \( \vdash c^* : A^* \).

2. (a) If \( \vdash A : \text{TYPE} \) then \( \vdash c^+ : A^+ c^* \).

(b) If \( \vdash A : \text{KIND} \) 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 \texttt{lo\_sp\_dk}. \( \square \)

**Proposition 2.** For every \( \ell \leftrightarrow r \in \Sigma_{\text{pre}} \), we have \( \ell^* \equiv_{\beta \Sigma} r^* \) and \( \ell^+ \equiv_{\beta \Sigma} r^+ \).

**Proof.** We only show the case \( \text{El } (a \rightsquigarrow_d b) \leftrightarrow \Pi(x : \text{El } a). \text{El } (b x) \).

We have \( (\text{El } (a \rightsquigarrow_d b))^* \equiv_{\beta \Sigma} \text{El } (a \rightsquigarrow_d b)^* \)
\( \equiv_{\beta \Sigma} \text{El } (a^* \rightsquigarrow_d (\lambda x^*. \pi (a^* x^*) (b^* x^+))) \)
\( \equiv_{\beta \Sigma} \Pi(x^* : \text{El } a^*). \text{El } (\pi (a^* x^*) (b^* x^+)) \)
\( \equiv_{\beta \Sigma} \Pi(x^* : \text{El } a^*) (x^+ : \text{Prf } (a^+ x^+)). \text{El } (b^* x^* x^+) \)
\( \equiv_{\beta \Sigma} \Pi(x^* : (\text{El } a)^*) (x^+ : (\text{El } a)^+ x^+). (\text{El } (b x))^* \)
\( \equiv_{\beta \Sigma} (\Pi(x : \text{El } a). \text{El } (b x))^* \)

and \( (\text{El } (a \rightsquigarrow_d b))^+ \equiv_{\beta \Sigma} \lambda(f : \text{El } (a \rightsquigarrow_d b)^*). \text{Prf } ((a \rightsquigarrow_d b)^+ f) \)
\( \equiv_{\beta \Sigma} \lambda(f : \text{El } (a \rightsquigarrow_d b)^*), \text{Prf } (\forall a^* (\lambda x^*. (a^* x^*) \Rightarrow_d (\lambda x^+. b^+ x^* x^+ (f x^* x^+)))) \)
\( \equiv_{\beta \Sigma} \lambda(f : \text{El } (a \rightsquigarrow_d b)^*). \Pi(x^* : \text{El } a^*), \text{Prf } ((a^+ x^+) \Rightarrow_d (\lambda x^+. b^+ x^* x^+ (f x^* x^+))) \)
\( \equiv_{\beta \Sigma} \lambda(f : (\text{El } (a \rightsquigarrow_d b)^*), \Pi(x^* : \text{El } a^*) (x^+ : \text{Prf } (a^+ x^+)). \text{Prf } (b^+ x^* x^+ (f x^* x^+)) \)
\( \equiv_{\beta \Sigma} \lambda(f : (\Pi(x : \text{El } a). \text{El } (b x))^*). \Pi(x^* : (\text{El } a)^*) (x^+ : (\text{El } a)^+ x^+). (\text{El } (b x))^+ (f x^* x^+) \)
\( \equiv_{\beta \Sigma} (\Pi(x : \text{El } a). \text{El } (b x))^+ . \)

The result has been checked in DEDUKTI for the four rewrite rules, see the \texttt{#ASSERT} commands in the file \texttt{lo\_sp\_dk}. \( \square \)

### 3.3 Interpretation of Theories

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

---

1All the DEDUKTI files are available at [https://github.com/thomastraversie/InterpDK](https://github.com/thomastraversie/InterpDK)
Definition 4 (Interpretation of theories). Let $S$ and $T$ be two theories with prelude encoding. We say that $S$ has an interpretation in $T$ when:

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

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

If we cannot interpret the rewrite rules of $S$ into conversions in $T$, we can nonetheless replace the rewrite rules of $S$ by equational axioms—that is by typed constants—and then interpret such constants in $T$. 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 $\lambda\Pi$-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 $S$, then $A^* \equiv_{\beta\Sigma} B^*$ and $A^+ \equiv_{\beta\Sigma} B^+$ in $T$.

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

- We have $(\lambda(x : A). t) u^* = (\lambda(x^* : A^*) (x^+ : A^+ x^*). t^*) u^+ \equiv_{\beta\Sigma} t^*[x^* \leftarrow u^*][x^+ \leftarrow u^+]$, which $\beta$-reduces to $t^*[x^* \leftarrow u^*][x^+ \leftarrow u^+]$, that is $(t[x \leftarrow u])^*$ following Proposition 5. Similarly, $((\lambda(x : A). t) u)^+ \equiv_{\beta\Sigma} (t[x \leftarrow u])^+$.
- For each $\ell \rightsquigarrow r \in S$ and any substitution $\theta$, we have $\ell^* \equiv_{\beta\Sigma} r^*$ by definition and Proposition 2. Using Proposition 3, we have $(\ell^\theta)^* = \ell^+ \theta^+ \equiv_{\beta\Sigma} (r^\theta)^+$.
- For closure by context, we only show the $\lambda$-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 $S$ has an interpretation in $T$, any provable judgment in $S$ is interpreted as a provable judgment in $T$. 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 $\mathcal{S}$ and $\mathcal{T}$ be two theories with prelude encoding, such that $\mathcal{S}$ has an interpretation in $\mathcal{T}$.

