Embedding Pure Type Systems in the lambda-Pi-calculus modulo

Denis Cousineau and Gilles Dowek

École polytechnique and INRIA
LIX, École polytechnique, 91128 Palaiseau Cedex, France.

Cousineau@lix.polytechnique.fr, http://www.lix.polytechnique.fr/~cousineau
Gilles.Dowek@polytechnique.edu, http://www.lix.polytechnique.fr/~dowek

Abstract. The lambda-Pi-calculus allows to express proofs of minimal predicate logic. It can be extended, in a very simple way, by adding computation rules. This leads to the lambda-Pi-calculus modulo. We show in this paper that this simple extension is surprisingly expressive and, in particular, that all functional Pure Type Systems, such as the system F, or the Calculus of Constructions, can be embedded in it. And, moreover, that this embedding is conservative under termination hypothesis.

The *$\lambda\Pi$-calculus* is a dependently typed lambda-calculus that allows to express proofs of minimal predicate logic through the Brouwer-Heyting-Kolmogorov interpretation and the Curry-de Bruijn-Howard correspondence. It can be extended in several ways to express proofs of some theory. A first solution is to express the theory in Deduction modulo [7, 9], *i.e.* to orient the axioms as rewrite rules and to extend the $\lambda\Pi$-calculus to express proofs in Deduction modulo [3]. We get this way the *$\lambda\Pi$-calculus modulo*. This idea of extending the dependently typed lambda-calculus with rewrite rules is also that of Intuitionistic type theory used as a logical framework [13].

A second way to extend the $\lambda\Pi$-calculus is to add typing rules, in particular to allow polymorphic typing. We get this way the *Calculus of Constructions* [4] that allows to express proofs of simple type theory and more generally the *Pure Type Systems* [2, 15, 1]. These two kinds of extensions of the $\lambda\Pi$-calculus are somewhat redundant. For instance, simple type theory can be expressed in deduction modulo [8], hence the proofs of this theory can be expressed in the $\lambda\Pi$-calculus modulo. But they can also be expressed in the Calculus of Constructions. This suggests to relate and compare these two ways to extend the $\lambda\Pi$-calculus.

We show in this paper that all functional Pure Type Systems can be embedded in the $\lambda\Pi$-calculus modulo using an appropriate rewrite system. This rewrite system is inspired both by the expression of simple type theory in Deduction modulo and by the mechanisms of universes *à la* Tarski [12] of Intuitionistic type theory. In particular, this work extends Palmgren's construction of an impredicative universe in type theory [14].

1 The $\lambda\Pi$-calculus

The *$\lambda\Pi$-calculus* is a dependently typed lambda-calculus that permits to construct types depending on terms, for instance a type $array\ n$, of arrays of size n, that depends on a term n of type nat. It also permits to construct a function f taking a natural number n as an argument and returning an array of size n. Thus, the arrow type $nat \Rightarrow array$ of simply typed lambda-calculus must be extended to a dependent product type $\Pi x : nat\ (array\ x)$ where, in the expression $\Pi x : A\ B$, the occurrences of the variable x are bound in B by the symbol Π (the expression $A \Rightarrow B$ is used as a shorter notation for the expression $\Pi x : A\ B$ when x has no free occurrence in B). When we apply the function f to a term n, we do not get a term of type $array\ x$ but of type $array\ n$. Thus, the application rule must include a substitution of the term n for the variable x. The symbol $array$ itself takes a natural number as an argument and returns a type. Thus, its type is $nat \Rightarrow Type$, *i.e.* $\Pi x : nat\ Type$. The terms $Type$, $nat \Rightarrow Type$, ... cannot have type $Type$, because Girard's paradox [10] could then be expressed in the system, thus we introduce a new symbol $Kind$ to type such terms. To form terms, like $\Pi x : nat\ Type$, whose type is $Kind$, we need a rule expressing that the symbol $Type$ has type $Kind$ and a new product rule allowing to form the type $\Pi x : nat\ Type$, whose type is $Kind$. Besides the variables such as x whose type has type $Type$, we must permit the declaration of variables such as nat of type $Type$, and more generally, variables such as $array$ whose type has type $Kind$. This leads to introduce the following syntax and typing rules.

Definition 1 (The syntax of $\lambda\Pi$). *The syntax of the $\lambda\Pi$-calculus is*

$$t = x \mid Type \mid Kind \mid \Pi x : t\ t \mid \lambda x : t\ t \mid t\ t$$

The α-equivalence and β-reduction relations are defined as usual and terms are identified modulo α-equivalence.

Definition 2 (The typing rules of $\lambda\Pi^-$).

$$\frac{}{[\,]\ \textit{well-formed}}\ \textbf{Empty}$$

$$\frac{\Gamma \vdash A : Type}{\Gamma[x : A]\quad \textit{well-formed}}\ \textbf{Declaration}\ x\ \textit{not in}\ \Gamma$$

$$\frac{\Gamma \vdash A : Kind}{\Gamma[x : A]\quad \textit{well-formed}}\ \textbf{Declaration2}\ x\ \textit{not in}\ \Gamma$$

$$\frac{\Gamma\ \textit{well-formed}}{\Gamma \vdash Type : Kind}\ \textbf{Sort}$$

$$\frac{\Gamma\ \textit{well-formed}\quad x : A \in \Gamma}{\Gamma \vdash x : A}\ \textbf{Variable}$$

$$\frac{\Gamma \vdash A : Type\quad \Gamma[x : A] \vdash B : Type}{\Gamma \vdash \Pi x : A\ B : Type}\ \textbf{Product}$$

$$\frac{\Gamma \vdash A : Type \quad \Gamma[x:A] \vdash B : Kind}{\Gamma \vdash \Pi x : A\ B : Kind}\ \textbf{Product2}$$

$$\frac{\Gamma \vdash A : Type \quad \Gamma[x:A] \vdash B : Type \quad \Gamma[x:A] \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}\ \textbf{Abstraction}$$

$$\frac{\Gamma \vdash t : \Pi x : A\ B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : (u/x)B}\ \textbf{Application}$$

It is useful, in some situations, to add a rule allowing to build type families by abstraction, for instance $\lambda x : nat\ (array\ (2 \times x))$ and rules asserting that a term of type $(\lambda x : nat\ (array\ (2 \times x))\ n)$ also has type $array\ (2 \times n)$. This leads to introduce the following extra typing rules.

