Programming Research Group

THE $\lambda\Pi$-CALCULUS WITH TYPE SIMILARITY

Jason Brown and Eike Ritter

PRG-TR-1-95

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

The $\lambda\Pi$-Calculus with Type Similarity

Jason Brown and Eike Ritter[*]

January 1995

Abstract

Motivated by the problems of the undecidablity of higher-order unification and hence the undecidability of $\lambda\Pi$-unification, Pym and Elliott give a weaker notion of typing for $\lambda\Pi$-objects : *type similarity*. In this paper we present a new calculus giving a formal theory of type similarity that captures this weaker notion of typing. We then apply a variant of Hardin's interpretation method to show that well-typed terms of this new calculus are confluent and normalizing for $\beta\eta$-reduction. This is done by defining two translations of the new calculus into the $\lambda\Pi$-calculus. We then use the results that well-typed $\lambda\Pi$-terms are confluent and strongly normalizing to get our results.

1 Introduction

The $\lambda\Pi$-calculus is an extension of the simply-typed λ-calculus with dependent function types, that is types can depend on the value of terms. As a consequence a unification procedure for the $\lambda\Pi$-calculus must unify not only terms, but also their types. To the authors' knowledge there are two presentations of such a procedure : [Pym92] which is an extension of Huet's [Hue75] procedure for higher-order unification and [Ell89, Ell90] based on a system of transformations in the flavour of Snyder & Gallier [SG89].

In the presentation of $\lambda\Pi$-unification both Pym and Elliott encounter difficulties in satisfying the condition of always unifying terms of the same type, an important condition[1] maintained by Huet's higher-order unification procedure. The difficulty arises owing to the semidecidability of higher-order unification[2], that is it is not possible in general to find a substitution, σ say, for the free variables of terms s, t such that $s\sigma = t\sigma$. At best one can find a *pre-unifier* [Hue75] which asserts the existence of a unifying substitution if one unifies terms of the same type. The combination of dependent types and the fact that unification in the $\lambda\Pi$-calculus requires types and terms to be unified, means that it is not always possible to ensure one is unifying terms of the same type. Consider the problem of unifying the pair of $\lambda\Pi$-terms

$$(D\,M\,y) \text{ and } (D\,M'\,N')$$

with respect to the signature

$$\Sigma = \langle A:\mathsf{Type}\ , C:\mathsf{Type}\ , B:A \to \mathsf{Type}\ , D:\Pi x:A.(B\,x) \to C\rangle$$

and context $\Gamma_1, y:(BM), \Gamma_2$ for terms M, M' and N' of the appropriate type. To unify $(D\,M\,y)$ and $(D\,M'\,N')$ one must unify M with M' and y with N'. The former two terms are of the same type, but the latter two are of different types namely $(B\,M)$ and $(B\,M')$ respectively. Unifying M and M' leads to y and N' having the same type, but this is a higher-order unification problem and hence undecidable in general. Therefore we will not be able to homogenize the types of y and N', but this does not mean that there is not a unifier for our original problem. In fact, the second pair, y with N' is easily unified by the substitution $[N'/y]$.

The types of y and N' are not equal (i.e., $\beta\eta$-equal) and hence any term the substitution is applied to, with y free, will not be typable using the typing rules of the $\lambda\Pi$. To overcome this Pym and Elliott

[*]jason.brown@comlab.ox.ac.uk, eike.ritter@comlab.ox.ac.uk

[1]It is required to show completeness of the unification procedure and used by the procedure as a preprocessor on filtering non-unifiable term. If a pair of terms to be unified are of different type, they are trivially non-unifiable.

[2]In fact even 2nd order unification is undecidable [Gol81].

introduce a *meta*-notation that weakens type equality and thus the $\lambda\Pi$-type system : *similarity* and *approximately well-typedness* respectively. Types are considered equal if they have the same structural shape. Substitutions of terms for variables where their types are structurally *similar* is allowed. Applying such a substitution results in terms whose types are structurally similar to their types before the substitution was applied. Elliott in [Ell89] states that approximately well-typed terms are neither Church-Rosser nor Strongly Normalizing. Also, Pym in [Pym92] states that objects and types under this weaker typing do not have these properties either. Indeed in the absence of a *formal* theory of type similarity, the Church-Rosser and Strong Normalization properties cannot be defined.

In this paper we define the $\lambda\Pi_\sim$-calculus, the $\lambda\Pi$-calculus with a weaker type equality suggested by Pym and Elliott. This new calculus gives a formal theory of type similarity that captures the notation used in [Pym92]. After defining this restriction to the $\lambda\Pi$-calculus we show that objects of this new calculus exhibit both the Church-Rosser and Strong Normalization properties.

In the next section we re-present the $\lambda\Pi$-calculus defined in [HHP93]. In section 3 we define the $\lambda\Pi_\sim$-calculus, the $\lambda\Pi$-calculus with type equality defined as Pym's type similarity. We choose Pym's definition of type similarity, as the motivation for this paper came from our work [Bro] on encoding unification in the Edinburgh Logical Framework (LF). The unification procedure we used there was one based on [Pym92]. Section 4 and 5 contain the main part of this paper presenting a Church-Rosser and Strong Normalization theorem for the $\lambda\Pi_\sim$-calculus.

2 Preliminaries

For completeness of our presentation we first present the $\lambda\Pi$-calculus, originally introduced in [HHP93]. The $\lambda\Pi$-calculus is a language with three levels : *objects*, *types and type families* and *kinds*. Objects are classified by type, types and type families by kinds. The kind Type classifies the types; the other kinds classify functions f which yield a type $fM_1 \cdots M_n$ when applied to objects $M_1, \ldots, M_n$ of certain types determined by the kind of f.