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

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

- **CONST**: By induction, we have $\Gamma \vdash \Gamma^* and $\Gamma \vdash A : s^*$. Since $c : A \in \mathcal{S}$, we have $\Gamma \vdash c^* : A^*$. We derive $\Gamma \vdash c^* : A^*$ by weakening. If $s \in TYPE$, then $\Gamma \vdash c^* : A^* c^*$ and we derive $\Gamma \vdash c^* : A^* c^*$ by weakening. If $s \in KIND$, then $\Gamma \vdash c^* : A^* \{c^*\}$ and we derive $\Gamma \vdash c^* : A^* \{c^*\}$ by weakening.

- **PROD**: By induction, we have $\Gamma \vdash \Gamma^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. Using PROD, we get $\Gamma \vdash \Gamma^* \vdash \Pi(x^* : A^*)(x^* : A^* x^*)$. $B^* : s^*$.

Suppose that $s \in TYPE$. By induction, $\Gamma \vdash \Gamma^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. Since $\Gamma \vdash \Gamma^* \vdash \lambda(f : (\Pi(x : A) B))(f(x^* x^*))$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$, we derive $\Gamma \vdash \Gamma^* \vdash (\Pi(x^* : A^*) \vdash \Gamma \vdash A^* x^*)$. $B^* : (f(x^* x^*))$ and $\Pi(x^* : A^*) \vdash \Gamma \vdash A^* \{t^*\}$. $\Gamma \vdash \Gamma^* : TYPE$, which corresponds to $\Gamma \vdash \Gamma \vdash A^* \{t^*\}$.

Suppose that $s \in KIND$ and that we have $\Gamma \vdash \Gamma^* \vdash t : (\Pi(x : A) B)^*$. Since $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$, we get $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. Using PROD, we conclude that $\Gamma \vdash \Gamma^* \vdash (\Pi(x : A) B)^* \vdash \Gamma \vdash A^* \{t^*\}$.

- **ABS**: By induction, we have $\Gamma \vdash \Gamma^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. We derive $\Gamma \vdash \Gamma^* : \Gamma \vdash (\Pi(x^* : A^*) (x^* : A^* x^*) x^*) : \Gamma \vdash A^* x^*)$. $B^*$, that is $\Gamma \vdash \Gamma^* : \Gamma \vdash (\Pi(x : A) B)^*$.

Suppose that $s \in TYPE$. By induction, we have $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. We derive $\Gamma \vdash \Gamma^* : \Gamma \vdash (\Pi(x^* : A^*) (x^* : A^* x^*) x^*) : \Gamma \vdash A^* x^*)$. $B^*$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. Using PROD, we conclude that $\Gamma \vdash \Gamma^* \vdash (\Pi(x : A) B)^* \vdash \Gamma \vdash A^* \{t^*\}$.

Suppose that $s \in KIND$. By induction, we have $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. We derive $\Gamma \vdash \Gamma^* : \Gamma \vdash (\Pi(x^* : A^*) (x^* : A^* x^*) x^*) : \Gamma \vdash A^* x^*)$. $B^*$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. Using PROD, we conclude that $\Gamma \vdash \Gamma^* \vdash (\Pi(x : A) B)^* \vdash \Gamma \vdash A^* \{t^*\}$.

- **APP**: By induction, we have $\Gamma \vdash \Gamma^* \vdash t^* : \Gamma \vdash A^* (x^* : A^* x^*)$. $B^*$, and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. Using PROD, we conclude that $\Gamma \vdash \Gamma^* \vdash (t u^* : B[x^* u])$. $x^* u^*$. Using Proposition 2 we conclude that $\Gamma \vdash \Gamma^* \vdash (t u^* : B[x^* u])$.

Suppose that $\Gamma \vdash \Pi(x : A) B : TYPE$ (and thus $\Gamma \vdash B : TYPE$). By induction, we have $\Gamma \vdash \Gamma^* \vdash t^* : \Pi(x^* : A^*) (x^* : A^* x^*)$. $B^*$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$ and $\Gamma \vdash \Gamma^* : \Gamma \vdash A^* : TYPE$. Using Proposition 3 we conclude that $\Gamma \vdash \Gamma^* \vdash (t u^* : B[x^* u])$. $x^* u^*$. Using Proposition 4 we conclude that $\Gamma \vdash \Gamma^* \vdash (t u^* : B[x^* u])$.