Definition 3 (The typing rules of $\lambda\Pi$). *The typing rules of $\lambda\Pi$ are those of $\lambda\Pi^-$ and*

$$\frac{\Gamma \vdash A : Type \quad \Gamma[x:A] \vdash B : Kind \quad \Gamma[x:A] \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}\ \textbf{Abstraction2}$$

$$\frac{\Gamma \vdash A : Type \quad \Gamma \vdash B : Type \quad \Gamma \vdash t : A}{\Gamma \vdash t : B}\ \textbf{Conversion}\ A \equiv_\beta B$$

$$\frac{\Gamma \vdash A : Kind \quad \Gamma \vdash B : Kind \quad \Gamma \vdash t : A}{\Gamma \vdash t : B}\ \textbf{Conversion2}\ A \equiv_\beta B$$

where $\equiv_\beta$ is the β-equivalence relation.

It can be proved that types are preserved by β-reduction, that β-reduction is confluent and strongly terminating and that each term has a unique type modulo β-equivalence.

The $\lambda\Pi$-calculus, and even the $\lambda\Pi^-$-calculus, can be used to express proofs of minimal predicate logic, following the Brouwer-Heyting-Kolmogorov interpretation and the Curry-de Bruijn-Howard correspondence. Let $\mathcal{L}$ be a language in predicate logic, we consider a context Γ formed with a variable ι of type $Type$ – or variables $\iota_1, ..., \iota_n$ of type $Type$ when $\mathcal{L}$ is many-sorted —, for each function symbol f of $\mathcal{L}$, a variable f of type $\iota \Rightarrow ... \Rightarrow \iota \Rightarrow \iota$ and for each predicate symbol P of $\mathcal{L}$, a variable P of type $\iota \Rightarrow ... \Rightarrow \iota \Rightarrow Type$.

To each formula P containing free variables $x_1, ..., x_p$ we associate a term P° of type $Type$ in the context $\Gamma, x_1 : \iota, ..., x_p : \iota$ translating each variable, function symbol and predicate symbol by itself and the implication symbol and the universal quantifier by a product.

To each proof π, in minimal natural deduction, of a sequent $A_1, ..., A_n \vdash B$ with free variables $x_1, ..., x_p$, we can associate a term π° of type B° in the context $\Gamma, x_1 : \iota, ..., x_p : \iota, \alpha_1 : A_1^\circ, ..., \alpha_n : A_n^\circ$. From the strong termination of the $\lambda\Pi$-calculus, we get cut elimination for minimal predicate logic. If B is an atomic formula, there is no cut free proof, hence no proof at all, of $\vdash B$.

2 The $\lambda\Pi$-calculus modulo

The $\lambda\Pi$-calculus allows to express proofs in pure minimal predicate logic. To express proofs in a theory $\mathcal{T}$, we can declare a variable for each axiom of $\mathcal{T}$ and consider proofs-terms containing such free variables, this is the idea of the Logical Framework [11]. However, when considering such open terms most benefits of termination, such as the existence of empty types, are lost.

An alternative is to replace axioms by rewrite rules, moving from predicate logic to *Deduction modulo* [7, 9]. Such extensions of type systems with rewrite rules to express proofs in Deduction modulo have been defined in [3] and [13]. We shall present now an extension of the $\lambda\Pi$-calculus: the $\lambda\Pi$-calculus modulo.

Recall that if Σ, Γ and Δ are contexts, a substitution θ, binding the variables declared in Γ, is said to be *of type* $\Gamma \rightsquigarrow \Delta$ *in* Σ if for all x declared of type T in Γ, we have $\Sigma\Delta \vdash \theta x : \theta T$, and that, in this case, if $\Sigma\Gamma \vdash u : U$, then $\Sigma\Delta \vdash \theta u : \theta U$.

A *rewrite rule* is a quadruple $l \longrightarrow^{\Gamma,T} r$ where Γ is a context and l, r and T are β-normal terms. Such a rule is said to be *well-typed* in the context Σ if, in the $\lambda\Pi$-calculus, the context $\Sigma\Gamma$ is well-formed and the terms l and r have type T in this context.

If Σ is a context, $l \longrightarrow^{\Gamma,T} r$ is a rewrite rule well-typed in Σ and θ is a substitution of type $\Gamma \rightsquigarrow \Delta$ in Σ then the terms θl and θr both have type θT in the context $\Sigma\Delta$. We say that the term θl *rewrites* to the term θr.

If Σ is a context and $\mathcal{R}$ a set of rewrite rules well-typed in the $\lambda\Pi$-calculus in Σ, then the *congruence generated by* $\mathcal{R}$, $\equiv_{\mathcal{R}}$, is the smallest congruence such that if t rewrites to u then $t \equiv_{\mathcal{R}} u$.

Definition 4 ($\lambda\Pi$-modulo). *Let Σ be a context and $\mathcal{R}$ a rewrite system in Σ. Let $\equiv_{\beta\mathcal{R}}$ be the congruence of terms generated by the rules of $\mathcal{R}$ and the rule β.*

The $\lambda\Pi$-calculus modulo $\mathcal{R}$ *is the extension of the $\lambda\Pi$-calculus obtained by replacing the relation $\equiv_\beta$ by $\equiv_{\beta\mathcal{R}}$ in the conversion rules*

$$\frac{\Gamma \vdash A : Type \quad \Gamma \vdash B : Type \quad \Gamma \vdash t : A}{\Gamma \vdash t : B}\ \textbf{Conversion}\ A \equiv_{\beta\mathcal{R}} B$$

$$\frac{\Gamma \vdash A : Kind \quad \Gamma \vdash B : Kind \quad \Gamma \vdash t : A}{\Gamma \vdash t : B}\ \textbf{Conversion2}\ A \equiv_{\beta\mathcal{R}} B$$

Notice that we can also extend the $\lambda\Pi^-$-calculus with rewrite rules. In this case, we introduce conversion rules, using the congruence defined by the system $\mathcal{R}$ alone.

Example 1. Consider the context $\Sigma = [P : Type,\ Q : Type]$ and the rewrite system $\mathcal{R}$ formed with the rule $P \longrightarrow (Q \Rightarrow Q)$. The term $\lambda f : P\ \lambda x : Q\ (f\ x)$ is well-typed in the $\lambda\Pi$-calculus modulo $\mathcal{R}$.

3 The Pure Type Systems

There are several ways to extend the functional interpretation of proofs to simple type theory. The first is to use the fact that simple type theory can be expressed in Deduction modulo with rewrite rules only [8]. Thus, the proofs of simple type theory can be expressed in the $\lambda\Pi$-calculus modulo, and even in the $\lambda\Pi^{-}$-calculus modulo. The second is to extend the $\lambda\Pi$-calculus by adding the following typing rules, allowing for instance the construction of the type $\Pi P : Type\ (P \Rightarrow P)$.

$$\frac{\Gamma \vdash A : Kind \quad \Gamma[x : A] \vdash B : Type}{\Gamma \vdash \Pi x : A\ B : Type}\ \textbf{Product3}$$

$$\frac{\Gamma \vdash A : Kind \quad \Gamma[x : A] \vdash B : Kind}{\Gamma \vdash \Pi x : A\ B : Kind}\ \textbf{Product4}$$

$$\frac{\Gamma \vdash A : Kind \quad \Gamma[x : A] \vdash B : Type \quad \Gamma[x : A] \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}\ \textbf{Abstraction3}$$

$$\frac{\Gamma \vdash A : Kind \quad \Gamma[x : A] \vdash B : Kind \quad \Gamma[x : A] \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}\ \textbf{Abstraction4}$$

We obtain the *Calculus of Constructions* [4].

