Conservativity of embeddings in the lambda-Pi calculus modulo rewriting (long version)
Ali Assaf

To cite this version:
Ali Assaf. Conservativity of embeddings in the lambda-Pi calculus modulo rewriting (long version). 2015. hal-01084165v3

HAL Id: hal-01084165
https://hal.archives-ouvertes.fr/hal-01084165v3
Submitted on 20 Apr 2015

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
Conservativity of embeddings in the \(\lambda\Pi\) calculus modulo rewriting (long version)

Ali Assaf\(^1,2\)

1 Inria, Paris, France
2 École polytechnique, Palaiseau, France

Abstract

The \(\lambda\Pi\) calculus can be extended with rewrite rules to embed any functional pure type system. In this paper, we show that the embedding is conservative by proving a relative form of normalization, thus justifying the use of the \(\lambda\Pi\) calculus modulo rewriting as a logical framework for logics based on pure type systems. This result was previously only proved under the condition that the target system is normalizing. Our approach does not depend on this condition and therefore also works when the source system is not normalizing.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases \(\lambda\Pi\) calculus modulo rewriting, pure type systems, logical framework, normalization, conservativity

1 Introduction

The \(\lambda\Pi\) calculus modulo rewriting is a logical framework that extends the \(\lambda\Pi\) calculus [10] with rewrite rules. Through the Curry-de Bruijn-Howard correspondence, it can express properties and proofs of various logics. Cousineau and Dowek [6] introduced a general embedding of functional pure type systems (FPTS), a large class of typed \(\lambda\)-calculi, in the \(\lambda\Pi\) calculus modulo rewriting: for any FPTS \(\lambda S\), they constructed the system \(\lambda\Pi/S\) using appropriate rewrite rules, and defined two translation functions \(|M|\) and \(\parallel A\parallel\) that translate respectively the terms and the types of \(\lambda S\) to \(\lambda\Pi/S\). This embedding is complete, in the sense preserves typing: if \(\Gamma \vdash_{\lambda S} M : A\) then \(\parallel \Gamma \parallel \vdash_{\lambda\Pi/S} |M| : \parallel A\parallel\). From the logical point of view, it preserves provability. The converse property, called conservativity, was only shown partially: assuming \(\lambda\Pi/S\) is strongly normalizing, if there is a term \(N\) such that \(\parallel \Gamma \parallel \vdash_{\lambda\Pi/S} N : \parallel A\parallel\) then there is a term \(M\) such that \(\Gamma \vdash_{\lambda S} M : A\).

Normalization and conservativity

Not much is known about normalization in \(\lambda\Pi/S\). Cousineau and Dowek [6] showed that the embedding preserves reduction: if \(M \rightarrow M'\) then \(|M| \rightarrow^+ |M'|\). As a consequence, if \(\lambda\Pi/S\) is strongly normalizing (i.e. every well-typed term normalizes) then so is \(\lambda S\), but the converse might not be true \textit{a priori}. This was not enough to show the conservativity of the embedding, so the proof relied on the unproven assumption that \(\lambda\Pi/S\) is normalizing. This result is insufficient if one wants to consider the \(\lambda\Pi\) calculus modulo rewriting as a general logical framework for defining logics and expressing proofs in those logics, as proposed in [4, 5]. Indeed, if the embedding turns out to be inconsistent then checking proofs in the logical framework has very little benefit.
2 Conservativity of embeddings in the \( \lambda \Pi \) calculus modulo rewriting

Consider the PTS \( \lambda \text{HOL} \) that corresponds to higher order logic [1]:

\[
S = \text{Prop, Type, Kind} \\
A = (\text{Prop : Type}), (\text{Type : Kind}) \\
R = (\text{Prop, Prop, Prop}), (\text{Type, Prop, Prop}), (\text{Type, Type, Type})
\]

This PTS is strongly normalizing, and therefore consistent. A polymorphic variant of \( \lambda \text{HOL} \) is specified by \( U^- = \text{HOL} + (\text{Kind, Type, Type}) \). It turns out that \( \lambda U^- \) is inconsistent: there is a term \( \omega \) such that \( \vdash \lambda U^- \omega : \Pi \alpha : \text{Prop} \alpha \) and which is not normalizing [1]. We motivate the need for a proof of conservativity with the following example.

▶ Example 1.1. The polymorphic identity function \( I = \lambda \alpha : \text{Type}. \lambda x : \alpha. x \) is not well-typed in \( \lambda \text{HOL} \), but it is well-typed in \( \lambda U^- \) and so is its type:

\[
\vdash \lambda U^- I : \Pi \alpha : \text{Type}. \alpha \rightarrow \alpha \\
\vdash \lambda U^- \Pi \alpha : \text{Type}. \alpha \rightarrow \alpha : \text{Type}
\]

However, the translation \( |I| = \lambda \alpha : u \text{Type}. \lambda x : \varepsilon \text{Type}. \alpha. x \) is well-typed in \( \lambda \Pi/\text{HOL} \):

\[
\vdash \lambda \Pi/\text{HOL} |I| : \Pi \alpha : u \text{Type}. \varepsilon \text{Type} \alpha \rightarrow \varepsilon \text{Type} \alpha \\
\vdash \lambda \Pi/\text{HOL} \Pi \alpha : u \text{Type}. \varepsilon \text{Type} \alpha \rightarrow \varepsilon \text{Type} \alpha : \text{Type}
\]

It seems that \( \lambda \Pi/\text{HOL} \), just like \( \lambda U^- \), allows more functions than \( \lambda \text{HOL} \), even though the type of \( |I| \) is not the translation of a \( \lambda \text{HOL} \) type. Is that enough to make \( \lambda \Pi/\text{HOL} \) inconsistent?

### Absolute normalization vs relative normalization

One way to answer the question is to prove strong normalization of \( \lambda \Pi/S \) by constructing a model, for example in the algebra of reducibility candidates [9]. Dowek [7] recently constructed such a model for the embedding of higher-order logic (\( \lambda \text{HOL} \)) and of the calculus of constructions (\( \lambda \text{C} \)). However, this technique is still very limited. Indeed, proving such a result is, by definition, at least as hard as proving the consistency of the original system. It requires specific knowledge of \( \lambda S \) and the construction of such a model can be very involved, such as for the calculus of constructions with an infinite universe hierarchy (\( \lambda \text{C}^\infty \)).

In this paper, we take a different approach and show that \( \lambda \Pi/S \) is conservative in all cases, even when \( \lambda S \) is not normalizing. Instead of showing that \( \lambda \Pi/S \) is strongly normalizing, we show that it is weakly normalizing relative to \( \lambda S \), meaning that proofs in the target language can be reduced to proofs in the source language. That way we prove only what is needed to show conservativity, without having to prove the consistency of \( \lambda S \) all over again. After identifying the main difficulties, we characterize a PTS completion [17, 16] \( S^* \) containing \( S \), and define an inverse translation from \( \lambda \Pi/S \) to \( \lambda S^* \). We then prove that \( \lambda S^* \) is a conservative extension of \( \lambda S \) using the reducibility method [18].

### Outline

The rest of the paper is organized as follows. In Section 2, we recall the theory of pure type systems. In Section 3, we present the framework of the \( \lambda \Pi \) calculus modulo rewriting. In Section 4, we introduce Cousineau and Dowek’s embedding of functional pure type systems in the \( \lambda \Pi \) calculus modulo rewriting. In Section 5, we prove the conservativity of the embedding using the techniques mentioned above. In Section 6, we summarize the results and discuss future work.
Pure type systems [1] are a general class of typed λ-calculi parametrized by a specification.