The $\lambda\Pi$-calculus is a formal system for deriving assertions of one of the following forms :

Σ **sig**	(Σ is a valid signature)
$\vdash_\Sigma \Gamma$	(Γ is a valid context in Σ)
$\Gamma \vdash_\Sigma K$	(K is a kind in Γ and Σ)
$\Gamma \vdash_\Sigma A : K$	(A has kind K in Γ and Σ)
$\Gamma \vdash_\Sigma M : A$	(M has type A in Γ and Σ)

where the syntax is specified by the following grammar

Σ	$::=$	$\langle\rangle \mid \Sigma, c\!:\!K \mid \Sigma, c\!:\!A$	signature
Γ	$::=$	$\langle\rangle \mid \Gamma, y\!:\!A$	context
K	$::=$	Type	types of terms
	$\mid$	$\Pi x\!:\!A.K$	type family
A	$::=$	c	constant
	$\mid$	$\Pi x\!:\!A.B$	dependent function type
	$\mid$	$\lambda x\!:\!A.B$	type family formation
	$\mid$	$A\,M$	type family instantiation
M	$::=$	c	constant
	$\mid$	x	variable
	$\mid$	$\lambda x\!:\!A.M$	abstraction
	$\mid$	$M\,N$	application

A *signature*, Σ, is a sequence of pairs associating constants with types or kinds, denoted by

$$\langle c_1\!:\!A_1, \ldots, c_n\!:\!A_n\rangle$$

The domain of Σ is the set of constants it defines, namely $\{c_1, \ldots, c_n\}$ and is denoted by $\mathbf{dom}(\Sigma)$. A *context*, Γ, is a sequence of pairs associating variables with types only, denoted

$$[y_1\!:\!A_1, \ldots, y_n\!:\!A_n]$$

The domain of Γ is the set of variables it defines, namely $\{y_1, \ldots, y_n\}$ and is denoted by $\mathbf{dom}(\Gamma)$.

The $\lambda\Pi$-calculus typing system, $\mathbf{N}$, is presented in Table 1. We write "$\mathbf{N}$ proves $\Gamma \vdash_\Sigma M : A$" to denote that the assertion $\Gamma \vdash_\Sigma M : A$ is derivable using the typing rules of $\mathbf{N}$. Further a term is said to be *well typed in a signature and context* if it can be shown to either be a kind, have a kind or have a type in that signature and context. A term is *well typed* if it is well typed in some signature and context.

The notation $U[M/x]$ is used to denote the result of substituting M for free occurrences of x in U, renaming bound variables to avoid capture. Terms are identified up to a change of variables, for example the terms $\lambda x\!:\!A.x$ and $\lambda y\!:\!A.y$ are identified.

2.1 Notions of Reduction

In this section the reduction relations β and η on $\lambda\Pi$-terms are defined using the notation of Barendregt [Bar84], Chapter 3. The *notions of reduction* β and η on $\lambda\Pi$-terms are defined by the following *contraction* rules :

(β - Contraction) Given terms $\lambda x\!:\!A.M$ and N and type $\lambda x\!:\!A.B$,

> **(Term)** $(\lambda x\!:\!A.M)N \longrightarrow_\beta M[N/x]$.
>
> **(Type)** $(\lambda x\!:\!A.B)N \longrightarrow_\beta B[N/x]$.

(η - Contraction) Given term M with variable $x \notin \mathsf{FV}(M)$ and type B with variable $y \notin \mathsf{FV}(B)$ then,

> **(Term)** $\lambda x\!:\!A.(M\,x) \longrightarrow_\eta M$.
>
> **(Type)** $\lambda x\!:\!A.(B\,x) \longrightarrow_\eta B$.

A notion of reduction ρ, where ρ is one of β, η or $\beta\eta$, induces a *one step ρ-reduction* by taking the *compatible closure* of ρ with respect to the rules in Table 2. Let $\rightarrow_\rho$ denote the one step ρ-reduction; this induces the following relations

(i). $\rightarrow^*_\rho$ (*ρ-reduction*) is the reflexive, transitive closure of $\rightarrow_\rho$;

(ii). $=_\rho$ (*ρ-equality*) is the equivalence relation generated by $\rightarrow^*_\rho$.

The definitional equality relation, as defined in [HHP93] and denoted by $\equiv$ in Table 1, coincides with $=_\beta$. This can be extended to $\beta\eta$, see [Sal91, Geu93], and it is this definition of equality that we assume for the remainder of this paper.

Remark : The canonical form of a type is

$$\Pi x_1\!:\!A_1.\ldots.\Pi x_n\!:\!A_n.a\,M_1\cdots M_p$$

for $n, p \geq 0$, where $A_1, 1 \leq i \leq n$ are canonical types, a is a constant of kind $\Pi y_1 : B_1.\ldots.\Pi y_p : B_p.\mathsf{Type}$ and $M_i, 1 \leq i \leq p$ are terms of the appropriate type. For the remainder of this paper we assume that all types are in canonical form. In fact it is possible to do better than this and remove the rule (B-ABS-FAM) without weakening the system. This result is due to Geuver [Geu93] who noted that once a signature and context are established one is only interested in judgements of the form $\Gamma \vdash_\Sigma M\!:\!A$ with A a type. One can assume Γ and Σ to be in canonical form and hence there is no need to be able to form types of the form $\lambda x : A.B$. We therefore assume the absence of the rule (B-ABS-FAM) from $\mathbf{N}$, and hence the absence of β and η contractions on types, for the remainder of this paper. This considerably simplifies our presentation. $\Box$