The rules of the simply typed λ-calculus, the $\lambda\Pi$-calculus and of the Calculus of Constructions can be presented in a schematic way as follows.

Definition 5 (Pure type system). *A* Pure Type System *[2, 15, 1] P is defined by a set S, whose elements are called* sorts*, a subset A of $S \times S$, whose elements are called* axioms *and a subset R of $S \times S \times S$, whose elements are called* rules*. The typing rules of P are*

$$\frac{}{[\,]\ \textit{well-formed}}\ \textbf{Empty}$$

$$\frac{\Gamma \vdash A : s}{\Gamma[x : A]\quad \textit{well-formed}}\ \textbf{Declaration}\ s \in S\ \textit{and}\ x\ \textit{not in}\ \Gamma$$

$$\frac{\Gamma\ \textit{well-formed}}{\Gamma \vdash s_1 : s_2}\ \textbf{Sort}\ \langle s_1, s_2\rangle \in A$$

$$\frac{\Gamma\ \textit{well-formed} \quad x : A \in \Gamma}{\Gamma \vdash x : A}\ \textbf{Variable}$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma[x : A] \vdash B : s_2}{\Gamma \vdash \Pi x : A\ B : s_3}\ \textbf{Product}\ \langle s_1, s_2, s_3\rangle \in R$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma[x : A] \vdash B : s_2 \quad \Gamma[x : A] \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}\ \textbf{Abstraction}\ \langle s_1, s_2, s_3\rangle \in R$$

$$\frac{\Gamma \vdash t : \Pi x : A\ B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : (u/x)B}\ \textbf{Application}$$

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : s \quad \Gamma \vdash t : A}{\Gamma \vdash t : B}\ \textbf{Conversion}\ s \in S \quad A \equiv_\beta B$$

The simply typed λ-calculus is the system defined by the sorts $Type$ and $Kind$, the axiom $\langle Type, Kind\rangle$ and the rule $\langle Type, Type, Type\rangle$. The $\lambda\Pi$-calculus is the system defined by the same sorts and axiom and the rules $\langle Type, Type, Type\rangle$ and $\langle Type, Kind, Kind\rangle$. The Calculus of Constructions is the system defined by the same sorts and axiom and the rules $\langle Type, Type, Type\rangle$, $\langle Type, Kind, Kind\rangle$, $\langle Kind, Type, Type\rangle$ and $\langle Kind, Kind, Kind\rangle$. Other examples of Pure Type Systems are Girard's systems F and Fω.

In all Pure Type Systems, types are preserved under reduction and the β-reduction relation is confluent. It terminates in some systems, such as the $\lambda\Pi$-calculus, the Calculus of Constructions, the system F and the system Fω. Uniqueness of types is lost in general, but it holds for the $\lambda\Pi$-calculus, the Calculus of Constructions, the system F and the system Fω, and more generally for all functional Pure Type Systems.

Definition 6 (Functional Type System). *A type system is said to be* functional *if*

$$\langle s_1, s_2\rangle \in A \text{ and } \langle s_1, s_3\rangle \in A \text{ implies } s_2 = s_3$$

$$\langle s_1, s_2, s_3\rangle \in R \text{ and } \langle s_1, s_2, s_4\rangle \in R \text{ implies } s_3 = s_4$$

4 Embedding functional Pure Type Systems in the $\lambda\Pi$-calculus modulo

We have seen that the $\lambda\Pi$-calculus modulo and the Pure Type Systems are two extensions of the $\lambda\Pi$-calculus. At a first glance, they seem quite different as the latter adds more typing rules to the $\lambda\Pi$-calculus, while the former adds more computation rules. But they both allow to express proofs of simple type theory.

We show in this section that functional Pure Type Systems can, in fact, be embedded in the $\lambda\Pi$-calculus modulo with an appropriate rewrite system.

4.1 Definition

Consider a functional Pure Type System $P = \langle S, A, R\rangle$. We build the following context and rewrite system.

The context Σ_P contains, for each sort s, two variables

$$U_s : Type \quad \text{and} \quad \varepsilon_s : U_s \Rightarrow Type$$

for each axiom $\langle s_1, s_2\rangle$, a variable

$$\dot{s_1} : U_{s_2}$$

and for each rule $\langle s_1, s_2, s_3\rangle$, a variable

$$\dot{\Pi}_{\langle s_1, s_2, s_3\rangle} : \Pi X : U_{s_1}\ (((\varepsilon_{s_1}\ X) \Rightarrow U_{s_2}) \Rightarrow U_{s_3})$$

The type U_s is called the *universe* of s and the symbol ε_s the *decoding function* of s.

The rewrite rules are

$$\varepsilon_{s_2}(\dot{s}_1) \longrightarrow U_{s_1}$$

in the empty context and with the type $Type$, and

$$\varepsilon_{s_3}(\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ X\ Y) \longrightarrow \Pi x : (\varepsilon_{s_1}\ X)\ (\varepsilon_{s_2}\ (Y\ x))$$

in the context $X : U_{s_1}, Y : (\varepsilon_{s_1}\ X) \Rightarrow U_{s_2}$ and with the type $Type$.

These rules are called the *universe-reduction rules*, we write $\equiv_P$ for the equivalence relation generated by these rules and the rule β and we call the $\lambda\Pi_P$-calculus the $\lambda\Pi$-calculus modulo these rewrite rules and the rule β. To ease notations, in the $\lambda\Pi_P$-calculus, we do not recall the context Σ_P in each sequent and write $\Gamma \vdash t : T$ for $\Sigma_P\Gamma \vdash t : T$, and we note $\equiv$ for $\equiv_P$ when there is no ambiguity.