**Definition 2.1.** A PTS specification is a triple $S = (S, A, R)$ where
- $S$ is a set of symbols called sorts
- $A \subseteq S \times S$ is a set of axioms of the form $(s_1 : s_2)$
- $R \subseteq S \times S \times S$ is a set of rules of the form $(s_1, s_2, s_3)$

We write $(s_1, s_2)$ as a short-hand for the rule $(s_1, s_2, s_2)$. The specification $S$ is functional if the relations $A$ and $R$ are functional, that is $(s_1, s_2) \in A$ and $(s_1, s'_2) \in A$ imply $s_2 = s'_2$, and $(s_1, s_2, s_3) \in R$ and $(s_1, s_2, s'_3) \in R$ imply $s_3 = s'_3$. The specification is full if for all $s_1, s_2 \in S$, there is a sort $s_3$ such that $(s_1, s_2, s_3) \in R$.

**Definition 2.2.** Given a PTS specification $S = (S, A, R)$ and a countably infinite set of variables $V$, the abstract syntax of λS is defined by the following grammar:

$$
\begin{align*}
\mathcal{T} :&= S | V | T \cdot T | \lambda V : T. T | \Pi V : T. T \\
\mathcal{C} :&:= C | V : T
\end{align*}
$$

We use lower case letters $x, y, \alpha, \beta, \ldots$ to denote variables, uppercase letters such as $M, N, A, B, \ldots$ to denote terms, and uppercase Greek letters such as $\Gamma, \Delta, \Sigma, \ldots$ to denote contexts. The set of free variables of a term $M$ is denoted by $\text{FV}(M)$. We write $A \to B$ for $\Pi x : A. B$ when $x \notin \text{FV}(B)$.

The typing rules of λS are presented in Figure 1. We write $\Gamma \vdash M : A$ instead of $\Gamma \vdash_{\lambda S} M : A$ when the context is unambiguous. We say that $M$ is a $\Gamma$-term when $\text{WF}(\Gamma)$ and $\Gamma \vdash M : A$ for some $A$. We say that $A$ is a $\Gamma$-type when $\text{WF}(\Gamma)$ and either $\Gamma \vdash A : s$ or $A = s$ for some $s \in S$. We write $\Gamma \vdash M : A : s$ as a shorthand for $\Gamma \vdash M : A \land \Gamma \vdash A : s$.

**Example 2.3.** The following well-known systems can all be expressed as functional pure type systems using the same set of sorts $S = \text{Type}, \text{Kind}$ and the same set of axioms $A = (\text{Type} : \text{Kind})$:

![Figure 1 Typing rules of λS](image-url)
Conservativity of embeddings in the $\lambda\Pi$ calculus modulo rewriting

- Simply-typed $\lambda$ calculus ($\lambda\rightarrow$):
  $\mathcal{R} = (\text{Type}, \text{Type})$

- System $\Gamma$ ($\Lambda\lambda$):
  $\mathcal{R} = (\text{Type}, \text{Type}), (\text{Kind}, \text{Type})$

- $\lambda\Pi$ calculus ($\lambda\Pi$):
  $\mathcal{R} = (\text{Type}, \text{Type}), (\text{Type}, \text{Kind})$

- Calculus of constructions ($\lambda\mathbb{C}$):
  $\mathcal{R} = (\text{Type}, \text{Type}), (\text{Kind}, \text{Type}), (\text{Type}, \text{Kind}), (\text{Kind}, \text{Kind})$

▶ Example 2.4. Let $I = \lambda_{\alpha:\text{Type}}.x : \alpha \rightarrow \alpha$ be the polymorphic identity function. The term $I$ is not well-typed in the simply typed $\lambda$ calculus but it is well-typed in the calculus of constructions $\lambda\mathbb{C}$:

\[ \vdash_{\lambda\mathbb{C}} I : \Pi_{\alpha:\text{Type}}. \alpha \rightarrow \alpha \]

The following properties hold for all pure type systems [1].

▶ Theorem 2.5 (Correctness of types). If $\Gamma \vdash_{\lambda S} M : A$ then $\text{WF}_{\lambda S}(\Gamma)$ and either $\Gamma \vdash_{\lambda S} A : s$ or $A = s$ for some $s \in S$, i.e. $A$ is a $\Gamma$-type.

The reason why we don’t always have $\Gamma \vdash_{\lambda S} A : s$ is that some sorts do not have an associated axiom, such as $\text{Kind}$ in Example 2.3, which leads to the following definition.

▶ Definition 2.6 (Top-sorts). A sort $s \in S$ is called a top-sort when there is no sort $s' \in S$ such that $(s : s') \in A$.

The following property is useful for proving properties about systems with top-sorts.

▶ Theorem 2.7 (Top-sort types). If $\Gamma \vdash_{\lambda S} A : s$ and $s$ is a top-sort then either $A = s'$ for some sort $s' \in S$ or $A = \Pi_{x:B}.C$ for some terms $B,C$.

▶ Theorem 2.8 (Confluence). If $M_1 \rightarrow_{\beta}^* M_2$ and $M_1 \rightarrow_{\beta}^* M_3$ then there is a term $M_4$ such that $M_2 \rightarrow_{\beta}^* M_4$ and $M_3 \rightarrow_{\beta}^* M_4$.

▶ Theorem 2.9 (Product compatibility). If $\Pi_{x:A}.B \equiv_{\beta} \Pi_{x:A'}.B'$ then $A \equiv_{\beta} A'$ and $B \equiv_{\beta} B'$.

▶ Theorem 2.10 (Subject reduction). If $\Gamma \vdash_{\lambda S} M : A$ and $M \rightarrow_{\beta}^* M'$ then $\Gamma \vdash_{\lambda S} M' : A$.

Finally, we state the following property for functional pure type systems.

▶ Theorem 2.11 (Uniqueness of types). Let $S$ be a functional specification. If $\Gamma \vdash_{\lambda S} M : A$ and $\Gamma \vdash_{\lambda S} M : B$ then $A \equiv_{\beta} B$.

In the rest of the paper, all the pure type systems we will consider will be functional.

3 The $\lambda\Pi$ calculus modulo rewriting

The $\lambda\Pi$ calculus, also known as LF and as $\lambda P$, is one of the simplest forms of $\lambda$ calculus with dependent types, and corresponds through the Curry-de Bruijn-Howard correspondence to a minimal first-order logic of higher-order terms. As mentioned in Example 2.3, it can be defined as the functional pure type system $\lambda\Pi$ with the following specification:

\[
\begin{align*}
\mathcal{S} & = \text{Type, Kind} \\
\mathcal{A} & = \text{Type : Kind} \\
\mathcal{R} & = (\text{Type, Type}), (\text{Type, Kind})
\end{align*}
\]
The \( \lambda \Pi \) calculus modulo rewriting extends the \( \lambda \Pi \) calculus with rewrite rules. By equating terms modulo a set of rewrite rules \( R \) in addition to \( \alpha \) and \( \beta \) equivalence, it can type more terms using the conversion rule, and therefore express theories that are more complex. The calculus can be seen as a variant of Martin-Löf’s logical framework \cite{martin-lof1980,martin-lof1984} where equalities are expressed as rewrite rules.