Table 1: The $\lambda\Pi$ System

Valid Signatures :	$$\frac{}{\langle\rangle\ \mathbf{sig}}$$	(B-EMPTY-SIG)
	$$\frac{\Sigma\ \mathbf{sig} \qquad \vdash_\Sigma K \qquad a \notin \mathrm{dom}(\Sigma)}{\Sigma, a\!:\!K\ \mathbf{sig}}$$	(B-KIND-SIG)
	$$\frac{\Sigma\ \mathbf{sig} \qquad \vdash_\Sigma A\!:\!\mathsf{Type} \qquad c \notin \mathrm{dom}(\Sigma)}{\Sigma, c\!:\!A\ \mathbf{sig}}$$	(B-TYPE-SIG)
Valid Contexts :	$$\frac{\Sigma\ \mathbf{sig}}{\vdash_\Sigma \langle\rangle}$$	(B-EMPTY-CTX)
	$$\frac{\vdash_\Sigma \Gamma \qquad \Gamma \vdash_\Sigma A\!:\!\mathsf{Type} \qquad x \notin \mathrm{dom}(\Gamma)}{\vdash_\Sigma \Gamma, x\!:\!A}$$	(B-TYPE-CTX)
Valid Kinds :	$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathsf{Type}}$$	(B-TYPE-KIND)
	$$\frac{\Gamma, x\!:\!A \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x\!:\!A.K}$$	(B-PI-KIND)
Valid Families :	$$\frac{\vdash_\Sigma \Gamma \qquad c\!:\!K \in \Sigma}{\Gamma \vdash_\Sigma c\!:\!K}$$	(B-CONST-FAM)
	$$\frac{\Gamma, x\!:\!A \vdash_\Sigma B\!:\!\mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x\!:\!A.B\!:\!\mathsf{Type}}$$	(B-PI-FAM)
	$$\frac{\Gamma, x\!:\!A \vdash_\Sigma B\!:\!K}{\Gamma \vdash_\Sigma \lambda x\!:\!A.B\!:\!\Pi x\!:\!A.K}$$	(B-ABS-FAM)
	$$\frac{\Gamma \vdash_\Sigma A\!:\!\Pi x\!:\!B.K \qquad \Gamma \vdash_\Sigma M\!:\!B}{\Gamma \vdash_\Sigma AM\!:\!K[M/x]}$$	(B-APP-FAM)
	$$\frac{\Gamma \vdash_\Sigma A\!:\!K \qquad \Gamma \vdash_\Sigma K' \qquad \Gamma \vdash_\Sigma K \equiv K'}{\Gamma \vdash_\Sigma A\!:\!K'}$$	(B-CONV-FAM)
Valid Objects :	$$\frac{\vdash_\Sigma \Gamma \qquad c\!:\!A \in \Sigma}{\Gamma \vdash_\Sigma c\!:\!A}$$	(B-CONST-OBJ)
	$$\frac{\vdash_\Sigma \Gamma \qquad x\!:\!A \in \Gamma}{\Gamma \vdash_\Sigma x\!:\!A}$$	(B-VAR-OBJ)
	$$\frac{\Gamma, x\!:\!A \vdash_\Sigma M\!:\!B}{\Gamma \vdash_\Sigma \lambda x\!:\!A.M\!:\!\Pi x\!:\!A.B}$$	(B-ABS-OBJ)
	$$\frac{\Gamma \vdash_\Sigma M\!:\!\Pi x\!:\!A.B \qquad \Gamma \vdash_\Sigma N\!:\!A}{\Gamma \vdash_\Sigma MN\!:\!B[N/x]}$$	(B-APP-OBJ)
	$$\frac{\Gamma \vdash_\Sigma M\!:\!A \qquad \Gamma \vdash_\Sigma A'\!:\!\mathsf{Type} \qquad \Gamma \vdash_\Sigma A \equiv A'}{\Gamma \vdash_\Sigma M\!:\!A'}$$	(B-CONV-OBJ)

Table 2: *Compatible* Closure

for kinds,

$$\frac{A \to_\rho A'}{\Pi x\!:\!A.K \to_\rho \Pi x\!:\!A'.K} \qquad \frac{K \to_\rho K'}{\Pi x\!:\!A.K \to_\rho \Pi x\!:\!A.K'}$$

then for types,

$$\frac{}{\lambda x\!:\!A.(B\,x) \to_\eta B} \qquad \frac{}{(\lambda x\!:\!A.B)N \to_\beta B[N/x]}$$

$$\frac{A \to_\rho A'}{A\,M \to_\rho A'\,M} \qquad \frac{M \to_\rho M'}{A\,M \to_\rho A\,M'}$$

$$\frac{A \to_\rho A'}{\Pi x\!:\!A.B \to_\rho \Pi x\!:\!A'.B} \qquad \frac{B \to_\rho B'}{\Pi x\!:\!A.B \to_\rho \Pi x\!:\!A.B'}$$

$$\frac{A \to_\rho A'}{\lambda x\!:\!A.B \to_\rho \lambda x\!:\!A'.B} \qquad \frac{B \to_\rho B'}{\lambda x\!:\!A.B \to_\rho \lambda x\!:\!A.B'}$$

and for terms,

$$\frac{}{\lambda x\!:\!A.(M\,x) \to_\eta M} \qquad \frac{}{(\lambda x\!:\!A.M)N \to_\beta M[N/x]}$$

$$\frac{A \to_\rho A'}{\lambda x\!:\!A.M \to_\rho \lambda x\!:\!A'.M} \qquad \frac{M \to_\rho M'}{\lambda x\!:\!A.M \to_\rho \lambda x\!:\!A.M'}$$

$$\frac{M \to_\rho M'}{M\,N \to_\rho M'\,N} \qquad \frac{N \to_\rho N'}{M\,N \to_\rho M\,N'}$$

3 $\lambda\Pi_\sim$-Calculus

In this section we introduce the notion of *similar* type based on Pym's definition in [Pym92] and we formalise the $\lambda\Pi_\sim$-calculus, the $\lambda\Pi$-calculus with type similarity.

Motivation for the definition of similar types is given by our example in the introduction. The problem of unifying y with N' yielded a substitution $[N'/y]$, but it was noted that their types were different : y has type $(B\,M)$ and N' has type $(B\,M')$. These types are identical if M and M' are unifiable, but a *pre-unifier* of these two terms only implies the existence of a unifier, it is not itself a full unifier. That is, suppose σ is a *pre-unifier* then $M\sigma \neq M'\sigma$ and therefore $(B\,M)\sigma \neq (B\,M')\sigma$, but we know there exists a unifier, θ say, such that $M\theta = M'\theta$. We seek a definition of type equality that is invariant under substitution, that is we want $(B\,M)\sigma = (B\,M')\sigma$.

Recall that the canonical form of a type is :

$$\Pi x_1\!:\!A_1.\ldots.\Pi x_n\!:\!A_n.a\,M_1\cdots M_p$$

Variables, free or bound, can only occur in the objects of a type, that is in $M_i, 1 \leq i \leq p$ or objects in $A_j, 1 \leq j \leq n$. In particular variables cannot occur at the head of a type, that is a cannot be a variable. This allows us to define type similarity as follows.

DEFINITION 3.1 [Similarity]

(i). Let A and B be types of kind Type , then A and B are similar, written $A \sim B$, if

$$A = \Pi x_1\!:\!A_1\ldots\Pi x_n\!:\!A_n.a\,M_1\ldots M_q$$

$$B = \Pi y_1\!:\!B_1\ldots\Pi y_n\!:\!B_n.a\,N_1\ldots N_q$$

and $A_i \sim B_i$ for $1 \leq i \leq n$.