Example 2. The embedding of the Calculus of Constructions is defined by the context

$$\dot{Type} : U_{Kind} \qquad U_{Type} : Type \qquad U_{Kind} : Type$$

$$\varepsilon_{Type} : U_{Type} \Rightarrow Type \qquad \varepsilon_{Kind} : U_{Kind} \Rightarrow Type$$

$$\dot{\Pi}_{\langle Type,Type,Type\rangle} : \Pi X : U_{Type}\ (((\varepsilon_{Type}\ X) \Rightarrow U_{Type}) \Rightarrow U_{Type})$$

$$\dot{\Pi}_{\langle Type,Kind,Kind\rangle} : \Pi X : U_{Type}\ (((\varepsilon_{Type}\ X) \Rightarrow U_{Kind}) \Rightarrow U_{Kind})$$

$$\dot{\Pi}_{\langle Kind,Type,Type\rangle} : \Pi X : U_{Kind}\ (((\varepsilon_{Kind}\ X) \Rightarrow U_{Type}) \Rightarrow U_{Type})$$

$$\dot{\Pi}_{\langle Kind,Kind,Kind\rangle} : \Pi X : U_{Kind}\ (((\varepsilon_{Kind}\ X) \Rightarrow U_{Kind}) \Rightarrow U_{Kind})$$

and the rules

$$\varepsilon_{Kind}(\dot{Type}) \longrightarrow U_{Type}$$

$$\varepsilon_{Type}(\dot{\Pi}_{\langle Type,Type,Type\rangle}\ X\ Y) \longrightarrow \Pi x : (\varepsilon_{Type}\ X)\ (\varepsilon_{Type}\ (Y\ x))$$

$$\varepsilon_{Kind}(\dot{\Pi}_{\langle Type,Kind,Kind\rangle}\ X\ Y) \longrightarrow \Pi x : (\varepsilon_{Type}\ X)\ (\varepsilon_{Kind}\ (Y\ x))$$

$$\varepsilon_{Type}(\dot{\Pi}_{\langle Kind,Type,Type\rangle}\ X\ Y) \longrightarrow \Pi x : (\varepsilon_{Kind}\ X)\ (\varepsilon_{Type}\ (Y\ x))$$

$$\varepsilon_{Kind}(\dot{\Pi}_{\langle Kind,Kind,Kind\rangle}\ X\ Y) \longrightarrow \Pi x : (\varepsilon_{Kind}\ X)\ (\varepsilon_{Kind}\ (Y\ x))$$

Definition 7 (Translation). *Let Γ be a context in a functional Pure Type System P and t a term well-typed in Γ, we defined the translation $|t|$ of t in Γ, that is a term in $\lambda\Pi_P$, as follows*

- $|x| = x$,
- $|s| = \dot{s}$,
- $|\Pi x : A\ B| = \dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ |A|\ (\lambda x : (\varepsilon_{s_1}\ |A|)\ |B|)$, *where s_1 is the type of A, s_2 is the type of B and s_3 the type of $\Pi x : A\ B$,*
- $|\lambda x : A\ t| = \lambda x : (\varepsilon_s\ |A|)\ |t|$,
- $|t\ u| = |t|\ |u|$.

Definition 8 (Translation as a type). *Consider a term A of type s for some sort s. The* translation of A as a type *is*

$$\|A\| = \varepsilon_s\ |A|.$$

Note that if A is a well-typed sort s' then

$$\|s'\| = \varepsilon_s\ \dot{s}' \equiv_P U_{s'}.$$

We extend this definition to non well-typed sorts, such as the sort $Kind$ in the Calculus of Constructions, by

$$\|s'\| = U_{s'}$$

The translation of a well formed context is defined by

$$\|[\,]\| = [\,] \quad \textit{and} \quad \|\Gamma[x:A]\| = \|\Gamma\|[x:\|A\|]$$

4.2 Soundness

Proposition 1. *1.* $|(u/x)t| = (|u|/x)|t|$, $\|(u/x)t\| = (|u|/x)\|t\|$.
2. If $t \longrightarrow_\beta u$ *then* $|t| \longrightarrow_\beta |u|$.

Proof. 1. By induction on t.
2. Because a β-redex is translated as a β-redex.

Proposition 2. $\|\Pi x : A\ B\| \equiv_P \Pi x : \|A\|\ \|B\|$

Proof. Let s_1 be the type of A, s_2 that of B and s_3 that of $\Pi x : A\ B$. We have $\|\Pi x : A\ B\| = \varepsilon_{s_3}\ |\Pi x : A\ B| = \varepsilon_{s_3}\ (\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ |A|\ (\lambda x : (\varepsilon_{s_1}\ |A|)\ |B|))$
$\equiv_P \Pi x : (\varepsilon_{s_1}\ |A|)\ (\varepsilon_{s_2}\ ((\lambda x : (\varepsilon_{s_1}\ |A|)\ |B|)\ x)) \equiv_P \Pi x : (\varepsilon_{s_1}\ |A|)\ (\varepsilon_{s_2}\ |B|)$
$= \Pi x : \|A\|\ \|B\|$.

Example 3. In the Calculus of Constructions, the translation as a type of $\Pi X : Type\ (X \Rightarrow X)$ is $\Pi X : U_{Type}\ ((\varepsilon_{Type}\ X) \Rightarrow (\varepsilon_{Type}\ X))$. The translation as a term of $\lambda X : Type\ \lambda x : X\ x$ is the term $\lambda X : U_{Type}\ \lambda x : (\varepsilon_{Type} X)\ x$. Notice that the former is the type of the latter. The generalization of this remark is the following proposition.

Proposition 3 (Soundness).

$$\textit{If}\ \ \Gamma \vdash t : B\ \ \textit{in}\ P\ \textit{then}\ \ \|\Gamma\| \vdash |t| : \|B\|\ \ \textit{in}\ \lambda\Pi_P.$$

Proof. By induction on t.

- If t is a variable, this is trivial.
- If $t = s_1$ then $B = s_2$ (where $\langle s_1, s_2\rangle$ is an axiom), we have $\dot{s_1} : U_{s_2} = \|s_2\|$.
- If $t = \Pi x : C\ D$, let s_1 be the type of C, s_2 that of D and s_3 that of t. By induction hypothesis, we have

$$\|\Gamma\| \vdash |C| : U_{s_1}\ \ \textit{and}\ \ \|\Gamma\|, x : \|C\| \vdash |D| : U_{s_2}$$

i.e.

$$\|\Gamma\|, x : (\varepsilon_{s_1}\ |C|) \vdash |D| : U_{s_2}$$

Thus

$$\|\Gamma\| \vdash (\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ |C|\ \lambda x : (\varepsilon_{s_1}\ |C|)\ |D|) : U_{s_3}$$

i.e.

$$\|\Gamma\| \vdash |\Pi x : C\ D| : \|s_3\|$$

– If $t = \lambda x : C\ u$, then we have

$$\Gamma, x : C \vdash u : D$$

and $B = \Pi x : C\ D$. By induction hypothesis, we have

$$\|\Gamma\|, x : \|C\| \vdash |u| : \|D\|$$

i.e.