We recall that a rewrite rule is a triple \([\Delta] M \rightsquigarrow N\) where \( \Delta \) is a context and \( M, N \) are terms such that \( \text{FV}(N) \subseteq \text{FV}(M) \). A set of rewrite rules \( R \) induces a reduction relation on terms, written \( \rightsquigarrow_R \), defined as the smallest contextual closure such that if \([\Delta] M \rightsquigarrow N \in R\) then \( \sigma(M) \rightsquigarrow_R \sigma(N) \) for any substitution \( \sigma \) of the variables in \( \Delta \). We define the relation \( \rightsquigarrow_{\beta R} \) as \( \rightsquigarrow_{\beta} \cup \rightsquigarrow_R \), the relation \( \equiv_R \) as the smallest congruence containing \( \rightsquigarrow_R \), and the relation \( \equiv_{\beta R} \) as the smallest congruence containing \( \rightsquigarrow_{\beta R} \).

\begin{definition}
A rewrite rule \([\Delta] M \rightsquigarrow N\) is well-typed in a context \( \Sigma \) when there is a term \( A \) such that \( \Sigma, \Delta \vdash_{\lambda \Pi} M : A \) and \( \Sigma, \Delta \vdash_{\lambda \Pi} N : A \).
\end{definition}

\begin{definition}
Let \( \Sigma \) be a well-formed \( \lambda \Pi \) context and \( R \) a set of rewrite rules that are well-typed in \( \Sigma \). The \( \lambda \Pi \) calculus modulo (\( \Sigma, R \)), written \( \lambda \Pi/ (\Sigma, R) \), is defined with the same syntax as the \( \lambda \Pi \) calculus, but with the typing rules of Figure 2. We write \( \lambda \Pi/ \) instead of \( \lambda \Pi/(\Sigma, R) \) when the context is unambiguous.
\end{definition}

\begin{example}
Let \( \Sigma \) be the context

\[ \alpha : \text{Type}, c : \alpha, f : \alpha \rightarrow \text{Type} \]

and \( R \) be the following rewrite rule

\[ [\cdot] f c \rightsquigarrow_\Pi y : \alpha, f y \rightarrow f y \]

Then the term

\[ \delta = \lambda x : f c. x c x \]

\end{example}
is well-typed in $\lambda\Pi/(\Sigma, R)$:

$\triangleright_{\lambda\Pi/(\Sigma, R)} \delta : f c \rightarrow f c$

Note that the term $\delta$ would not be well-typed without the rewrite rule, even if we replace all the occurrences of $f c$ in $\delta$ by $\Pi y : a. f y \rightarrow f y$.

The system $\lambda\Pi$ is a pure type system and therefore enjoys all the properties mentioned in Section 2. The behavior of $\lambda\Pi/(\Sigma, R)$ however depends on the choice of $(\Sigma, R)$. In particular, some properties analogous to those of pure type systems depend on the confluence of the relation $\rightarrow_{\beta R}$.

- **Theorem 3.4 (Correctness of types).** If $\Gamma \vdash_{\lambda\Pi} M : A$ then $WF_{\lambda\Pi}(\Gamma)$ and either $\Gamma \vdash_{\lambda\Pi} A : s$ for some $s \in \{\text{Type}, \text{Kind}\}$ or $A = \text{Kind}$.

- **Theorem 3.5 (Top-sort types).** If $\Gamma \vdash_{\lambda\Pi} A : \text{Kind}$ then either $A = \text{Type}$ or $A = \Pi x : B. C$ for some terms $B, C$ such that $\Gamma, x : B \vdash_{\lambda\Pi} C : \text{Kind}$.

Assuming $\rightarrow_{\beta R}$ is confluent, the following properties hold [3].

- **Theorem 3.6 (Product compatibility).** If $\Pi x : A. B \equiv_{\beta R} \Pi x : A'. B'$ then $A \equiv_{\beta R} A'$ and $B \equiv_{\beta R} B'$.

- **Theorem 3.7 (Subject reduction).** If $\Gamma \vdash_{\lambda\Pi} M : A$ and $M \rightarrow_{\beta R} M'$ then $\Gamma \vdash_{\lambda\Pi} M' : A$.

- **Theorem 3.8 (Uniqueness of types).** If $\Gamma \vdash_{\lambda\Pi} M : A$ and $\Gamma \vdash_{\lambda\Pi} M : B$ then $A \equiv_{\beta R} B$.

### 4 Embedding FPTS’s in the $\lambda\Pi$ calculus modulo

In this section, we present the embedding of functional pure type systems in the $\lambda\Pi$ calculus modulo rewriting as introduced by Cousineau and Dowek [6]. In this embedding, sorts are represented as *universes à la Tarski*, as introduced by Martin-Löf [12] and later developed by Luo [11] and Palmgren [14]. The embedding is done in two steps. First, given a pure type system $\lambda S$, we construct $\lambda\Pi/S$ by giving an appropriate signature and rewrite system. Second, we define a translation from the terms and types of $\lambda S$ to the terms and types of $\lambda\Pi/S$. The proofs of the theorems in this section can be found in the original paper [6].

**Definition 4.1 (The system $\lambda\Pi/S$).** Consider a functional pure type system specified by $S = (\mathcal{S}, \mathcal{A}, \mathcal{R})$. Define $\Sigma_S$ to be the well-formed context containing the declarations:

- $u_s : \text{Type}$ for all $s \in \mathcal{S}$
- $\varepsilon_s : u_s \rightarrow \text{Type}$ for all $s \in \mathcal{S}$
- $s_1 : u_{s_2}$ for all $s_1, s_2 \in \mathcal{A}$
- $\hat{s}_{s_1 s_2 s_3} : \Pi \alpha : u_{s_1}, (\varepsilon_{s_1} \alpha \rightarrow u_{s_2}) \rightarrow u_{s_3}$ for all $s_1, s_2, s_3 \in \mathcal{R}$

Let $R_S$ be the well-typed rewrite system containing the rules

$[\cdot] \varepsilon_{s_2} s_1 \rightsquigarrow u_{s_1}$

for all $s_1 : s_2 \in \mathcal{A}$, and

$[\Delta_{s_1 s_2 s_3}] \varepsilon_{s_2} (\hat{s}_{s_1 s_2 s_3} A B) \rightsquigarrow \Pi x : (\varepsilon_{s_2} (B x))$ for all $(s_1, s_2, s_3) \in \mathcal{R}$, where $\Delta_{s_1 s_2 s_3} = (A : u_{s_1}, B : (\varepsilon_{s_2} \alpha \rightarrow u_{s_2}))$. The system $\lambda\Pi/S$ is defined as the $\lambda\Pi$ calculus modulo $(\Sigma_S, R_S)$, that is, $\lambda\Pi/(\Sigma_S, R_S)$. 
Theorem 4.2 (Confluence). The relation $\rightarrow_{\beta R}$ is confluent.

The translation is composed of two functions, one from the terms of $\lambda S$ to the terms of $\lambda \Pi / S$, the other from the types of $\lambda S$ to the types of $\lambda \Pi / S$.

