LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - The University of Edinburgh

A Unification Algorithm for the $\lambda\Pi$-Calculus

by

David J Pym

LFCS Report Series **ECS-LFCS-92-229**
LFCS **August 1992**
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Copyright © 1992, LFCS

Copyright © 1992, Laboratory for Foundations of Computer Science
University of Edinburgh. All rights reserved.

Reproduction of all or part of this work is
permitted for educational or research use on
condition that this copyright notice is included
in any copy.

Preprint of an article to appear in the International Journal of Foundations of Computer Science: please refer only to this journal article. Please note that the typesetting of this preprint may differ from that of the aforesaid journal article.

A UNIFICATION ALGORITHM FOR THE $\lambda\Pi$-CALCULUS

DAVID J. PYM
University of Edinburgh*
Laboratory for Foundations of Computer Science
Department of Computer Science
The King's Buildings
Edinburgh EH9 3JZ
Scotland, U.K.

ABSTRACT

We present a unification algorithm for the $\lambda\Pi$-calculus, the lambda calculus with first-order dependent function types. Our algorithm is an extension of Huet's algorithm for the simply-typed lambda calculus to first-order dependent function types.

In the simply-typed lambda calculus one attempts to unify a pair of terms of the same type: in the $\lambda\Pi$-calculus types are dependent on terms so one must unify not only terms, but their types as well. Accordingly, we identify a suitable notion of *similarity* of types, and only attempt to unify a pair of terms of similar type: if the types are not similar then they are not unifiable. Since Huet's algorithm does not enumerate all of the unifiers of a given pair of terms, the strategy of first unifying pairs of types — by identifying suitable pairs of subterms for unification — is not complete. Accordingly, we must unify terms and their types simultaneously, taking care to maintain all of the conditions that are necessary to ensure the well-formedness of the resulting calculated substitution.

Our notion of substitution is an algebraic one: substitutions are morphisms of $\lambda\Pi$-contexts. However, in order to define our algorithm we must work with *psubstitutions* and *pcontexts* — substitutions and contexts, respectively, in which variables are replaced by terms of *similar*, not $\beta(\eta)$-equal, type.

Keywords: Unification; lambda calculus; dependent types; higher-order logic; logical frameworks; (syntactic) contextual categories.

*The work of this paper was completed whilst the author was visiting the Center for Study of Language and Information (CSLI), Stanford University, during the Spring quarter of 1988. The author was then supported by a U.K. Science and Engineering Research Council CASE Award in association with the B.P. Research Centre.

1 Introduction

In this paper we present a unification algorithm for the $\lambda\Pi$-calculus, a theory of first-order dependent function types.[1]

There are three important existing works on higher-order unification, namely Pietrzykowski's algorithm for the complete-enumeration of unifiers in second-order logic, Jensen and Pietrzykowski's fundamental paper on the extension of this work to an algorithm for the complete-enumeration of unifiers in ω-order logic (simply-typed λ-calculus) [17] and Huet's seminal work on searching for the existence of unifiers in simply-typed λ-calculus [14], [15]. The work of Huet [14], [15] forms the basis of this paper.

Our algorithm can be used to give a full account of effective proof-search and logic programming in the setting of the $\lambda\Pi$-calculus. This idea is explored in [24] and [23].

In the simply-typed λ-calculus we attempt to unify terms of the same type. However in the $\lambda\Pi$-calculus we must unify not only objects, but their types as well. We must adapt the approach of [14] to allow this.

For example consider the pair of terms

$$\lambda x{:}o\,.\,y(x,u) : \Pi x{:}o\,.\,f(x,u) \tag{1}$$

where $f : o \to o \to \text{Type}$, and

$$\lambda x{:}o\,.\,z(x,v) : \Pi x{:}o\,.\,g(x,w)\,, \tag{2}$$

where $g : o \to o \to \text{Type}$. The types in 1 and 2 are unified if f is substituted for g and u is substituted for w. In general we shall have to unify the types of the Π-abstracted variables as well, but in this case they are both just the constant o. Applying this substitution to the objects, we are left with the problem of unifying

$$\lambda x{:}o\,.\,y(x,u) \quad \text{and} \quad \lambda x{:}o\,.\,z(x,v)\,,$$

which now have the same type. These are unified if u is substituted for v and y is substituted for z.