$$\|\Gamma\|, x : (\varepsilon_{s_1}\ |C|) \vdash |u| : \|D\| \quad \textit{then} \quad \|\Gamma\| \vdash \lambda x : (\varepsilon_{s_1}\ |C|)\ |u| : \Pi x : \|C\|\ \|D\|$$

i.e.

$$\|\Gamma\| \vdash |t| : \|\Pi x : C\ D\|$$

– If $t = u\ v$, then we have

$$\Gamma \vdash u : \Pi x : C\ D, \quad \Gamma \vdash v : C$$

and $B = (v/x)D$. By induction hypothesis, we get

$$\|\Gamma\| \vdash |u| : \|\Pi x : C\ D\| = \Pi x : \|C\|\ \|D\| \quad \textit{and} \quad \|\Gamma\| \vdash |v| : \|C\|$$

Thus

$$\|\Gamma\| \vdash |t| : (|v|/x)\|D\| = \|(v/x)D\|$$

4.3 Termination

Proposition 4. *If $\lambda\Pi_P$ terminates then P terminates.*

Proof. Let t_1 be a well-typed term in P and $t_1, t_2, ...$ be a reduction sequence of t_1 in P. By Proposition 3, the term $|t_1|$ is well-typed in $\lambda\Pi_P$ and, by Proposition 1, $|t_1|, |t_2|, ...$ is a reduction sequence of $|t_1|$ in $\lambda\Pi_P$. Hence it is finite.

4.4 Confluence

Proposition 5. *For any functional Pure Type System P, the relation $\longrightarrow$ is confluent in $\lambda\Pi_P$*

Like that of pure λ-calculus, the reduction relation of $\lambda\Pi_P$ is not strongly confluent. Thus, we introduce another reduction relation ($\twoheadrightarrow$) that can reduce, in one step, none to all the $\beta\mathcal{R}$-redices that appears in a term, that is strongly confluent and such that $\twoheadrightarrow^* = \longrightarrow^*$. Then, from the confluence of the relation $\twoheadrightarrow$, we deduce that of the relation $\longrightarrow$. See the long version of the paper for the full proof.

5 Conservativity

Let P be a functional Pure Type System. We could attempt to prove that if the type $\|A\|$ is inhabited in $\lambda\Pi_P$, then A is inhabited in P, and more precisely that if Γ is a context and A a term in P and t a term in $\lambda\Pi_P$, such that $\|\Gamma\| \vdash t : \|A\|$, then there exists a term u of P such that $|u| = t$ and $\Gamma \vdash u : A$. Unfortunately this property does not hold in general as shown by the following counterexamples.

Example 4. If P is the simply-typed lambda-calculus, then the polymorphic identity is not well-typed in P, in particular:
$nat : Type \not\vdash ((\lambda X : Type\ \lambda x : X\ x)\ nat) : (nat \Rightarrow nat)$
however, in $\lambda\Pi$, we have
$nat : \|Type\| \vdash ((\lambda X : \|Type\|\ \lambda x : \|X\|\ x)\ |nat|) : \|nat \Rightarrow nat\|$.

Example 5. If $\langle s_1, s_2, s_3\rangle \in R$, $\Sigma_P \vdash \dot{\Pi}_{\langle s_1,s_2,s_3\rangle} : \|\Pi X : s_1\ ((X \Rightarrow s_2) \Rightarrow s_3)\|$ but the term $\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}$ is not the translation of any term of P.

Therefore, we shall prove a slightly weaker property: that if the type $\|A\|$ is inhabited *by a normal term* in $\lambda\Pi_P$, then A is inhabited in P. Notice that this restriction vanishes if $\lambda\Pi_P$ is terminating.

We shall prove, in a first step, that if $\|\Gamma\| \vdash t : \|A\|$, and t is a weak η-long normal term then there exists a term in u such that such that $|u| = t$ and $\Gamma \vdash u : A$. Then we shall get rid of this restriction on weak η-long forms.

Definition 9. *A term t of $\lambda\Pi_P$ is a weak η-long term if and only if each occurrence of $\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}$ in t, is in a subterm of the form $(\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ t_1\ t_2)$ (*i.e. *each occurrence of $\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}$ is η-expanded).*

Definition 10 (Back translation). *We suppose that P contains at least one sort: s_0. Then we define a translation from $\lambda\Pi_P$ to P as follows:*

- $x^* = x, \qquad s^* = s_0 \qquad \dot{s}^* = s, \qquad U_s^* = s,$
- $(\Pi x : A\ B)^* = \Pi x : A^*\ B^*,$
- $(\lambda x : A\ t)^* = \lambda x : A^*\ t^*,$
- $(\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ A\ B)^* = \Pi x : A^*\ (B^*\ x),$
- $(\varepsilon_s\ u)^* = u^*,$
- $(t\ u)^* = t^*\ u^*$ *otherwise.*

Proposition 6. *The back translation $(.)^*$ is a right inverse of $|.|$ and $\|.\|$* i.e. *for all t such that $|t|$ (*resp. *$\|t\|$) is well defined, $|t|^* = t$ (*resp. *$\|t\|^* = t$).*

Proof. By induction on the structure of t.

Proposition 7. *For all t, u terms and x variable of $\lambda\Pi_P$,*

1. $((u/x)t)^* = (u^*/x)t^*$
2. *If $t \longrightarrow u$ then $t^* \longrightarrow_\beta^* u^*$ in P.*

Proof. 1. By induction on t.
2. If $t \longrightarrow_\beta u$ then $t^* \longrightarrow_\beta u^*$, and if $t \longrightarrow_{\mathcal{R}} u$, then $t^* = u^*$.

Proposition 8. *For all terms A, B of P and C, D of $\lambda\Pi_P$ (such that $\|A\|$ and $\|B\|$ are well defined),*

1. *If $A \equiv_\beta B$, then $\|A\| \equiv \|B\|$.*
2. *If $C \equiv D$, then $C^* \equiv_\beta D^*$.*
3. *If $\|A\| \equiv \|B\|$, then $A \equiv_\beta B$.*
4. *If $C \equiv \|A\|$, then $C \equiv \|C^*\|$.*

Proof. 1. By induction on the length of the path of β-reductions and β-expansions between A and B, and by Proposition 1.
2. By the same reasoning as for the first point, using Proposition 7.
3. By the second point and Proposition 6.
4. By the first and second points and Proposition 6.

Proposition 9 (Conservativity). *If there exists a context Γ, a term A of P, and a term t, in weak η-long normal form, of $\lambda\Pi_P$, such that $\|\Gamma\| \vdash t : \|A\|$, Then there exists a term u of P such that $|u| \equiv t$ and $\Gamma \vdash u : A$.*

Proof. By induction on t.