Definition 4.3. The translation $|M|_\Gamma$ of $\Gamma$-terms and the translation $\|A\|_\Gamma$ of $\Gamma$-types are mutually defined as follows.

$$
|s|_\Gamma = s \\
|x|_\Gamma = x \\
|M N|_\Gamma = |M|_\Gamma |N|_\Gamma \\
|\lambda x : A . M|_\Gamma = \lambda x : \|A\|_\Gamma . |M|_{\Gamma , x : A} \\
|\Pi x : A . B|_\Gamma = \pi_{s_1, s_2, s_3} |A|_\Gamma (\lambda x : \|A\|_\Gamma . |B|_{\Gamma , x : A}) \\
$$

where $\Gamma \vdash A : s_1$

and $\Gamma , x : A \vdash B : s_2$

and $(s_1, s_2, s_3) \in R$

$$
\|s\|_\Gamma = u_s \\
\|\Pi x : A . B|_\Gamma = \Pi x : \|A\|_\Gamma . \|B\|_{\Gamma , x : A} \\
\|A\|_\Gamma = \varepsilon_s |A|_\Gamma \text{ where } \Gamma \vdash A : s
$$

Note that this definition is redundant but it is well-defined up to $\equiv_{\beta R}$. In particular, because some $\Gamma$-types are also $\Gamma$-terms, there are two ways to translate them, but they are equivalent:

$$
\varepsilon_{s_2} s_1 \equiv_{\beta R} u_{s_1} \\
\varepsilon_{s_3} |\Pi x : A . B|_\Gamma \equiv_{\beta R} \Pi x : \|A\|_\Gamma . \|B\|_{\Gamma , x : A}
$$

This definition is naturally extended to well-formed contexts as follows.

$$
\|\cdot\| = \cdot \\
\|\Gamma , x : A\| = \|\Gamma\| , x : \|A\|_\Gamma 
$$

Example 4.4. The polymorphic identity function of the Calculus of constructions $\lambda C$ is translated as

$$
|I| = \lambda \alpha : u_{\text{Type}} . \lambda x : \varepsilon_{\text{Type}} \alpha . x
$$

and its type $A = \Pi \alpha : \text{Type} . \alpha \rightarrow \alpha$ is translated as

$$
|A| = \pi_{\text{Kind}, \text{Type}, \text{Type}} \text{Type} (\lambda \alpha : u_{\text{Type}} . |A\alpha|)
$$

where $A\alpha = \alpha \rightarrow \alpha$ and

$$
|A\alpha| = \hat{\pi}_{\text{Type}, \text{Type}} \alpha (\lambda x : \varepsilon_{\text{Type}} \alpha . \varepsilon_{\text{Type}} \alpha)
$$

The identity function applied to itself is translated as

$$
|I A I| = |I| |A| |I|
$$

The embedding is complete, in the sense that all the typing relations of $\lambda S$ are preserved by the translation.

Theorem 4.5 (Completeness). For any context $\Gamma$ and terms $M$ and $A$, if $\Gamma \vdash_{\lambda S} M : A$ then $\|\Gamma\| \vdash_{\lambda \Pi / S} |M|_\Gamma : \|A\|_\Gamma$. 


5 Conservativity

In this section, we prove the converse of the completeness property. One could attempt to prove that if \( \parallel M \parallel_{\Pi/S} : \parallel A \parallel_{\Gamma} \) then \( \Gamma \vdash_{\Pi/S} M : A \). However, that would be too weak because the translation \( |M|_\Gamma \) is only defined for well-typed terms. A second attempt would be to define inverse translations \( \varphi(M) \) and \( \psi(A) \) and prove that if \( \Gamma \vdash_{\Pi/S} M : A \) then \( \psi(\Gamma) \vdash_{\lambda S} \varphi(M) : \psi(A) \), but that would not work either because not all terms and types of \( \Pi/S \) correspond to valid terms and types of \( \lambda S \), as was shown in Example 1.1. Therefore the property that we want to prove is: if there is a term \( N \) such that \( \parallel \Gamma \parallel_{\Pi/S} N : \parallel A \parallel_{\Gamma} \) then there is a term \( M \) such that \( \Gamma \vdash_{\lambda S} M : A \).

The main difficulty is that some of these external terms can be involved in witnessing valid \( \lambda S \) types, as illustrated by the following example.

Example 5.1. Consider the context \( \text{nat} : \text{Type} \). Even though the polymorphic identity function \( I \) and its type are not well-typed in \( \lambda \text{HOL} \), they can be used in \( \lambda \Pi/\text{HOL} \) to construct a witness for \( \text{nat} \rightarrow \text{nat} \).

\[
\text{nat} : \text{nat} \rightarrow \text{nat} \qquad (\text{nat} = \lambda \text{HOL}. (\text{nat} \rightarrow \text{nat}))
\]

We can normalize the term \( |I| \text{nat} \rightarrow \lambda \text{HOL}. (\text{nat} \rightarrow \text{nat}) \) to \( \lambda x : \text{nat}. x \) which is a term that corresponds to a valid \( \lambda \text{HOL} \) term: it is the translation of the term \( \lambda x : \text{nat}. x \). However, as discussed previously, we cannot restrict ourselves to normal terms because we do not know if \( \lambda \Pi/S \) is normalizing.

To prove conservativity, we will therefore need to address the following issues:

1. The system \( \Pi/S \) can type more terms than \( \lambda S \).
2. These terms can be used to construct proofs for the translation of \( \lambda S \) types.
3. The \( \Pi/S \) terms that inhabit the translation of \( \lambda S \) types can be reduced to the translation of \( \lambda S \) terms.

We will proceed as follows. First, we will eliminate \( \beta \)-redexes at the level of \( \text{Kind} \) by reducing \( \Pi/S \) to a subset \( \Pi^-/S \). Then, we will extend \( \lambda S \) to a minimal completion \( \lambda S^* \) that can type more terms than \( \lambda S \), and show that \( \Pi^-/S \) corresponds to \( \lambda S^* \) using inverse translations \( \varphi(M) \) and \( \psi(A) \). Finally, we will show that \( \lambda S^* \) terms inhabiting \( \lambda S \) types can be reduced to \( \lambda S \) terms. The procedure is summarized in the following diagram.

\[
\begin{array}{c}
\lambda \Pi/S \quad \text{(Lemma 5.3)} \\
\downarrow \beta^* \\
\lambda \Pi^-/S \\
\lambda S \quad \text{(Lemma 5.22)} \\
\downarrow \beta^* \\
\lambda S^* \\
\end{array}
\]

\[\text{(Theorem 4.5)} \quad |M|_{\Pi/S} \quad \parallel M \parallel_{\Pi/S} \quad \varphi(M) \quad \psi(A) \quad \text{(Lemma 5.14)}\]

5.1 Eliminating \( \beta \)-redexes at the level of \( \text{Kind} \)

In \( \Pi/S \), we can have \( \beta \)-redexes at the level of \( \text{Kind} \) such as \( \lambda (\lambda x : A. u_x). M \). These redexes are artificial and are never generated by the forward translation of any PTS. We show here that they can always be safely eliminated.

Definition 5.2. A \( \Gamma \)-term \( M \) of type \( C \) is at the level of \( \text{Kind} \) (resp. \( \text{Type} \)) if \( \Gamma \vdash C : \text{Kind} \) (resp. \( \Gamma \vdash C : \text{Type} \)). We define \( \Pi^-/S \) terms as the subset of well-typed \( \Pi/S \) terms that do not contain any Kind-level \( \beta \)-redexes.
Lemma 5.3. For any $\lambda\Pi/S$ context $\Gamma$ and $\Gamma$-term $M$, there is a $\lambda\Pi^-/S$ term $M^-$ such that $M \rightarrow^\ast_{\beta^d} M^-$. 

Proof. Reducing a Kind-level $\beta$-redex ($\lambda x : A. B) N$ does not create other Kind-level $\beta$-redexes because $N$ is at the level of Type. Indeed, in the $\lambda\Pi$ calculus modulo rewriting the only Kind rule is (Type, Kind, Kind). Therefore $N : A : Type$. If $N$ reduces to a $\lambda$-abstraction then the only redexes it can create are at the level of Type. Therefore, the number of Kind-level $\beta$-redexes strictly decreases, so any Kind-level $\beta$-reduction strategy will terminate. 