The differences between the work of this paper and that of [14] may be summarized as follows:

- The need to unify not only objects but their types as well forces us to devise a procedure which reduces the problem of unifying a pair of objects (and their types) to that of unifying a set of pairs of objects. A further complication at this point is that the components of such pairs are not of equal type; although their types are *similar* in a certain technical sense to be defined.
- Considerable extra difficulty in proving that "flexible-flexible" pairs are always unifiable;
- The need to keep track of the variables that have been introduced by the algorithm: we want to know in what context our unified terms are well-typed. This need arises when unifying "flexible-flexible" pairs and in the procedure *match*;

[1] The $\lambda\Pi$-calculus is also known as the type system of the Logical Framework or LF [11], [12], [1].

- The need for the procedure *match* to introduce new disagreement pairs to the disagreement set so that certain types are unified;
- The need to maintain an ordering on disagreement sets so that the unifiability of flexible-flexible disagreement sets can be guaranteed;
- Our notion of substitution is categorical — this is the natural notion in this setting: fortunately, this does not affect the applicability of the methods of [14], although a little more work is required.

Thus the work of this paper is to show that the work of [14] is applicable provided we take care in handling these extra difficulties.

We first provide the necessary syntax, notation and a suitable definition of substitutions; and discuss the nature of unifiers. We then proceed to give a unification algorithm in the style of Huet [14], which tests for the *existence* of unifiers, and returns a complete set of "initial segments" of the unifiers of a pair of $\lambda\Pi$-calculus terms. Of course, this algorithm is, in general, non-terminating: higher-order unification is semi-decidable even in the case of simple types. The undecidability proofs for the theory of simple types [15] can be adapted to the $\lambda\Pi$-calculus.

Finally, we consider how to obtain an algorithm based on that of Jensen and Pietrzykowski [17], for the *complete-enumeration* of the set of unifiers of a pair of $\lambda\Pi$-calculus terms. It is of some theoretical importance that such a procedure exists.

Elliott [9] has also studied unification algorithms for the $\lambda\Pi$-calculus[2], independently, and at the same time as us. His solution is similar to ours, although his presentation diverges from the format of [14] rather more in that it exploits a notion of *unification problem*. Such unification problems are used to successively approximate a solution via *rigid-rigid transformations* and *flexible-rigid transformations*. Elliott [9] considers only $\beta\eta$-equality, whereas we begin with β-equality and observe that, just as in [14], an algorithm for th $\beta\eta$-equality case arises as a simple restriction of the one for β-equality.

2 The $\lambda\Pi$-calculus

2.1 Syntax

We present the $\lambda\Pi$-calculus, a language introduced in [11], [12]. The $\lambda\Pi$-calculus is a language with entities of three levels: *objects*, *types and families of types*, and *kinds*. Objects are classified by types, types and families of types by kinds. The kind Type classifies the types; the other kinds classify functions f which yield a type $f(M_1)\ldots(M_n)$ when applied to objects $M_1, \ldots, M_n$ of certain types determined by the kind of f. Any function definable in the system has a type as domain, while its range can either be a type, if it is an object, or a kind, if it is a family of types. The $\lambda\Pi$-calculus is therefore predicative.

The theory we shall deal with is a formal system for deriving assertions of one of the following shapes:

[2] In fact, Elliott's doctoral thesis [9] is a study of a variety of extensions of the simply-typed notion of higher-order unification, including first-order Σ-types.