- If t is a well-typed product or sort, then it cannot be typed by a translated type (by confluence of $\lambda\Pi_P$).
- If $t = \lambda x : B\ t'$. The term t is well typed, thus there exists a term C of $\lambda\Pi_P$, such that $\|\Gamma\| \vdash t : \Pi x : B\ C$. Therefore $\|A\| \equiv \Pi x : B\ C$ (α).
  And $\Pi x : B\ C \equiv \|(\Pi x : B\ C)^*\| = \|\Pi x : B^*\ C^*\| \equiv \Pi x : \|B^*\|\ \|C^*\|$
  In particular (by confluence of $\lambda\Pi_P$),

  $$B \equiv \|B^*\|, \quad C \equiv \|C^*\| \quad \text{and} \quad \|\Gamma\| \vdash \lambda x : B\ t' \ :\ \Pi x : \|B^*\|\ \|C^*\|$$

  Therefore $\|\Gamma\|, x : \|B^*\| \vdash t' : \|C^*\|$. The term $\lambda x : B\ t'$ is in weak η-long normal form, thus t' is also in weak η-long normal form, and, by induction hypothesis, there exists a term u' of P, such that $|u'| \equiv t'$ and
  $\Gamma, x : B^* \vdash u' : C^*$. Therefore $\Gamma \vdash \lambda x : B^*\ u' \ :\ \Pi x : B^*\ C^*$ (β). Moreover, $A \equiv_\beta \Pi x : B^*\ C^*$ by (α) and Proposition 8. Thus, by the conversion rule of P, we get $\Gamma \vdash \lambda x : B^*\ u' \ :\ A$.
  And $|\lambda x : B^*\ u'| = \lambda x : \|B^*\|\ |u'| \equiv \lambda x : B\ t' = t$.
- If t is an application or a variable, as it is normal, it has the form $x\ t_1\ ...\ t_n$ for some variable x and terms $t_1, ..., t_n$. We have $\|\Gamma\| \vdash x\ t_1\ ...\ t_n : \|A\|$ (α_0).
  - If x is a variable of the context Σ_P,
    - If $x = s_1$ (where $\langle s_1, s_2\rangle$ is an axiom of P),
      then $n = 0$ (because t is well typed) and $\|A\| = U_{s_2}$.
      We have $\vdash s_1 : s_2$ in P, therefore $\Gamma \vdash s_1 : s_2$.
    - If $x = U_s$ (where s is a sort of P), then $n = 0$ and $\|A\| \equiv Type$.
      That's an absurdity by confluence of $\lambda\Pi_P$.

$*$ If $x = \varepsilon_s$ (where s is a sort of P), then, as t is well typed $n \leq 1$.
  $\star$ If $n = 1$, then $\|\Gamma\| \vdash t_1 : U_s$, and $\|A\| \equiv Type$ (absurdity).
  $\star$ If $n = 0$, then $\|A\| \equiv U_s \Rightarrow Type$, thus by Propositions 8 and 2, $U_s \Rightarrow Type \equiv \|(U_s \Rightarrow Type)^*\| = \|s \Rightarrow s_0\| \equiv \|s\| \Rightarrow \|s_0\|$.
  Therefore $Type \equiv \|s_0\|$ (absurdity).

$*$ If $x = \dot{\Pi}_{\langle s_1,s_2,s_3\rangle}$ (where $\langle s_1, s_2, s_3\rangle$ is a rule of P), then as t is well-typed and in weak η-long form, $n = 2$. We have $\|A\| \equiv U_{s_3}$ thus $A \equiv s_3$ by Proposition 8.
And $\|\Gamma\| \vdash t_1 : U_{s_1}$ *i.e.* $\|\Gamma\| \vdash t_1 : \|s_1\|$.
And $\|\Gamma\|, t_1 : U_{s_1} \vdash t_2 : ((\varepsilon_{s_1} t_1) \Rightarrow U_{s_2})$ (α_1)
t_1 is also in weak η-long normal form, then, by induction hypothesis, there exists a term u_1 of P such that:
$$|u_1| \equiv t_1 \quad and \quad \Gamma \vdash u_1 : s_1 \quad (\beta_1)$$
Then, by (α_1), $\|\Gamma\|, t_1 : \|s_1\| \vdash t_2 : \|u_1 \Rightarrow s_2\|$.
In particular, $\|\Gamma\|, t_1 : \|s_1\| \vdash t_2 : \|u_1\| \Rightarrow \|s_2\|$.
However t_2 is also in weak η-long normal form, then there exists a term t'_2 (in weak η-long normal form) of $\lambda\Pi_P$ such that
$$t_2 = \lambda x : \|u_1\|\ t'_2 \quad and \quad \|\Gamma\|, x : \|u_1\| \vdash t'_2 : \|s_2\|$$
By induction hypothesis, there exists a term u'_2 of P, such that
$$|u'_2| \equiv t'_2 \quad and \quad \Gamma, x : u_1 \vdash u'_2 : s_2 \quad (\beta_2)$$
Then we choose $u = \Pi x : u_1\ u'_2$ that verifies $\Gamma \vdash u : s_3$, by (β_1), (β_2), and the fact that $\langle s_1, s_2, s_3\rangle$ is a rule of P. And, finally, $|u| = \dot{\Pi}_{\langle s_1,s_2,s_3\rangle}|u_1|\ (\lambda x : (\varepsilon_{s_1}|u_1|)\ |u'_2| \equiv \dot{\Pi}_{\langle s_1,s_2,s_3\rangle} t_1\ t_2 = t$

$\multimap$ If x is a variable of the context Γ,
For $k \in \{0, .., n\}$, let (H_k) be the statement: *"The term $x\ t_1\ ...\ t_k$ is typable in $\|\Gamma\|$ and its type is in the image of $\|.\|$"*.
We first prove (H_0),...,(H_n) by induction.
  $\star$ $k = 0 : x$ is a variable of the context Γ, then there exists a well typed term or a sort T in P such that Γ contains $x : T$. Therefore $\|\Gamma\|$ contains $x : \|T\|$.
  $\star$ $0 \leq k \leq n - 1$: We suppose (H_k).
  $x\ t_1\ ...\ t_{k+1}$ is well typed in Γ, then there exists terms D and E of $\lambda\Pi_P$ such that $\|\Gamma\| \vdash t_{k+1} : D$ (δ_1) ,
  $\|\Gamma\| \vdash x\ t_1\ ...\ t_k : \Pi y : D\ E$ (δ_2), and $\|\Gamma\| \vdash x\ t_1\ ...\ t_{k+1} : E$ (δ_3).
  However, by (H_k), we can type $x\ t_1\ ...\ t_k$ by a translated type in $\|\Gamma\|$, then by (δ_2) and Proposition 8, $\Pi y : D\ E \equiv \Pi y : \|D^*\|\ \|E^*\|$.
  In particular, $E \equiv \|E^*\|$ (η_1) ,
  We conclude, by (δ_3), (η_1) and the conversion rule of $\lambda\Pi_P$.
Then, if $n = 0$, we take $u = x$ and Γ contains $x : T$ with $\|T\| \equiv \|A\|$.
And, if $n > 0$, then, by (α_0), there exists terms B and C of $\lambda\Pi_P$ such that $\|\Gamma\| \vdash t_n : B$ (θ_1) and $\|\Gamma\| \vdash x\ t_1\ ...\ t_{n-1} : \Pi y : B\ C$ (θ_2) with $\|A\| \equiv (t_n/y)C$ (θ_3) . Then, by (H_{n-1}), (θ_2), and Proposition 8.4, $\Pi y : B\ C \equiv \Pi y : \|B^*\|\ \|C^*\|$, therefore $B \equiv \|B^*\|$ and $C \equiv \|C^*\|$.
Thus, $\|\Gamma\| \vdash t_n : \|B^*\|$ and $\|\Gamma\| \vdash x\ t_1\ ...\ t_{n-1} : \|\Pi y : B^*\ C^*\|$.
t_n and $x\ t_1\ ...\ t_{n-1}$ are both in weak η-long normal form, then, by induction hypothesis, there exists terms w_1 and w_2 of P such that:

$$|w_1| \equiv x\ t_1\ ...t_{n-1} \text{ and } \Gamma \vdash w_1 : \Pi y : B^*\ C^*$$
$$|w_2| \equiv t_n \text{ and } \Gamma \vdash w_2 : B^*$$

Let $u = w_1\ w_2$, we have:

$$|u| = |w_1|\ |w_2| \equiv x\ t_1\ ...\ t_{n-1}\ t_n \text{ and } \Gamma \vdash u : (w_2/y)C^*.$$

However, by (θ_3) and Proposition 8, we have:
$A \equiv (t_n^*/y)C^* \equiv (w_2/y)C^*$, and, finally, $\Gamma \vdash u : A$.

Finally, we get rid of the weak η-long form restriction with the following Propositions.

Proposition 10. *For all terms A, B of $\lambda\Pi_P$, and for all well typed term or sort C of P,*

1. *If $A \longrightarrow B$ then $A" \longrightarrow^* B"$*
2. *If $A \equiv B$ then $A" \equiv B"$*
3. *If A is in weak η-long form, then $A" \longrightarrow_\beta^* A$, in particular $A" \equiv A$*
4. *$\|C\|" \equiv \|C\|$*
5. *If $A \equiv \|C\|$ then $A" \equiv A$*

Proof. 1. If $A \longrightarrow_\beta B$, then $A" \longrightarrow_\beta B"$ (by induction on A).
If $A \longrightarrow_{\mathcal{R}} B$,
- for all axiom $\langle s_1, s_2\rangle$, $(\varepsilon_{s_2}\ (\dot{s_1}))" = \varepsilon_{s_2}\ (\dot{s_1}) \longrightarrow_{\mathcal{R}} U_{s_1} = (U_{s_1})"$.
- for all rule $\langle s_1, s_2, s_3\rangle$, $(\varepsilon_{s_3}(\dot{\Pi}_{\langle s_1,s_2,s_3\rangle} C\ D))" =$
  $\varepsilon_{s_3}((\lambda x : U_{s_1} \lambda y : ((\varepsilon_{s_1}\ x) \Rightarrow U_{s_2})\ (\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ x\ y))C"\ D")$
  $\longrightarrow_\beta^2 \varepsilon_{s_3}(\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}C"\ D") \longrightarrow_{\mathcal{R}} \Pi x : (\varepsilon_{s_1}\ C")\ (\varepsilon_{s_2}\ (D"\ x))$
  $= \Pi x : (\varepsilon_{s_1}\ C")\ (\varepsilon_{s_2}\ (D\ x)")$

2. By induction on the number of derivations and expansions from A to B.
3. By induction on A, remarking that $(\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ t_1\ t_2)" \longrightarrow_\beta^2 \dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ t_1"\ t_2"$.
4. A translated term $\|C\|$ is in weak η-long form.
5. If $A \equiv \|C\|$ then $A" \equiv \|C\|" \equiv \|C\|$, by the the second and fourth points.

Proposition 11. *Let t be a normal term of $\lambda\Pi_P$,*

$$\textit{if } \|\Gamma\| \vdash t : \|A\| \textit{ then } \|\Gamma\| \vdash t" : \|A\|$$

Proof. By induction on t.

- If t is a well-typed product or sort, then it cannot be typed by a translated type (by confluence of $\lambda\Pi_P$).
- If $t = \lambda x : B\ u$, then there exists a term C of $\lambda\Pi_P$, such that $\|A\| \equiv \Pi x : B\ C\ (\alpha)$, with $\Gamma, x : B \vdash u : C$.
  By (α), we have $B \equiv \|B^*\|\ (\beta)$ and $C \equiv \|C^*\|$. Thus $\Gamma, x : \|B^*\| \vdash u : \|C^*\|$. Then, by induction hypothesis, we have $\Gamma, x : \|B^*\| \vdash u" : \|C^*\|$, therefore $\Gamma \vdash \lambda x : \|B^*\|\ u" : \Pi x : \|B^*\|\ \|C^*\| \equiv \|A\|$ thus $\Gamma \vdash \lambda x : B\ u" : \|A\|$, by (β). Finally, by (β) and the Proposition 10.5, $\lambda x : B\ u" \equiv \lambda x : B"\ u"$, therefore, by subject reduction, $\Gamma \vdash t" = \lambda x : B"\ u" : \|A\|$
- If t is an application or a variable, as it is normal, it has the form $x\ t_1\ ...\ t_n$ for some variable x and terms t_1, ..., t_n. We have $\|\Gamma\| \vdash x\ t_1\ ...\ t_n : \|A\|$ (α_0).

$\multimap$ If x is a variable of the context Σ_P,
  * If $x = \dot{s_1}$ (where $\langle s_1, s_2\rangle$ is an axiom of P), then $n = 0$ (because t is well typed) and we have $(\dot{s_1})" = \dot{s_1}$.
  * If $x = U_s$ (where s is a sort of P), then $n = 0$ and $\|A\| \equiv Type$. That's an absurdity by confluence of $\lambda\Pi_P$.
  * If $x = \varepsilon_s$ (where s is a sort of P), then, as t is well typed $n \leq 1$.
    * If $n = 1$, then $\|\Gamma\| \vdash t_1 : U_s$, and $\|A\| \equiv Type$ (absurdity).
    * If $n = 0$, we have $(\varepsilon_s)" = \varepsilon_s$
  * If $x = \dot{\Pi}_{\langle s_1,s_2,s_3\rangle}$ (where $\langle s_1, s_2, s_3\rangle$ is a rule of P), then as t is well-typed, $n \leq 2$. Moreover, $\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}$, $(\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ t_1)$, and $(\dot{\Pi}_{\langle s_1,s_2,s_3\rangle}\ t_1\ t_2)$ have the same types than their weak η-long forms.