Example 5.4. The term $I_1 = \lambda \alpha : u : Type. \lambda x : \varepsilon : Type. ((\lambda \beta : u : Type. \beta) \alpha). x$ is in $\lambda\Pi^-/HOL$. The term $I_2 = \lambda \alpha : u : Type. \lambda x : ((\lambda \beta : u : Type. \varepsilon : Type. \beta) \alpha). x$ is not in $\lambda\Pi^-/HOL$ but $I_2 \rightarrow_{\beta} \lambda \alpha : u : Type. \lambda x : \varepsilon : Type. \alpha. x$ which is in $\lambda\Pi^-/HOL$.

5.2 Minimal completion

To simplify our reducibility proof in the next section, we will translate $\lambda\Pi/S$ back to a pure type system, but since it cannot be $\lambda S$ we will define a slightly larger PTS called $\lambda S^*$ that contains $\lambda S$ and that will be easier to manipulate than $\lambda\Pi/S$.

The reason we need a larger PTS is that we have types that do not have a type, such as top-sorts because there is no associated axiom. Similarly, we can sometimes prove $\Gamma, x : A \vdash_{\lambda S} M : B$ but cannot abstract over $x$ because there is no associated product rule. Completions of pure type systems were originally introduced by Severi [17, 16] to address these issues by injecting $\lambda S$ into a larger pure type system.

Definition 5.5 (Completion [16]). A specification $S' = (S', A', R')$ is a completion of $S$ if

1. $S \subseteq S', A \subseteq A', R \subseteq R'$, and
2. for all sorts $s_1 \in S$, there is a sort $s_2 \in S'$ such that $(s_1 : s_2) \in A'$, and
3. for all sorts $s_1, s_2 \in S'$, there is a sort $s_3 \in S'$ such that $(s_1, s_2, s_3) \in R'$.

Notice that all the top sorts of $\lambda S$ are typable in $\lambda S'$ and that $\lambda S'$ is full, meaning that all products are typable. These two properties reflect exactly the discrepancy between $\lambda S$ and $\lambda\Pi^-/S$. Not all completions are conservative though, so we define the following completion.

Definition 5.6 (Minimal completion). We define the minimal completion of $S$, written $S^*$, to be the following specification:

$$
S^* = S \cup \{\tau\}
$$

$$
A^* = A \cup \{(s_1 : \tau) | s_1 \in S, \exists s_2, (s_1 : s_2) \in A\}
$$

$$
R^* = R \cup \{(s_1, s_2, \tau) | s_1, s_2 \in S^*, \exists s_3, (s_1, s_2, s_3) \in R\}
$$

where $\tau \not\in S$. 

We add a new top-sort $\tau$ and axioms $s : \tau$ for all previous top-sorts $s$, and complete the rules to obtain a PTS full. The new system is a completion by Definition 5.5 and it is minimal in the sense that we generically added the smallest number of sorts, axioms, and rules so that the result is guaranteed to be conservative. Any well-typed term of $\lambda S$ is also well-typed in $\lambda S^*$, but just like $\lambda \Pi^{-}/S$, this system allows more functions than $\lambda S$.

**Example 5.7.** The polymorphic identity function is well-typed in $\lambda HOL^*$.

\[ \Gamma \vdash_{\lambda HOL^*} I : \Pi \alpha : \text{Type}. \alpha \to \alpha \]

\[ \Gamma \vdash_{\lambda HOL^*} \Pi \alpha : \text{Type}. \alpha \to \alpha : \tau \]

Next, we define inverse translations that translate the terms and types of $\lambda \Pi^{-}/S$ to the terms and types of $\lambda S^*$.

**Definition 5.8 (Inverse translations).** The inverse translation of terms $\varphi(M)$ and the inverse translation of types $\psi(A)$ are mutually defined as follows.

\[
\varphi(s) = s \\
\varphi(\pi_{s_1,s_2,s_3}) = \lambda \alpha : s_1. \lambda \beta : (\alpha \to s_2). \Pi x : \alpha. \beta x \\
\varphi(x) = x \\
\varphi(M N) = \varphi(M) \varphi(N) \\
\varphi(\lambda x : A. M) = \lambda x : \psi(A). \varphi(M)
\]

\[
\psi(u_s) = s \\
\psi(\varepsilon_s M) = \varphi(M) \\
\psi(\Pi x : A. B) = \Pi x : \psi(A). \psi(B)
\]

Note that this is only a partial definition, but it is total for $\lambda \Pi^{-}/S$ terms. In particular, it is an inverse of the forward translation in the following sense.

**Lemma 5.9.** For any $\Gamma$-term $M$ and $\Gamma$-type $A$,
1. $\varphi([M]_\Gamma) \equiv_\beta M$,
2. $\psi([|A|]_\Gamma) \equiv_\beta A$.

**Proof.** By induction on $M$ or $A$. We show the product case where $M = \Pi x : A. B$. By induction hypothesis, $\varphi([|A|]) \equiv_\beta A$ and $\varphi([|B|]) \equiv_\beta B$. Therefore

\[
\varphi([M]) = (\lambda \alpha. \lambda \beta. \Pi x : \alpha. \beta x) \varphi([|A|]) (\lambda x. \varphi([|B|])) \\
\rightarrow_\beta^* \Pi x : \varphi([|A|]). \varphi([|B|]) \\
\equiv_\beta \Pi x : A. B
\]

Next we show that the inverse translations preserve typing.

**Lemma 5.10.**
1. $\varphi(M[x \backslash N]) = \varphi(M)[x \backslash \varphi(N)]$
2. $\psi(A[x \backslash N]) = \psi(A)[x \backslash \varphi(N)]$

**Proof.** By induction on $M$ or $A$. We show the product case $A = \Pi y : B. C$. Without loss of generality, $y \neq x$ and $y \notin N$ and $y \notin \varphi(N)$. Then $\Pi y : B. C[x \backslash N] = \Pi y : B[x \backslash N]. C[x \backslash N]$. 
By induction hypothesis, $\psi(B[x \setminus N]) = \psi(B)[x \upharpoonright \varphi(N)]$ and $\psi(C[x \setminus N]) = \psi(C)[x \upharpoonright \varphi(N)]$. Therefore

$$
\psi(A[x \setminus N]) = \Pi y: \psi(B)[x \upharpoonright \varphi(N)], \psi(C)[x \upharpoonright \varphi(N)]
= \Pi x: \psi(B), \psi(C)[x \upharpoonright \varphi(N)]
= \psi(\Pi x: B. C)[x \upharpoonright \varphi(N)]
$$

► **Lemma 5.11.**

1. If $M \rightarrow_{\beta R} N$ then $\varphi(M) \rightarrow_{\beta} \varphi(N)$
2. If $A \rightarrow_{\beta R} B$ then $\psi(A) \rightarrow_{\beta} \psi(B)$

**Proof.** By induction on $M$ or $A$. We show the base cases.
- Case $M = (\lambda x: A_1. M_1) N_1$, $N = M_1[x \setminus N_1]$. Then $\varphi(M) = (\lambda x: \psi(A_1). \varphi(M_1)) \varphi(N_1)$.
  Therefore $\varphi(M) \rightarrow_{\beta} \varphi(M_1)[x \upharpoonright \varphi(N_1)]$ which equals $\varphi(M_1[x \setminus N_1])$ by Lemma 5.10.
- Case $A = \epsilon s, B = u_s$. Then $\psi(A) = s = \psi(B)$.
- Case $A = \epsilon s_1, (s_1s_2s_3 A_1 B_1), B = \Pi x: \epsilon s_1 A_1. \epsilon s_2 (B_1 x)$. Then

$$
\psi(A) = (\lambda o: \lambda \beta. \Pi x: \alpha. \beta x) \varphi(A_1) \varphi(B_1)
\rightarrow_{\beta} \Pi x: \varphi(A_1). \varphi(B_1) x
= \psi(\Pi x: A_1. B_1 x)
$$

► **Lemma 5.12.**

1. If $M \equiv_{\beta R} N$ then $\varphi(M) \equiv_{\beta} \varphi(N)$
2. If $A \equiv_{\beta R} B$ then $\psi(A) \equiv_{\beta} \psi(B)$

**Proof.** Follows from Lemma 5.11.

Because the forward translation of contexts does not introduce any type variable, we define the following restriction on contexts.