$\vdash \Sigma$ sig	Σ is a signature
$\vdash_\Sigma \Gamma$ context	Γ is a context
$\Gamma \vdash_\Sigma K$ kind	K is a kind
$\Gamma \vdash_\Sigma A{:}K$	A has kind K
$\Gamma \vdash_\Sigma M{:}A$	M has type A

where the syntax is specified by the following grammar:

$$\begin{array}{lcl} \Sigma & ::= & \langle\rangle \mid \Sigma, c:K \mid \Sigma, c:A \\ \Gamma & ::= & \langle\rangle \mid \Gamma, x:A \\ K & ::= & \mathrm{Type} \mid \Pi x:A.K \\ A & ::= & c \mid \Pi x:A.B \mid \lambda x:A.B \mid AM \\ M & ::= & c \mid x \mid \lambda x:A.M \mid MN \end{array}$$

We let M and N range over expressions for objects, A and B for types and families of types, K for kinds, x and y over variables, and c over constants. We also allow f, g to range variables where the intention is that, in general, these have higher types.

We refer to the collection of (constants) variables declared in a (signature) context (Σ) Γ as (DomΣ) Dom(Γ). We assume α-conversion throughout. The inference rules of the $\lambda\Pi$-calculus appear in Table 1. We shall refer to this system as N because it is a system of natural deduction. We write N proves $\Gamma \vdash_\Sigma M{:}A$, *etc.*, to denote that the assertion $\Gamma \vdash_\Sigma M{:}A$ is provable in the system N and we shall sometimes write simply $\Gamma \vdash_\Sigma M{:}A$, where no confusion can arise.

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 notion of $\beta(\eta)$-reduction, written $\rightarrow_{\beta(\eta)}$, can be defined both at the level of objects and at the level of types and families of types in the obvious way; for the details, see [12]. $M =_{\beta(\eta)} N$ iff $M \rightarrow^*_{\beta(\eta)} P$ and $N \rightarrow^*_{\beta(\eta)} P$ for some term P, where * denotes transitive closure. For simplicity we shall write $\rightarrow_{\beta(\eta)}$ for $\rightarrow^*_{\beta(\eta)}$. We write NF($U$) to denote the $\beta(\eta)$-normal form of the expression U.[3] We write $U = V$ to denote the α-equality of the expressions U and V and $U \equiv V$ to denote their syntactic identity (with the subscript "def" to denote definitions).

We write $A \rightarrow B$ for $\Pi x{:}A.B$ when x does not occur free in B and $A \rightarrow K$ for $\Pi x{:}A.K$ when x does not occur free in K.

A summary of the major metatheorems pertaining to N and its reduction properties are given by Theorem 2.1.

THEOREM 2.1 (THE BASIC METATHEORY OF THE $\lambda\Pi$-CALCULUS)

Let α range over basic assertions of the form $A{:}K$ and $M{:}A$.

1. Thinning is an admissible rule: if N proves $\Gamma \vdash_\Sigma \alpha$ and N proves $\vdash_{\Sigma,\Sigma'} \Gamma, \Gamma'$ context, then N proves $\Gamma, \Gamma' \vdash_{\Sigma,\Sigma'} \alpha$.

[3] Unless stated otherwise, this will be just β. Theroem 2.1 holds for both β and $\beta\eta$, *q.v.* [11], [12], [27], [6].

2. Transitivity is an admissible rule: if N proves $\Gamma \vdash_\Sigma M{:}A$ and N proves $\Gamma, x{:}A, \Delta \vdash_\Sigma \alpha$, then N proves $\Gamma, \Delta[M/x] \vdash_\Sigma \alpha[M/x]$.

3. Uniqueness of types and kinds: if N proves $\Gamma \vdash_\Sigma M{:}A$ and N proves $\Gamma \vdash_\Sigma M{:}A'$, then $A =_{\beta(\eta)} A'$, and similarly for kinds.

4. Subject reduction: if N proves $\Gamma \vdash_\Sigma M{:}A$ and $M \to^*_{\beta(\eta)} M'$, then N proves $\Gamma \vdash_\Sigma M'{:}A$, and similarly for types.

5. All well-typed terms are strongly normalizing.

6. All well-typed terms are Church-Rosser.

7. Each of the five relations defined by the inference system of Table 1 is decidable, as is the property of being well-typed.

8. Predicativity: if N proves $\Gamma \vdash_\Sigma M : A$ then the type-free λ-term obtained by erasing all type information from M can be typed in the Curry type assignment system ([13], Ch. 15, pp. 203-223).