(ii). Similarity on raw terms is defined as the smallest congruence relation including $A \sim B$ implies $\lambda x\colon A.M \sim \lambda x\colon B.N$.

□

With respect to this relation we have that $(B\,M)\sigma$ and $(B\,M')\sigma$ are equivalent, which is what we required. We define the notion of reduction for the $\lambda\Pi_{\sim}$-calculus to be the notion of reduction for $\lambda\Pi$. The judgements for well-formed expressions are the one for the $\lambda\Pi$-calculus with definitional equality replaced by similarity. The task now is to show both Church-Rosser and strong normalization for this system. Eventually, we want to derive these properties from their counterparts in $\lambda\Pi$. The key idea of this proof is a suitable translation of the $\lambda\Pi_{\sim}$-calculus into the $\lambda\Pi$-calculus. This translation is only well-defined if the $\lambda\Pi_{\sim}$-calculus has the subject reduction property, i.e. that $\Gamma \vdash M\colon A$ and $M \longrightarrow^{*}_{\beta\eta} M'$ implies $\Gamma \vdash M'\colon A$.

The proof of this property in the $\lambda\Pi$-calculus is very complicated because unicity of Π-types is very difficult to establish. The situation is different for the $\lambda\Pi_{\sim}$-calculus. Because the simulation essentially counts only the number of terms to which a type constant is applied to and not their structure, we can show that any type is similar to a well-defined canonical form. This in turn implies directly the unicity of Π-types.

PROPOSITION 3.2 (UNICITY OF Π-TYPES)

If $\Pi x\colon A_1.B_1 \sim \Pi x\colon A_2.B_2$ *holds, then we have also* $A_1 \sim A_2$ *and* $B_1 \sim B_2$. □

PROOF. By induction on the number of Π abstractions in B_1 and B_2.

Subject reduction is an immediate consequence of this proposition.

4 Church-Rosser Theorem

In this section we show that well-typed $\lambda\Pi_{\sim}$-terms are Church-Rosser. This is achieved by embedding the $\lambda\Pi_{\sim}$-calculus into the $\lambda\Pi$-calculus, we then use the fact that well-typed $\lambda\Pi$-terms are Church-Rosser [Sal91, Coq91] to give the required result for $\lambda\Pi_{\sim}$-terms. The embedding is a translation from $\lambda\Pi_{\sim}$-terms to $\lambda\Pi$-terms, such that all terms that are well-typed in the $\lambda\Pi_{\sim}$-calculus are translated into well-typed $\lambda\Pi$-terms. This enables us to observe that translated well-typed $\lambda\Pi_{\sim}$-terms exhibit the Church-Rosser property. Then by showing that all $\lambda\Pi_{\sim}$-terms and their translations are similar and that similarity commutes with $\beta\eta$-reduction we lift the Church-Rosser property to well-typed $\lambda\Pi_{\sim}$-terms. This method is a variant of the interpretation method, which Hardin [Har89] used to show confluence for categorical combinators.

In order to distinguish $\lambda\Pi_{\sim}$and $\lambda\Pi$-type deductions we superscript the turnstile of the $\lambda\Pi_{\sim}$-type rules with $\sim$. Further we represent the translation of $\lambda\Pi_{\sim}$-terms by putting a bar over them, that is $\overline{M}$ denotes the translation of the $\lambda\Pi_{\sim}$-term M. The translations of types, kinds, contexts and signatures are denoted likewise. Motivation for the definition of the translation is given by considering the well-typedness condition it must respect. To show that our translation respects typing we must prove the following

$$\Gamma \vdash^{\sim}_{\Sigma} M\colon A \text{ iff } \overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{M}\colon\overline{A}$$

where Σ, Γ, A and M are $\lambda\Pi_{\sim}$-signature, -context, -type and term respectively. Notice that because the $\lambda\Pi_{\sim}$-type A is translated into the $\lambda\Pi$-type $\overline{A}$, all similar types must translate to one and the same $\lambda\Pi$-type. This is achieved by translating the variant parts of similar types, that is the terms within similar types, to a constant of the $\lambda\Pi$. Regardless of the form of the terms in the types one always gets the same constant under translation of these terms. We will see that this requires us to make the translation of terms, types and contexts dependent on the signature.

In defining the translation for $\lambda\Pi_{\sim}$-signatures, terms and types, one should pay close attention to the translation of types. Each type is translated with respect to a translated signature. In translating signatures each type constant a with kind $\Pi x_1 : A_n. \cdots \Pi x_n : A_n.\mathsf{Type}$ results in the addition of n new constants to the signature. Now recall that the body of a $\lambda\Pi_{\sim}$-type is of the form $a\,M_1 \cdots M_n$ for terms $M_1, \cdots, M_n$, this is translated to a $\lambda\Pi$-type body of the form $a\,c_1 \cdots c_n$ where $c_1, \ldots, c_n$ are the new constants that were added to the signature when a was translated. This selection of constants enables the translation of $\lambda\Pi_{\sim}$-types to $\lambda\Pi$-types to satisfy the criteria discussed above. It also means that type and term translations are dependent on a translated signature.

DEFINITION 4.1 Let Σ be a $\lambda\Pi_{\sim}$-signature, and let A and M be $\lambda\Pi_{\sim}$-types and terms defined with respect to the signature Σ. We define $\lambda\Pi$-signatures, types and terms $\overline{\Sigma}$, $\overline{A}^{\Sigma}$ and $\overline{M}^{\Sigma}$ as follows:

Signatures :

$$
\begin{aligned}
\overline{\langle\,\rangle} &= \langle\,\rangle \\
\overline{\Sigma, c : A} &= \overline{\Sigma}, c : \overline{A}^{\Sigma} \\
\overline{\Sigma, a : \mathsf{Type}} &= \overline{\Sigma}, a : \mathsf{Type} \\
\overline{\Sigma, a : K} &= \overline{\Sigma}, \\
&\quad\ a : \Pi x_1 : A'_1 \ldots \Pi x_n : A'_n(x_1 \cdots x_{n-1}).\mathsf{Type}\ , \\
&\quad\ c_1 : A'_1, \ldots, c_n : A'_n(c_1 \cdots c_{n-1})
\end{aligned}
$$

where

(i). $K = \Pi x_1 : A_1. \cdots .\Pi x_n : A_n(x_1 \cdots x_n).\mathsf{Type}$;

(ii). $A'_i = \overline{A_i}^{\Sigma}$ for $1 \leq i \leq n$;

(iii). $c_j \notin \mathrm{dom}(\overline{\Sigma})$ for $1 \leq j \leq n$.