► **Definition 5.13** (Object context). We say that $\Gamma$ is an object context if $\Gamma \vdash_{\lambda S/S} A : \text{Type}$ for all $x : A \in \Gamma$. If $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ is an object context, we define $\psi(\Gamma)$ as $(x_1 : \psi(A_1), \ldots, x_n : \psi(A_n))$.

► **Lemma 5.14.** For any $\lambda_{\Pi/S}$ object context $\Gamma$ and terms $M, A$:

1. If $\text{WF}_{\lambda_{\Pi/S}}(\Gamma)$ then $\text{WF}_{\lambda S^*}(\psi(\Gamma))$.
2. If $\Gamma \vdash_{\lambda_{\Pi/S}} M : A : \text{Type}$ then $\psi(\Gamma) \vdash_{\lambda S^*} \varphi(M) : \psi(A)$.
3. If $\Gamma \vdash_{\lambda_{\Pi/S}} A : \text{Type}$ then $\psi(\Gamma) \vdash_{\lambda S^*} \psi(A) : s$ for some sort $s \in S^*$.

**Proof.** By induction on the derivation. The details of the proof can be found in the Appendix.

### 5.3 Reduction to $\lambda S$

In order to show that $\lambda S^*$ is a conservative extension of $\lambda S$, we prove that $\beta$-reduction at the level of $\tau$ terminates. A straightforward proof by induction would fail because contracting a $\tau$-level $\beta$-redex can create other such redexes. To solve this, we adapt Tait’s reducibility method [18]. The idea is to strengthen the induction hypothesis of the proof by defining a predicate by induction on the type of the term.
Conservativity of embeddings in the $\lambda$ calculus modulo rewriting

Definition 5.15. The predicate $\Gamma \vdash_s M : A$ is defined as $\WF_{\lambda S}(\Gamma)$ and $\Gamma \vdash_{\lambda S} M : A : s$ for some sort $s$ and:
- if $s \neq \tau$ or $A = s'$ for some $s' \in S$ then $\Gamma \vdash_s M : A$ iff $M \rightarrow_{\beta} M'$ and $A \rightarrow_{\beta} A'$ for some $M', A'$ such that $\Gamma \vdash_{\lambda S} M' : A'$.
- if $s = \tau$ and $A = \Pi x : B. C$ for some $B, C$ then $\Gamma \vdash_s M : A$ iff for all $N$ such that $\Gamma \vdash_s N : B$, $\Gamma \vdash_s M N : C[x\backslash N]$.

Note that recursive definition covers all cases thanks to Theorem 2.7. To show that it is well-founded, we define the following measure of $A$.

Definition 5.16. If $\WF_{\lambda S}(\Gamma)$ and $\Gamma \vdash_{\lambda S} A : s$ then $\mathcal{H}_s(A)$ is defined as:

\[
\begin{align*}
\mathcal{H}_s(A) &= 0 & \text{if } s \neq \tau \\
\mathcal{H}_s(s') &= 0 & \text{if } s = \tau \\
\mathcal{H}_s(\Pi x : B. C) &= 1 + \max(\mathcal{H}_s(B) + \mathcal{H}_s(C)) & \text{if } s = \tau
\end{align*}
\]

Lemma 5.17. If $\Gamma, x : B \vdash_{\lambda S} C : \tau$ and $\Gamma \vdash_{\lambda S} N : B$ then $\mathcal{H}_s(C[x\backslash N]) = \mathcal{H}_s(C)$.

Proof. By induction on $C$.

Corollary 5.18. Definition 5.15 is well-founded.

Proof. The measure $\mathcal{H}_s(A)$ strictly decreases in the definition.

The predicate we defined is compatible with $\beta$-equivalence.

Lemma 5.19. If $\Gamma \vdash_s M : A$ and $\Gamma \vdash_{\lambda S} M' : A$ and $M \equiv_{\beta} M'$ then $\Gamma \vdash_s M' : A$.

Proof. By induction on the height of $A$.

Lemma 5.20. If $\Gamma \vdash_s M : A$ and $\Gamma \vdash_{\lambda S} A' : s$ and $A \equiv_{\beta} A'$ then $\Gamma \vdash_s M : A'$.

Proof. By induction on the height of $A$.

We extend the definition of the inductive predicate to contexts and substitutions before proving the main general lemma.

Definition 5.21. If $\WF_{\lambda S}(\Gamma)$, $\WF_{\lambda S}(\Gamma')$, and $\sigma$ is a substitution for the variables of $\Gamma$, then $\Gamma' \vdash_s \sigma : \Gamma$ when $\Gamma' \vdash_s \sigma(x) : \sigma(A)$ for all $(x : A) \in \Gamma$. 
Lemma 5.22. If $\Gamma \vdash_{\lambda S} M : A : s$ then for any context $\Gamma'$ and substitution $\sigma$ such that $WF_{\lambda S}(\Gamma')$ and $\Gamma' \models_{S} \sigma : \Gamma$, $\Gamma' \models_{S} \sigma(M) : \sigma(A)$.

Proof. By induction on the derivation of $\Gamma \vdash_{\lambda S} M : A$. The details of the proof can be found in the Appendix.

Corollary 5.23. Suppose $WF_{\lambda S}(\Gamma)$ and either $\Gamma \vdash_{\lambda S} A : s$ or $A = s$ for some $s \in S$. If $\Gamma \vdash_{\lambda S} M : A$ then $M \rightarrow_{\beta}^{\ast} M'$ such that $\Gamma \vdash_{\lambda S} M' : A$.

Proof. Taking $\sigma$ as the identity substitution, there are terms $M'$ and $A'$ such that $M \rightarrow_{\beta}^{\ast} M'$ and $A \rightarrow_{\beta}^{\ast} A'$ and $\Gamma \vdash_{\lambda S} M' : A'$. If $A = s \in S$ then $A' = s$ and we are done. Otherwise by conversion we get $\Gamma \vdash_{\lambda S} M' : A$.

We now have all the tools to prove the main theorem.

Theorem 5.24 (Conservativity). For any $\Gamma$-type $A$ of $\lambda S$, if there is a term $N$ such that $[\Gamma] \vdash_{\lambda \Pi/S} N : [A]_{\Gamma}$ then there is a term $M$ such that $\Gamma \vdash_{\lambda S} M : A$.

Proof. By Lemma 5.3, there is a $\lambda \Pi^{-}/S$ term $N^{-}$ such that $N \rightarrow_{\beta}^{\ast} N^{-}$. By subject reduction, $[\Gamma] \vdash_{\lambda \Pi/S} N^{-} : [A]_{\Gamma}$. By Lemmas 5.14 and 5.9, $\Gamma \vdash_{\lambda S} \varphi(N^{-}) : A$. By Corollary 5.23, there is a term $M$ such that $\varphi(N^{-}) \rightarrow_{\beta}^{\ast} M$ and $\Gamma \vdash_{\lambda S} M : A$.