9. Strengthening is an admissible rule: if N proves $\Gamma, x : A, \Gamma' \vdash_\Sigma \alpha$ and if $x \notin \mathrm{FV}(\Gamma') \cup \mathrm{FV}(\alpha)$ then N proves $\Gamma, \Gamma' \vdash_\Sigma \alpha$. □

The proof of Theorem 2.1 is rather complicated. One method, due to Salvesen [27] (see also [12],) adapts the methods developed by van Daalen in his thesis [7] to this type theory. The main difficulty here lies in obtaining the Church-Rosser property in the presence of η-conversion. The essential step in obtaining the proof of this property is to first reformulate the $\lambda\Pi$-calculus as a system with equality judgements in which type labels are explicit, *i.e.*, the assertions of equality have shape $\Gamma \vdash_\Sigma M = N : A$, *etc.*. This step is sufficient to allow the methods of van Daalen to go through.

Valid Signature

$$\frac{}{\vdash \langle\rangle \ \text{sig}} \tag{3}$$

$$\frac{\vdash \Sigma \ \text{sig} \quad \vdash_\Sigma K \ \text{kind} \quad c \notin \text{Dom}(\Sigma)}{\vdash \Sigma, c : K \ \text{sig}} \tag{4}$$

$$\frac{\vdash \Sigma \ \text{sig} \quad \vdash_\Sigma A\text{: Type} \quad c \notin \text{Dom}(\Sigma)}{\vdash \Sigma, c : A \ \text{sig}} \tag{5}$$

Valid Context

$$\frac{\vdash \Sigma \ \text{sig}}{\vdash_\Sigma \langle\rangle \ \text{context}} \tag{6}$$

$$\frac{\vdash_\Sigma \Gamma \ \text{context} \quad \Gamma \vdash_\Sigma A\text{: Type} \quad x \notin \text{Dom}(\Gamma)}{\vdash_\Sigma \Gamma, x : A \ \text{context}} \tag{7}$$

Valid Kinds

$$\frac{\vdash_\Sigma \Gamma \ \text{context}}{\Gamma \vdash_\Sigma \text{Type kind}} \tag{8}$$

$$\frac{\Gamma \vdash_\Sigma A\text{: Type} \quad \Gamma, x : A \vdash_\Sigma K \ \text{kind}}{\Gamma \vdash_\Sigma \Pi x : A.K \ \text{kind}} \tag{9}$$

Valid Elements of a Kind

$$\frac{\vdash_\Sigma \Gamma \ \text{context} \quad c : K \in \Sigma}{\Gamma \vdash_\Sigma c\text{:} K} \tag{10}$$

$$\frac{\Gamma \vdash_\Sigma A\text{: Type} \quad \Gamma, x : A \vdash_\Sigma B\text{: Type}}{\Gamma \vdash_\Sigma \Pi x : A.B\text{: Type}} \tag{11}$$

$$\frac{\Gamma \vdash_\Sigma A\text{: Type} \quad \Gamma, x : A \vdash_\Sigma B\text{:} K}{\Gamma \vdash_\Sigma \lambda x : A.B\text{:} \Pi x : A.K} \tag{12}$$

$$\frac{\Gamma \vdash_\Sigma B\text{:} \Pi x : A.K \quad \Gamma \vdash_\Sigma N\text{:} A}{\Gamma \vdash_\Sigma BN\text{:} K[N/x]} \tag{13}$$

$$\frac{\Gamma \vdash_\Sigma A\text{:} K \quad \Gamma \vdash_\Sigma K' \ \text{kind} \quad K =_{\beta(\eta)} K'}{\Gamma \vdash_\Sigma A\text{:} K'} \tag{14}$$