Types :

$$
\begin{aligned}
\overline{\Pi x : A.B}^{\Sigma} &= \Pi x : \overline{A}^{\Sigma}.\overline{B}^{\Sigma} \\
\overline{a\,M_1 \cdots M_n}^{\Sigma} &= a\,c_1 \cdots c_n
\end{aligned}
$$

where $\overline{\Sigma} = \overline{\Sigma}_1, a : K, c_1 : A_1, \ldots, c_n : A_n(c_1, \cdots, c_{n-1}), \overline{\Sigma}_2$ for signatures $\overline{\Sigma}_1$,$\overline{\Sigma}_2$ and kind $K = \Pi x_1 : A_1. \cdots .\Pi x_n : A_n.\mathsf{Type}$.

Objects :

$$
\begin{aligned}
\overline{x}^{\Sigma} &= x \\
\overline{c}^{\Sigma} &= c \\
\overline{\lambda x : A.M}^{\Sigma} &= \lambda x : \overline{A}^{\Sigma}.\overline{M}^{\Sigma} \\
\overline{M\,N}^{\Sigma} &= \overline{M}^{\Sigma}\,\overline{N}^{\Sigma}
\end{aligned}
$$

where $c : A \in \overline{\Sigma}$ for some A.

□

The translation can be extended to kinds in the obvious way. Translating a context with respect to a given signature consists of translating each element of that context. One can easily convince oneself that the translation on signatures is well-founded by considering the lexicographical ordering on pairs of natural numbers. Let the measure on our translation be the pair where the first element is the size of the signature and the second element is the size of terms. Each step of the translation strictly decreases this measure.

For the remainder of this paper we assume a fixed signature Σ. To simplify notation we shall leave out superscripted signatures on translations, preferring to write $\overline{M}$ for $\overline{M}^{\Sigma}$. We will need the following substitution lemma in the proofs below.

LEMMA 4.2 *Let A be a $\lambda\Pi_{\sim}$-type and M, N be $\lambda\Pi_{\sim}$-objects* *(i).* $\overline{A[N/x]} = \overline{A}[\overline{N}/x]$

(ii). $\overline{M[N/x]} = \overline{M}[\overline{N}/x]$

PROOF. (i). By induction on the structure of A. The interesting case is when $A = a\, M_1 \cdots M_n$.

$$\begin{array}{rcl} \overline{(a\, M_1 \cdots M_n)[N/x]} & = & \overline{a(M_1[N/x]) \cdots (M_n[N/x])} \\ & = & a\, c_1 \cdots c_n \\ & = & \overline{(a\, c_1 \cdots c_n)}[\overline{N}/x] \\ & = & \overline{(a\, M_1 \cdots M_n)}[\overline{N}/x] \end{array}$$

(ii). By a simple induction on the structure of M.

As outlined above, the property of the translation that we need to show is that it respects typing. This property is stated by the following lemma.

LEMMA 4.3 *Let* Γ, M, A *be* $\lambda\Pi_{\sim}$*-context, -signature, -term and -type respectively, then*

$$\Gamma \vdash^{\sim}_{\Sigma} M : A \textit{ iff } \overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{M} : \overline{A}$$

PROOF. The proof requires mutually dependent inductions on all forms of judgements. Most cases of both inductions presented below are tedious and not very enlightening, we therefore restrict ourselves to showing a few of the more interesting sub-cases.

only if : By induction on the generation of $\Gamma \vdash^{\sim}_{\Sigma} M : A$.

1. Assume that
$$\frac{\Sigma\ \mathbf{sig} \qquad \vdash^{\sim}_{\Sigma} K \qquad a \notin \mathrm{dom}(\Sigma)}{\Sigma, a : K\ \mathbf{sig}}\ (\textsf{B-KIND-SIG})$$
then proceed by case analysis on the form of the kind K.
   (a) Suppose that $K = \mathsf{Type}$. By the induction hypothesis we have $\overline{\Sigma}$ **sig** and $\vdash_{\overline{\Sigma}} \overline{\mathsf{Type}}$ and $a \notin \mathrm{dom}(\Sigma)$. Then by applying the rule (B-KIND-SIG) we get :
$$\frac{\overline{\Sigma}\ \mathbf{sig} \qquad \vdash_{\overline{\Sigma}} \overline{\mathsf{Type}} \qquad a \notin \mathrm{dom}(\Sigma)}{\overline{\Sigma}, a : \overline{\mathsf{Type}}\ \mathbf{sig}}$$
   (b) Suppose that $K = \Pi x_1 : A_1 . \ldots . \Pi x_n : A_n . \mathsf{Type}$. By the induction hypothesis we have $\overline{\Sigma}$ **sig**, $\vdash_{\overline{\Sigma}} \overline{K}$ and $a \notin \mathrm{dom}(\overline{\Sigma})$. By definition of the translation, n new constants are added to the signature. By applying the rule (B-KIND-SIG) we get :
$$\frac{\overline{\Sigma}\ \mathbf{sig} \qquad \vdash_{\overline{\Sigma}} \overline{K} \qquad a \notin \mathrm{dom}(\Sigma)}{\overline{\Sigma}, a : \overline{K}, c_1 : A'_1, \cdots c_n : A'_n\ \mathbf{sig}}$$
   where $A'_i = \overline{A_i}(c_1, \cdots, c_{i-1})$ for all $1 \leq i \leq n$. It is possible to convince oneself that the resulting signature is valid since we know that in the derivation of $\vdash_{\overline{\Sigma}} \overline{K}$ that
$$x_1 : \overline{A_1}, \cdots, x_{i-1} : \overline{A_{i-1}}(x_1, \cdots, x_{i-2}) \vdash_{\overline{\Sigma}} \overline{A_i}(x_1, \cdots, x_{i-1}) : \mathsf{Type}$$
   for all $1 < i \leq n$ and that $\vdash_{\overline{\Sigma}} A_1 : \mathsf{Type}$. Since the translation adds a new constant of the appropriate type for each abstracted variable we have that
$$\vdash_{\overline{\Sigma}_i} \overline{A_i}(c_1, \cdots, c_{i-1}) : \mathsf{Type}$$
   where $\overline{\Sigma}_i = \overline{\Sigma}, a : \overline{K}, c_1 : \overline{A_1}, \cdots, c_{i-1} : \overline{A_{i-1}}(c_1, \cdots, c_{i-2})$ for $1 < i \leq n$ and $\overline{\Sigma}_1 = \overline{\Sigma}, a : \overline{K}$.