6 Conclusion

We have shown that $\lambda \Pi/S$ is conservative even when $\lambda S$ is not normalizing. Even though $\lambda \Pi/S$ can construct more functions than $\lambda S$, it preserves the semantics of $\lambda S$. This effect is similar to various conservative extensions of pure type systems such as pure type systems with definitions [17], pure type systems without the $\Pi$-condition [16], or predicative (ML) polymorphism [15]. Inconsistency in pure type systems usually does not come from the ability to type more functions, but from the possible impredicativity caused by assigning a sort to the type of these functions. It is clear that no such effect arises in $\lambda \Pi/S$ because there is no constant $\pi_{s_{1},s_{2}}$, associated to the type of illegal abstractions.

One could ask whether the techniques we used are adequate. While the construction of $\lambda S^{\ast}$ is not absolutely necessary, we feel that it simplifies the proof and that it helps us better understand the behavior of $\lambda \Pi/S$ by reflecting it back into a pure type system. The relative normalization steps of Section 5.3 correspond to the normalization of a simply typed $\lambda$ calculus. Therefore, it is not surprising that we had to use Tait’s reducibility method. However, our proof can be simplified in some cases. A PTS is complete when it is a completion of itself. In that case, the construction of $S^{\ast}$ is unnecessary. The translations $\varphi(M)$ and $\psi(A)$ translate directly into $\lambda S$, and Section 5.3 can be omitted. This is the case for example for the calculus of constructions with infinite type hierarchy ($\lambda C^{\infty}$) [17], which is the basis for proof assistants such as Coq and Matita.

The results of this paper can be extended in several directions. They could be adapted to show the conservativity of other embeddings, such as that of the calculus of inductive constructions (CIC) [4]. They also indirectly imply that $\lambda \Pi/S$ is weakly normalizing when $\lambda S$ is weakly normalizing because the image of a $\lambda S$ term is normalizing [6]. The strong normalization of $\lambda \Pi/S$ when $\lambda S$ is strongly normalizing is still an open problem. The Barendregt-Geuvers-Klop conjecture states that any weakly normalizing PTS is also strongly normalizing [8]. There is evidence that this conjecture is true [2], in which case we hope that its proof could be adapted to prove the strong normalization of $\lambda \Pi/S$. Weak normalization could also be used as an intermediary step for constructing models by induction on types in order to prove strong normalization.
Conservativity of embeddings in the $\lambda\Pi$ calculus modulo rewriting

Acknowledgments

We thank Gilles Dowek and Guillaume Burel for their support and feedback, as well as Frédéric Blanqui, Raphaël Cauderlier, and the various anonymous referees for their comments and suggestions on previous versions of this paper.

References

Appendix

Proof details

Lemma (5.14). For any \(\lambda\Pi^-/S\) object context \(\Gamma\) and terms \(M, A\):

1. If \(WF_{\lambda\Pi^-/S}(\Gamma)\) then \(WF_{\lambda\Pi^+/S}(\psi(\Gamma))\).
2. If \(\Gamma \vdash_{\lambda\Pi^-/S} M : \text{Type} \) then \(\psi(\Gamma) \vdash_{\lambda\Pi^+/S} \varphi(M) : \psi(A)\).
3. If \(\Gamma \vdash_{\lambda\Pi^-/S} A : \text{Type} \) then \(\psi(\Gamma) \vdash_{\lambda\Pi^+/S} \psi(A) : s \) for some sort \(s \in S^*\).

Proof. By induction on the derivation.

1. There are 2 cases.

   - **Empty**
     
     \[\text{Empty}\]
     
     Then \(WF(\cdot)\) trivially.

   - **Declaration**
     
     \[WF(\Gamma) \quad \Gamma \vdash A : \text{Type} \quad x \notin \Sigma, \Gamma\]
     
     Then \(x \notin \psi(\Gamma)\). By induction hypothesis, \(WF(\psi(\Gamma))\) and \(\psi(\Gamma) \vdash \psi(A) : s\) for some sort \(s \in S^*\). Therefore \(WF(\psi(\Gamma), x : \psi(A))\).

2. There are 4 cases.

   - **Variable**
     
     \[WF(\Gamma) \quad (x : A) \in \Sigma, \Gamma\]
     
     \[\Gamma \vdash x : A\]
     
     By induction hypothesis, \(WF(\psi(\Gamma))\).

     a. If \(x = s_1\) then \(A = u_{s_2}\) and \((s_1 : s_2) \in A\). Therefore \(\psi(\Gamma) \vdash s_1 : s_2\).

     b. If \(x = \pi_{s_1, s_2, s_3}\) then \(A = \Pi u : u_{s_1}.(\varepsilon_{s_1} \alpha \to u_{s_2}) \to u_{s_3}\) and \((s_1, s_2, s_3) \in R\).

     Therefore \(\psi(\Gamma), \alpha : s_1, \beta : \alpha \to s_2 \vdash \Pi \alpha : \alpha.\beta x : s_3\), which implies \(\psi(\Gamma) \vdash (\lambda \alpha : s_1.\lambda \beta : (\alpha \to s_2).\Pi \alpha : \alpha.\beta x) : \Pi \alpha : s_1. (\alpha \to s_2) \to s_3\).

     c. Otherwise \((x : A) \in \Gamma\), so \((x : \psi(A)) \in \psi(\Gamma)\). By induction hypothesis, \(WF(\psi(\Gamma))\).

     Therefore \(\psi(\Gamma) \vdash x : \psi(A)\).

   - **Application**
     
     \[\Gamma \vdash M : \Pi x : A. B \quad \Gamma \vdash N : A\]
     
     \[\Gamma \vdash M N : B[x \{N\}]\]
     
     By induction hypothesis, \(\psi(\Gamma) \vdash \varphi(M) : \Pi x : \psi(A).\psi(B)\) and \(\psi(\Gamma) \vdash \varphi(N) : \psi(A)\).

     Therefore \(\psi(\Gamma) \vdash \varphi(M) \varphi(N) : \psi(B)[x \{\varphi(N)\}]\). By Lemma 5.10, \(\psi(\Gamma) \vdash \varphi(M) \varphi(N) : \psi(B[x \{N\}]\)

   - **Abstraction**
     
     \[\Gamma \vdash \Pi x : A. B : \text{Type} \quad \Gamma, x : A \vdash M : B\]
     
     \[\Gamma \vdash \lambda x : A. M : \Pi x : A. B\]
     
     By induction hypothesis, \(\psi(\Gamma) \vdash \Pi x : \psi(A).\psi(B) : s\) and \(\psi(\Gamma), x : \psi(A) \vdash \varphi(M) : \psi(B)\) for some sort \(s \in S^*\). Therefore \(\psi(\Gamma) \vdash (\lambda x : \psi(A).\varphi(M)) : \Pi x : \psi(A).\psi(B)\).
There are 4 cases.

**Variable**

\[
\text{WF}(\Gamma) \quad ((x : \text{Type}) \in \Sigma, \Gamma)
\]

Since \(\Gamma\) is an object context we must have \(x \in \Sigma\), so \(x = u_{s_1}\) for some \(s_1 \in \mathcal{S}\). By induction hypothesis, \(\text{WF}(\Gamma)\). By definition, there is a sort \(s_2 \in \mathcal{S}^*\) such that \((s_1 : s_2) \in \mathcal{A}^*\). Therefore \(\psi(\Gamma) \vdash s_1 : s_2\).