- **CONV**: We conclude using the induction hypotheses and Proposition 4.
Given an interpretation of a source theory \( S \) in a target theory \( T \), the results proved in \( S \) are automatically transported to \( T \). The interpretation of \( S \) in \( T \) only requires the parameters \( c^i \) and \( c^+ \) in \( T \) for each user-defined constant \( c \) of \( S \). Once we have an interpretation of \( S \) in \( T \), it is possible to prove that \( S \) is consistent provided that \( T \) is so. In the \( \lambda \Pi \)-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 \( S \) and \( T \) be two theories with prelude encoding, such that \( S \) has an interpretation in \( T \). If \( T \) is consistent, then \( S \) is consistent too.

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

### 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 \( \text{nat}^+ := \text{int} \). An integer is a non-negative natural number, so the predicate asserting that an integer is a natural number is defined by \( \text{nat}^+ := \lambda z. z \geq 0_i \). The interpretation of \( 0_n \) is given by \( 0^+_n := 0_i \), and we choose \( 0^+_n := a x^+_1 \) for the proof of \( 0^+_n \geq 1_0 \). We take \( \text{succ}_n^+ := \lambda x^+. \text{succ}_i x^+ \) and \( \text{succ}_n^+ := \lambda x^+. \lambda x^+. \text{succ}_i x^+ \). For the interpretation of \( \geq_n \), we choose \( \geq^+_n := \lambda y^+. \lambda x^+. \lambda y^+. x^+ \geq_i y^+ \). Given that \( \geq_n \) returns a proposition, the parameter \( \geq^+_n \) must have type \( \Pi x^+ \). \( \Pi x^+ \). \( \Pi y^+ \). \( \Prf (x^+ \geq_i y^+) \rightarrow \Prf (x^+ \geq_i y^+) \) has an immediate inhabitant. The interpretation of \( a x^+_1 \) is given by \( (a x^+_1)^+ := \lambda x^+. \lambda x^+. a x^+_1 : x^+ \). Since \( a x^+_1 \) returns a proof, and by definition of \( \Prf^+ \), both \( (a x^+_1)^+ \) and \( (a x^+_1)^+ \) have the same type, so we can take \( (a x^+_1)^+ := (a x^+_1)^+ \). The parameters for \( a x^+_2 \) and \( a x^+_3 \) are chosen correspondingly.

When defining the parameter \( \text{rec}_n^+ \), we assume \( P^+ \) of type \( \Pi (x^+ : \text{El nat}^+). \Prf (x^+ \geq_i 0_i) \rightarrow \text{El o} \). We must apply \( \text{rec}_i \) to a predicate of type \( \text{El nat}^+ \rightarrow \text{El o} \), 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.

\[
\text{proof}_\text{irr} : \Pi (p : \text{El o})(h h' : \Prf p)(Q : \Prf p \rightarrow \text{El o}). \Prf (Q h) \rightarrow \Prf (Q h')
\]

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 theories of \( \mathbb{T}_n \) in \( \mathbb{T}_i \). For instance, we can show in \( \mathbb{T}_n \) that \( \vdash \text{thm : } \Pi (x : \text{El int}). \Prf (\text{succ}_n x \geq_i 0_n) \), where \( \text{thm} \) is a proof that uses \( \text{rec}_n, a x^+_1, a x^+_2 \) and \( a x^+_3 \). The interpretation of \( \mathbb{T}_n \) in \( \mathbb{T}_i \) allows us to directly derive \( \vdash \text{thm}^+ : \Pi (x^+ : \text{El int}). \Prf (x^+ \geq_i 0_i) \rightarrow \Prf (\text{succ}_i x^+ \geq_i 0_i) \) in \( \mathbb{T}_i \).

The complete interpretation of natural numbers into integers has been formalized in DEDUKTI, and is available in the file \text{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 $\text{eta} : \text{El graph} \to \text{El node} \to \text{El node} \to \text{El o}$ is such that $\text{eta} a x y$ is the proposition asserting that there is an edge in pointed graph $a$ from node $y$ to node $x$. The operator $\text{root} : \text{El graph} \to \text{El node}$ returns the root of a pointed graph, and $\text{cr} : \text{El graph} \to \text{El node} \to \text{El graph}$ is such that $\text{cr} a x$ corresponds to the pointed graph $a$ in which the root is now at node $x$.

The different constructors on sets—unions, pairs, power sets 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 $\text{El graph}$ 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 $\mathcal{S}$ has an interpretation in a target theory $\mathcal{T}$, then the theorems proved in $\mathcal{S}$ come for free in $\mathcal{T}$. At the end, we obtain a relative consistency result, establishing that the consistency of the theory $\mathcal{T}$ entails the consistency of the theory $\mathcal{S}$.

This interpretation applies when $\mathcal{S}$ can be embedded into $\mathcal{T}$. In particular, we allow the interpretation of a type $A$ of $\mathcal{S}$ by a more general type $A^*$ of $\mathcal{T}$. As a consequence, we ensure that, for every term $t$ of type $A$ in $\mathcal{S}$, its interpretation $t^*$ of type $A^*$ in $\mathcal{T}$ 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