2. Assume that
$$\frac{\Gamma \vdash^{\sim}_{\Sigma} A : \Pi x : B.K \quad \Gamma \vdash^{\sim}_{\Sigma} M : B}{\Gamma \vdash^{\sim}_{\Sigma} A\, M : K[M/x]} \text{ (B-APP-FAM)}$$
By the induction hypothesis we have $\overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{A} : \overline{\Pi x : B.K}$. By definition of the translation we have that $\overline{A\, M}$ is mapped to $\overline{A}\, c$ for constant $c \in \mathrm{dom}(\overline{\Sigma})$. By the previous subcase we know that c is uniquely determined by the head of A and that $\overline{\Gamma} \vdash_{\overline{\Sigma}} c : \overline{B}$. Thus we can apply the rule (B-APP-FAM) and lemma 4.2 to get the result we require :
$$\frac{\overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{A} : \overline{\Pi x : B.K} \quad \overline{\Gamma} \vdash_{\overline{\Sigma}} c : \overline{B}}{\overline{\Gamma} \vdash_{\overline{\Sigma}} (\overline{A}\, c) : \overline{K}[c/x]}$$

3. Assume that
$$\frac{\Gamma \vdash^{\sim}_{\Sigma} M : A \quad \Gamma \vdash^{\sim}_{\Sigma} A' : \mathsf{Type} \quad \Gamma \vdash^{\sim}_{\Sigma} A \sim A'}{\Gamma \vdash^{\sim}_{\Sigma} M : A'} \text{ (B-CONV-OBJ)}$$
By the induction hypothesis we have $\overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{M} : \overline{A}$ and $\overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{A} : \overline{\mathsf{Type}}$. The crucial point is that if $A \sim A'$ then $\overline{A} \equiv \overline{A'}$, that is the translated types are definitionally equal. This ensures that the definition of the translation is not dependent on the form of the derivation. We can now apply the rule (B-CONV-OBJ) to get
$$\frac{\overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{M} : \overline{A} \quad \overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{A'} : \overline{\mathsf{Type}} \quad \overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{A} \equiv \overline{A'}}{\overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{M} : \overline{A'}}$$

if : By induction on the generation of $\overline{\Gamma} \vdash_{\overline{\Sigma}} \overline{M} : \overline{A}$. This direction is easier, just forget the new constants that were added to the Σ to obtain $\overline{\Sigma}$.

□

The following three lemmas are required to show that similarity commutes with $\beta\eta$-reduction. The following lemma states that any $\lambda\Pi_{\sim}$-sequence of reductions of a $\lambda\Pi_{\sim}$-term is translated into a $\lambda\Pi$-sequence of reductions on the translated term, and vice versa.

Lemma 4.4 *Let U, V be $\lambda\Pi_{\sim}$-terms, then*
$$U \longrightarrow^{*}_{\beta\eta} V \textit{ implies } \overline{U} \longrightarrow^{*}_{\beta\eta} \overline{V}$$

Proof. First suppose that $U \longrightarrow^{*}_{\beta\eta} V$ is a one step reduction. The proof proceeds by induction on the structure of U. We show the interesting cases, the other cases are easier.

1. Let $U = (\lambda x : A.M)N$ and $V = M[N/x]$ by β-reduction
$$\begin{array}{rcll} \overline{(\lambda x : A.M)N} & = & (\lambda x : \overline{A}.\overline{M})\overline{N} & \\ & \longrightarrow_{\beta} & \overline{M}[\overline{N}/x] & \\ & = & \overline{M[N/x]} & \text{by lemma 4.2} \end{array}$$

2. let $U = \lambda x : A.(M x)$ with $x \notin \mathsf{FV}(M)$ and $V = M$ by η-reduction
$$\begin{array}{rcl} \overline{\lambda x : A.(M x)} & = & \lambda x : \overline{A}.(\overline{M}\overline{x}) \\ & = & \lambda x : \overline{A}.(\overline{M} x) \\ & \longrightarrow_{\eta} & \overline{M} \end{array}$$

3. let $U = a\, M_1 \cdots M_n$ for some n, $M_i \longrightarrow_{\beta\eta} M_i'$ for some $1 \le i \le n$ and $V = a\, M_1 \cdots M_i' \ldots M_n$. By definition of the translation we have $\overline{M_i} = c_i = \overline{M_i'}$ where c_i is uniquely determined by a. Then
$$\begin{array}{rcl} \overline{a\, M_1 \cdots M_i \cdots M_n} & = & a\, c_1 \cdots c_i \cdots c_n \\ & = & \overline{a\, M_1 \cdots M_i' \cdots M_n} \end{array}$$

The general statement follows by transitivity. $\square$

The following lemma states that if a translated term contracts then its contractum remains within the image of the translation. This lemma is the crucial component in enabling us to lift Church-Rosser to well-typed $\lambda\Pi_{\sim}$-terms.

LEMMA 4.5 *Let U be a $\lambda\Pi_{\sim}$-term and V a $\lambda\Pi$-term then,*

$$\overline{U} \longrightarrow^*_{\beta\eta} V \textit{ implies } \exists W.\overline{W} = V$$

PROOF. First suppose that $\overline{U} \longrightarrow^*_{\beta\eta} V$ is a one step reduction. Proof proceeds by induction on the structure of U. We show only the interesting sub-cases as the others are similar.

Base : $U = a\, M_1 \cdots M_n$. By definition of the translation $\overline{U} = a\, c_1 \cdots c_n$. Since there are no possible contractions of $\overline{U}$ the statement is trivially true.