$\multimap$ If x is a variable of the context Γ,
  * If $n = 0$, we have $x" = x$.
  * If $n > 0$, then there exists terms B and C of $\lambda\Pi_P$ such that $\|\Gamma\| \vdash t_n : B\ \ (\alpha_1)$ and $\|\Gamma\| \vdash x\ t_1\ ...\ t_{n-1} : \Pi y : B\ C\ \ (\alpha_2)$ with $\|A\| \equiv (t_n/y)C\ \ (\alpha_3)$. As in the proof of Proposition 9, we can type $x\ t_1\ ...\ t_{n-1}$ by a translated type, then $\Pi y : B\ C \equiv \Pi y : \|B^*\|\ \|C^*\|$. In particular, $B \equiv \|B^*\|$ and $C \equiv \|C^*\|$.
    Thus, $\|\Gamma\| \vdash t_n : \|B^*\|$ and $\|\Gamma\| \vdash x\ t_1\ ...\ t_{n-1} : \|\Pi y : B^*\ C^*\|$.
    By induction hypothesis, we have $\|\Gamma\| \vdash t_n" : \|B^*\|$ and $\|\Gamma\| \vdash x\ t_1" \ ...\ t_{n-1}" : \Pi y : \|B^*\|\ \|C^*\|$. Finally, by (α_3) and Proposition 10.5, $\|\Gamma\| \vdash t" = x\ t_1" \ ...\ t_n" : (t_n"/y)C \equiv \|A\|$.

Theorem 1. *Let P be a functional Pure Type System, such that $\lambda\Pi_P$ is terminating. The type $\|A\|$ is inhabited by a closed term in $\lambda\Pi_P$ if and only if the type A is inhabited by a closed term in P.*

Proof. If A has a closed inhabitant in P, then by Proposition 3, $\|A\|$ has a closed inhabitant in $\lambda\Pi_P$. Conversely, if $\|A\|$ has a closed inhabitant then, by termination of $\lambda\Pi_P$ and Proposition 11, it has a closed inhabitant in weak η-long normal form and by Proposition 9, A has a closed inhabitant in P.

Remark 1. This conservativity property we have proved is similar to that of the Curry-de Bruijn-Howard correspondence. If the type A° is inhabited in $\lambda\Pi$-calculus, then the proposition A is provable in minimal predicate logic, but not all terms of type A° correspond to proofs of A. For instance, if A is the proposition $(\forall x\ P(x)) \Rightarrow P(c)$, then the normal term $\lambda\alpha : (\Pi x : \iota\ (P\ x))\ (\alpha\ c)$ corresponds to a proof of A but the term $\lambda\alpha : (\Pi x : \iota\ (P\ x))\ (\alpha\ ((\lambda y : \iota\ y)\ c))$ does not.

Remark 2. There are two ways to express proofs of simple type theory in the $\lambda\Pi$-calculus modulo. We can either use directly the fact that simple type theory can be expressed in Deduction modulo [8] or first express the proofs of simple type theory in the Calculus of Constructions and then embed the Calculus of Constructions in the $\lambda\Pi$-calculus modulo.

These two solutions have some similarities, in particular if we write o the symbol U_{Type}. But they have also some differences: the function $\lambda x\ x$ of simple

type theory is translated as the symbol I — or as the term $\lambda 1$ — in the first case, using a symbol I — or the symbols λ and 1 — specially introduced in the context to express this particular theory, while it is expressed as $\lambda x\ x$ using the symbol λ of the $\lambda\Pi$-calculus modulo in the second.

More generally in the second case, we exploit the similarities of the $\lambda\Pi$-calculus modulo and simple type theory — the fact that they both allow to express functions — to simplify the expression while the first method is completely generic and uses no particularity of simple type theory. This explains why this first expression requires only the $\lambda\Pi^{-}$-calculus modulo, while the second requires the conversion rule to contain β-conversion.

References

1. H. Barendregt, Lambda calculi with types, *Handbook of Logic in Computer Science*, S. Abramsky, D. Gabbay, and T. Maibaum (eds.), Oxford University Press, 1992, pp. 117-309.
2. S. Berardi, Towards a mathematical analysis of the Coquand-Huet Calculus of Constructions and the other systems in Barendregt's cube, *manuscript*, 1988.
3. F. Blanqui, Definitions by rewriting in the Calculus of Constructions. *Mathematical Structures in Computer Science*, 15, 1, 2005, pp. 37-92.
4. T. Coquand and G. Huet, The Calculus of Constructions, *Information and Computation*, 76, 1988, pp. 95-120.
5. D. Cousineau, *Un plongement conservatif des* Pure Type Systems *dans le lambda Pi modulo*, Master Parisien de Recherche en Informatique, 2006.
6. D. Dougherty, S. Ghilezan, P. Lescanne and S. Likavec , Strong normalization of the dual classical sequent calculus, *LPAR-2005*
7. G. Dowek, Th. Hardin, and C. Kirchner, Theorem proving modulo, *Journal of Automated Reasoning*, 31, 2003, pp. 33-72.
8. G. Dowek, Th. Hardin, and C. Kirchner, HOL-lambda-sigma: an intentional first-order expression of higher-order logic, *Mathematical Structures in Computer Science*, 11, 2001, pp. 1-25.
9. G. Dowek and B. Werner, Proof normalization modulo, *The Journal of Symbolic Logic*, 68, 4, 2003, pp. 1289-1316.
10. J.Y. Girard, Interprétation Fonctionnelle et Élimination des Coupures dans l'Arithmétique d'Ordre Supérieur, *Thèse de Doctorat*, Université Paris VII, 1972.
11. R. Harper, F. Honsell, and G. Plotkin, A framework for defining logics, *Journal of the ACM*, 40, 1, 1993, pp. 143-184.
12. P. Martin-Löf, Intuitionistic Type Theory, *Bibliopolis*, 1984.
13. B. Nordström, K. Petersson, and J.M. Smith, Martin-Löf's type theory. *Handbook of Logic in Computer Science*, S. Abramsky, D. Gabbay, and T. Maibaum (eds.), Clarendon Press, 2000, pp. 1-37.
14. E. Palmgren, On universes in type theory, *Twenty five years of constructive type theory*, Oxford Logic Guides, 36, Oxford University Press, 1998, pp. 191-204.
15. J. Terlouw, Een nadere bewijstheoretische analyse van GSTT's, *manuscript*, 1989.