**Application**

\[
\Gamma \vdash M : \Pi x : A. B \quad \Gamma \vdash N : A
\]

Since \(\Gamma\) is an object context and \(M N\) is not a \(\beta\)-redex, we must have \(M = \varepsilon_{s_1}\) and \(\Pi x : A. B = u_{s_1} \rightarrow \text{Type}\) and \(N : u_{s_1}\) for some \(s_1 \in \mathcal{S}\). By induction hypothesis, \(\psi(\Gamma) \vdash \phi(N) : s_1\).

**Product**

\[
\Gamma \vdash A : \text{Type} \quad \Gamma, x : A \vdash B : \text{Type}
\]

By induction hypothesis, \(\psi(\Gamma) \vdash \phi(A) : s_1\) and \(\psi(\Gamma), x : \phi(A) \vdash \phi(B) : s_2\) for some sorts \(s_1, s_2 \in \mathcal{S}^*\). By definition, there is a sort \(s_3 \in \mathcal{S}^*\) such that \((s_1, s_2, s_3) \in \mathcal{R}^*\). Therefore \(\psi(\Gamma) \vdash (\Pi x : \phi(A). \phi(B)) : s_3\).

**Conversion**

\[
\Gamma \vdash A : B \quad \Gamma \vdash B : \text{Kind} \quad B \equiv_{\beta R} \text{Type}
\]

We must have \(B = \text{Type}\). By induction hypothesis, \(\psi(\Gamma) \vdash \phi(A) : s\) for some sort \(s \in \mathcal{S}^*\).

---

**Lemma (5.22).** If \(\Gamma \vdash_{\Lambda S} M : A : s\) then for any context \(\Gamma'\) and substitution \(\sigma\) such that \(\text{WF}_{\Lambda S}(\Gamma')\) and \(\Gamma' \models_S \sigma : \Gamma, \Gamma' \models_S \sigma(M) : \sigma(A)\).

**Proof.** By induction on the derivation of \(\Gamma \vdash_{\Lambda S} M : A\).

- **Sort**
  \[
  \text{WF}(\Gamma) \quad (s_1 : s_2) \in \mathcal{A}^*
  \]
  Since \(s_2 : s_1\), we must have \(s_2 \neq \tau\), so \((s_1 : s_2) \in \mathcal{A}\). Therefore \(\Gamma' \vdash_{\Lambda S} s_1 : s_2\), which implies \(\Gamma' \models_S s_1 : s_2\).

- **Variable**
  \[
  \Gamma \vdash x : A
  \]
  Then \(\Gamma' \models_S \sigma(M) : \sigma(A)\) by definition of \(\Gamma' \models_S \sigma : \Gamma\).
A. Assaf 17

Without loss of generality, \( x \not\in \Gamma' \), so \( \sigma(B[x\setminus N]) = \sigma(B)[x\setminus \sigma(N)] \). By induction hypothesis, \( \Gamma' \models_S \sigma(M) : \Pi x : \sigma(A). B \) and \( \Gamma' \models_S \sigma(N) : \sigma(A) \).

1. If \( \Gamma \vdash_{\lambda S} A : s_3 \neq \tau \) then \( \Gamma \vdash_{\lambda S} \Pi x : A : s_1 \) and \( \Gamma, x : A \vdash_{\lambda S} B : s_2 \) for some \( s_1, s_2, s_3 \) such that \( (s_1, s_2, s_3) \in S \), which also means that \( \Gamma \vdash_{\lambda S} B[x\setminus N] : s_2 \neq \tau \). By induction hypothesis, \( \sigma(M) \rightarrow^\ast_{\beta} M', \sigma(A) \rightarrow A' \) and \( \sigma(B) \rightarrow B' \) such that \( \Gamma' \vdash_{\lambda S} \Pi x : A'. B' \) and \( \sigma(N) \rightarrow N' \), \( \sigma(A) \rightarrow A'' \) such that \( \Gamma' \vdash_{\lambda S} N' : A'' \). By confluence and subject reduction, we can assume \( A' = A'' \). Therefore \( \Gamma' \vdash_{\lambda S} A'. B' : B'[x\setminus N] \). Since \( B[x\setminus N] \rightarrow^* B'[x\setminus N]' \), this implies \( \Gamma' \models_S M : B[x\setminus N] \).

2. Otherwise \( \Gamma \vdash_{\Pi x : A. B : \tau} \). By definition, \( \Gamma' \models_S \sigma(M) : \sigma(N) : \sigma(B)[x\setminus \sigma(N)] \).

Without loss of generality, \( x \not\in \Gamma' \).

1. If \( s \neq \tau \) then by induction hypothesis, \( \sigma(A) \rightarrow A' \) and \( \sigma(B) \rightarrow B' \) such that \( \Gamma' \vdash_{\lambda S} A' : s_1 \) for some \( s_1 \neq \tau \), so \( \Gamma \models_S A : s_1 \), which implies \( \Gamma', x : A' \models_S \sigma : (\Gamma, x : A) \). By induction hypothesis, \( \sigma(M) \rightarrow M' \) and \( \sigma(B) \rightarrow B'' \) such that \( \Gamma', x : A' \vdash_{\lambda S} M' : B'' \). By confluence and subject reduction, we can assume \( B' = B'' \). Therefore \( \Gamma' \vdash_{\lambda S} (\lambda x : A'. M') : \Pi x : A'. B' \), which implies \( \Gamma' \models_S (\lambda x : A. M) : \Pi x : A. B \).

2. If \( s = \tau \) then for all \( N \) such that \( \Gamma' \models_S N : \sigma(A) \), we have \( \Gamma \models_S (\sigma, N/x) : (\Gamma, x : A) \). By induction hypothesis, \( \Gamma' \models_S (\sigma, N/x)(M) : (\sigma, N/x)(B) \). Since \( x \not\in \Gamma' \), we have \( (\sigma, N/x)(M) = \sigma(M)[x\setminus N] \) and \( (\sigma, N/x)(B) = \sigma(B)[x\setminus N] \). Therefore \( \Gamma' \models_S \sigma(M)[x\setminus N] : \sigma(B)[x\setminus N] \). By Lemma 5.19, \( \Gamma' \models_S (\lambda x : \sigma(B). \sigma(M)) : N : \sigma(B)[x\setminus N] \). Therefore \( \Gamma' \models_S (\lambda x : \sigma(B). \sigma(M)) : \Pi x : A. B \).

Without loss of generality, \( x \not\in \Gamma' \). Since \( s_3 : s \), we must have \( s_3 \neq \tau \), so \( (s_1, s_2, s_3) \in S \), which also means \( s_1 \neq \tau \) and \( s_2 \neq \tau \). By induction hypothesis, \( \sigma(A) \rightarrow A' \) such that \( \Gamma' \vdash_{\lambda S} A' : s_1 \). This means that \( WF_{\lambda S}(\Gamma', x : A') \) and \( \Gamma', x : A' \models_S \sigma(x/x) : (\Gamma, x : A) \). By induction hypothesis, \( \sigma(B) \rightarrow B' \) such that \( \Gamma' \vdash_{\lambda S} B' : s_2 \). Therefore \( \Gamma' \vdash_{\lambda S} (\Pi x : A'. B') : s_3 \), which implies \( \Gamma' \models_S (\Pi x : A'. B') : s_3 \).

By induction hypothesis, \( \Gamma' \models_S \sigma(M) : \sigma(A) \). Since \( A \equiv_{\beta} B \), we have \( \sigma(A) \equiv_{\beta} \sigma(B) \).

By Lemma 5.20, \( \Gamma' \models_S \sigma(M) : \sigma(A) \).