Step :
1. Suppose $U = (\lambda x : A.M)N$ and so $\overline{U} = (\lambda x : \overline{A}.\overline{M})\overline{N}$. Let $V = \overline{M}[\overline{N}/x]$ be obtained from $\overline{U}$ by β-reduction. By lemma 4.2 we have $V = \overline{M[N/x]}$ and hence let $W = M[N/x]$.
2. Suppose $U = \lambda x : A.(M\, x)$ and so $\overline{U} = \lambda x : \overline{A}.(\overline{M}\, x)$. Let $V = \overline{M}$ be obtained from $\overline{U}$ by η-reduction and hence let $W = M$.

The general statement follows by transitivity. $\square$

From the definition of the translation it is clear that any term and its translation are similar. This observation relies on the fact that similar types have the same head constant which is unaffected by the translation. The next lemma generalises this observation by stating that the translation of two similar terms are themselves similar.

LEMMA 4.6 *Let U, V be $\lambda\Pi_{\sim}$-terms, then*

$$U \sim V \textit{ iff } \overline{U} = \overline{V}$$

PROOF. By a simple induction on the generation of $U \sim V$. $\square$

Finally we combine the above three lemmas to show that similarity of terms commutes with $\beta\eta$-reduction.

LEMMA 4.7 *Let U, U', V be $\lambda\Pi_{\sim}$-terms. If $U \sim V$ and $U \longrightarrow^*_{\beta\eta} U'$ then there exists V' such that $U' \sim V'$ and $V \longrightarrow^*_{\beta\eta} V'$.*

PROOF. By previous lemma $\overline{U} = \overline{V}$. Then from the fact that $U \longrightarrow^*_{\beta\eta} U'$ and lemma 4.4 we know that $\overline{U} \longrightarrow^*_{\beta\eta} \overline{U'}$. Hence there exists a T such that $\overline{V} \longrightarrow^*_{\beta\eta} T$ and $\overline{U'} = T$. By lemma 4.5 there exists a V' say, such that $\overline{V'} = T$ and so we have that $\overline{V} \longrightarrow^*_{\beta\eta} \overline{V'}$. Then by contracting the redexes of V that correspond to those contracted in the reduction $\overline{V} \longrightarrow^*_{\beta\eta} \overline{V'}$ we have that $V \longrightarrow^*_{\beta\eta} V'$. Finally we have $U' \sim \overline{U'} = T = \overline{V'} \sim V'$, that is $U' \sim V'$. $\square$

We now state and prove the Church-Rosser theorem.

THEOREM 4.8 (CHURCH-ROSSER) *Let Σ, Γ, M, A be a $\lambda\Pi_{\sim}$-signature, -context, -term and -type such that $\Gamma \vdash^{\sim}_{\Sigma} M : A$. If $M \longrightarrow^*_{\beta\eta} M_1$ and $M \longrightarrow^*_{\beta\eta} M_2$ for some M_1, M_2 then there exists M_1' and M_2' such that $M_1 \longrightarrow^*_{\beta\eta} M_1'$ and $M_2 \longrightarrow^*_{\beta\eta} M_2'$ and $M_1' \sim M_2'$.* . $\square$

PROOF. Using the previous two lemmas and Church-Rosser for the $\lambda\Pi$-calculus we have.

$$\begin{tikzcd}[column sep=small]
 & M \arrow[dl, "\beta\eta"'] \arrow[dr, "\beta\eta"] & \\
M_1 \arrow[d, dotted, "\beta\eta"'] & & M_2 \arrow[d, dotted, "\beta\eta"'] \\
M_1' & & M_2'
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}[column sep=small]
 & \overline{M} \arrow[dl, dotted, "\beta\eta"'] \arrow[dr, dotted, "\beta\eta"] & \\
\overline{M_1} \arrow[dr, "\beta\eta"'] & & \overline{M_2} \arrow[dl, "\beta\eta"] \\
 & \overline{M_3} &
\end{tikzcd}$$

$\lambda\Pi_{\sim}$-Calculus $\qquad\qquad$ $\lambda\Pi$-Calculus

$\overline{M}, \overline{M_1}$ and $\overline{M_2}$ are the translations of M, M_1 and M_2 respectively. By lemma 4.6 we know that $M \sim \overline{M}$. Then by two applications of lemma 4.7 we have that $\overline{M} \longrightarrow^*_{\beta\eta} N_1$ and $\overline{M} \longrightarrow^*_{\beta\eta} N_2$ with $M_1 \sim N_1 (\sim \overline{M_1})$ and $M_2 \sim N_2 (\sim \overline{M_2})$.

By Church-Rosser of well-typed $\lambda\Pi$-terms there exists an $\lambda\Pi_{\sim}$-term, N_3 say, such that $N_1 \longrightarrow^*_{\beta\eta} N_3$ and $N_2 \longrightarrow^*_{\beta\eta} N_3$. Then by lemma 4.7 and $M_1 \sim N_1$ we have that there exists a $\lambda\Pi$-term, M_1' say, such that $M_1 \longrightarrow^*_{\beta\eta} M_1'$ with $M_1' \sim N_3$. By another application of lemma 4.7 there exists a $\lambda\Pi$-terms, M_2' say, such that $M_1 \longrightarrow^*_{\beta\eta} M_2'$ such that $M_1' \sim N_3$.

Finally by transitivity of $\sim$ we have $M_1' \sim M_2'$. □

Remark : This is the not the usual Church-Rosser theorem one is used to seeing. In the standard Church-Rosser theorem M_1' and M_2' would be identical up to a change of variables, that is α-equivalent. For us to recover the standard theorem we would need to define identical terms as α-equivalent up to type similarity. For example we would need to identify $\lambda x : A.x$ with $\lambda y : B.y$ if $A \sim B$. But, as will be noted in the next section this would mean the unavoidable loss of Strong Normalization for well-typed $\lambda\Pi_{\sim}$-terms. □

5 Strong Normalization Theorem

In this section we show, by translating $\lambda\Pi_{\sim}$-terms into $\lambda\Pi$-terms, that well-typed $\lambda\Pi_{\sim}$-terms are Strongly Normalizing. Such a translation must preserve the structure of $\lambda\Pi_{\sim}$-terms and the number of $\beta\eta$-reductions. In addition the translation must preserve the well-typedness of terms. With these properties and the fact that well-typed $\lambda\Pi$-terms are Strongly Normalizing[3] we get the result we want for well-typed $\lambda\Pi_{\sim}$-terms.

It is not possible to use the translation we defined in the previous section since it is not structure preserving and hence does not preserve the number of $\beta\eta$-reductions. This is owing to the fact that when types are translated the objects of applications are discarded. We can recover structure preservation and hence the number of $\beta\eta$-reductions by preserving these objects under the translation. We redefine the translation of types as follows.

Types :

$$\begin{aligned}
\overline{\Pi x : A.B}^{\Sigma} &= \Pi x : \overline{A}^{\Sigma}.\overline{B}^{\Sigma} \\
\overline{a\, M_1 \cdots M_n}^{\Sigma} &= (\lambda x_1 : A_1 \ldots \lambda x_n : A_n . a\, c_1 \cdots c_n)\, \overline{M_1}^{\Sigma} \cdots \overline{M_n}^{\Sigma}
\end{aligned}$$

where $\overline{\Sigma} = \overline{\Sigma}_1, a : K, c_1 : A_1, \ldots, c_n : A_n(c_1, \cdots, c_{n-1}), \overline{\Sigma}_2$ for signatures $\overline{\Sigma}_1$,$\overline{\Sigma}_2$ and kind $K = \Pi x_1 : A_1 . \cdots . \Pi x_n : A_n . \mathsf{Type}$.

[3]This is a consequence of Geuvers' proof [Geu93] of Strong Normalization for the Calculus of Constructions.

Remark : Identifying α-equivalent $\lambda\Pi_{\sim}$-types up to type similarity would mean that this translation would not preserve $\beta\eta$-reductions. This is due to the interaction between $\beta\eta$-reduction and term identity. Typically $\beta\eta$-reduction of a term implicitly allows the replacing of one term for an identical term at any point in the reduction. This works if α-equivalent terms are considered identical since α-reduction does not introduce new $\beta\eta$-redexes. Unfortunately including type similarity in the definition of identity on terms can introduce new $\beta\eta$-redexes. For example, as in the remark above let us define terms identical if they are α-equivalent up to type similarity, written $\equiv_{\sim}$.We then have that $\lambda x : A.x \equiv_{\sim} \lambda y : B.y$ if $A \sim B$. We see that this can lead to infinite reductions, for example

$$\lambda x : (a\,N).x \longrightarrow^{*}_{\beta\eta} \lambda x : (a\,N').x \equiv_{\sim} \lambda x : (a\,N).x \longrightarrow^{*}_{\beta\eta} \cdots$$

In the definition of the $\lambda\Pi_{\sim}$-calculus given above we avoid this problem by using the usual definition of term identity, that is α-equivalence. Type similarity is a seperate equivalence relation on types and hence does not effect $\beta\eta$-reductions. $\Box$

To obtain Strong Normalization for well-typed $\lambda\Pi_{\sim}$-terms we must show that the translation preserves well-typedness and $\beta\eta$-reductions. Showing that the translation preserves well-typedness is similar to showing it for the translation in the last section. Since the only difference in the translations is the way types are translated we need only convince ourselves that the new way to translate types respects well-typing.

Consider the $\lambda\Pi_{\sim}$-type $a\,M_1 \cdots M_n$ for some terms $M_1, \ldots, M_n$. If $a\,M_1 \cdots M_n$ is well-typed in the $\lambda\Pi_{\sim}$-calculus then the type of a's actual parameters are similar to the types of the formal parameters. Then by lemma 4.6, under the translation the types of the translated actual and formal parameters will be equivalent. Hence the term

$$(\lambda x_1 : A_1 \ldots \lambda x_n : A_n.a\,c_1 \cdots c_n)\,M_1 \cdots M_n$$

which is the result of translating the type $a\,M_1 \cdots M_n$ will be well-typed in the $\lambda\Pi$-calculus.

Since the translation respects $\beta\eta$-reductions and well-typedness and that well-typed $\lambda\Pi$-terms are Strongly Normalizing we get that well-typed $\lambda\Pi_{\sim}$-terms are also Strongly Normalizing.

6 Conclusion

In this paper we presented the $\lambda\Pi_{\sim}$-calculus, providing a formal theory of type similarity that captures the notation used in [Pym92]. This calculus enabled us to formulate and prove the Church-Rosser and Strong Normalization properties for terms involving this weaker form of typing, something that could not be done in either [Pym92] or [Ell89].

Acknowledgements

We would like to thank David Pym for many helpful discussions clarifying points on his work that we generally misunderstood and for providing many comments and corrections to drafts of this paper. We are also grateful to Healfdene Goguen and Ralph Loader for their comments and corrections of earlier drafts. Jason Brown would like to thank his supervisor Lincoln Wallen for suggesting the topic of study from which this work emerged.

Jason Brown is funded by an EPSRC studentship and Eike Ritter is funded by an EPSRC grant, number GR/J46616.

References

[Bar84] H. P. Barendregt. *The Lambda Calculus, its syntax and semantics.* North Holland, 1984.

[Bro] Jason Brown. Encodings of Unification in a Logical Framework. DPhil thesis, University of Oxford, forthcoming, 199–.

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In Huet and Plotkin, editors, *Logical Frameworks*. Cambridge University Press, 1991.

[Ell89] Conal Elliott. Higher-order unification with dependent type functions. In N. Dershowitz, editor, *Proceedings of the 3^{rd} International Conference on Rewriting Techniques and Applications*, volume 355 of *LNCS*, pages 121–136. Springer-Verlag, 1989.

[Ell90] Conal Elliott. *Extensions and Applications of Higher-Order Unification*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1990.

[Geu93] Herman Geuvers. *Logics and Type Systems*. PhD thesis, Katholieke Universiteit Nijmegen, 1993.

[Gol81] W. Goldfarb. The Undecidability of the Second Order Unification Problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[Har89] Thérèse Hardin. Confluence results for the pure strong categorical logic CCL λ-calculi as subsystems of CCL. *Theoretical Computer Science*, 65:291–342, 1989.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

[Hue75] Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[Pym92] David Pym. A unification algorithm for the $\lambda\Pi$-Calculus. *International Journal of Foundations of Computer Science*, 3(3):333–378, 1992.

[Sal91] A. Salvesen. The Church-Rosser property for $\beta\eta$-reduction. Manuscript, 1991.

[SG89] Wayne Snyder and Jean Gallier. Higher-order unification revisited : Complete sets of transformations. *Journal of Symbolic Computation*, 8:101–140, 1989.