Valid Elements of a Type

$$\frac{\vdash_\Sigma \Gamma \ \text{context} \quad c : A \in \Sigma}{\Gamma \vdash_\Sigma c\text{:} A} \tag{15}$$

$$\frac{\vdash_\Sigma \Gamma \ \text{context} \quad x : A \in \Gamma}{\Gamma \vdash_\Sigma x\text{:} A} \tag{16}$$

$$\frac{\Gamma \vdash_\Sigma A\text{: Type} \quad \Gamma, x : A \vdash_\Sigma M\text{:} B}{\Gamma \vdash_\Sigma \lambda x : A.M\text{:} \Pi x : A.B} \tag{17}$$

$$\frac{\Gamma \vdash_\Sigma M\text{:} \Pi x : A.B \quad \Gamma \vdash_\Sigma N\text{:} A}{\Gamma \vdash_\Sigma MN\text{:} B[N/x]} \tag{18}$$

$$\frac{\Gamma \vdash_\Sigma M\text{:} A \quad \Gamma \vdash_\Sigma A'\text{: Type} \quad A =_{\beta(\eta)} A'}{\Gamma \vdash_\Sigma M\text{:} A'} \tag{19}$$

Table 1.

The reader is referred to [27] for the details of the proof.[4] Similar properties are proved for the system with just β-reduction in [12]. We note that in this proof in order to obtain Part 8, we first prove decidability. In this proof, we prove the Church-Rosser property, strong normalization and require the presence of type labels in order to prove the decidability of the type theory.

An alternative approach to the proof of the Church-Rosser property, due to Coquand, is given in [6].

Following [11] we outline only the proof of strong normalization, Theorem 2.1, Part 5, since it can be proved independently of the other metatheoretic results. Moreover it yields a corollary that is useful for characterizing the terms that are definable in the $\lambda\Pi$-calculus.

We define a translation $\mathcal{T}$ of the $\lambda\Pi$-calculus types and kinds into S, the set of simple types over a base type ω, and a translation $|-|$ of $\lambda\Pi$-calculus type families and objects into $\Lambda(K)$, the set of untyped λ-terms with a set of constants $K = \{\pi_\sigma | \sigma \in S\}$. These translations are extended to signatures and contexts in the obvious way.

DEFINITION 2.2 (TRANSLATION TO SIMPLE TYPES)

The definition of the translation divides conveniently into two clauses:

- The $\mathcal{T}$-clause:
  - Kinds
$$\begin{aligned}\mathcal{T}(\text{ Type}) &= \omega\\ \mathcal{T}(\Pi x{:}A.K) &= \tau(A) \to \tau(K)\end{aligned}$$
  - Types and type families
$$\begin{aligned}\mathcal{T}(c) &= \omega\\ \mathcal{T}(\Pi x{:}A.B) &= \mathcal{T}(A) \to \mathcal{T}(B)\\ \mathcal{T}(\lambda x{:}A.B) &= \mathcal{T}(B)\\ \mathcal{T}(AM) &= \mathcal{T}(A)\end{aligned}$$
- The $|-|$-clause:
  - Types and type families
$$\begin{aligned}|c| &= c\\ |\Pi x{:}A.B| &= \pi_{\mathcal{T}(A)}|A|(\lambda x.|B|)\\ |\lambda x{:}A.B| &= (\lambda y.\lambda x.|B|)|A| \quad (y \notin \mathrm{FV}(\lambda x.|B|))\\ |AM| &= |A|\,|M|\end{aligned}$$

    where we assume that x and y are distinct.

[4] Harper [10] also considers an equational formulation of the $\lambda\Pi$-calculus as a basis for the construction of *environment models* [18] of the $\lambda\Pi$-calculus.

– Objects

$$\begin{array}{ll} |c| & = c \\ |x| & = x \\ |\lambda x{:}A.M| & = (\lambda y.\lambda x.|M|)|A| \quad (y \notin \mathrm{FV}(\lambda x.|M|)) \\ |MN| & = |M|\,|N| \end{array}$$

where we assume that x and y are distinct. □

The precise sense in which this definition is consistent is stated in the following lemma:

LEMMA 2.3 (TRANSLATION CONSISTENCY)

1. If N proves $\Gamma \vdash_\Sigma A{:}K$, then $\mathcal{T}(\Gamma) \vdash_{\mathcal{T}(\Sigma),\{\pi_\sigma|\pi_\sigma:\omega\to(\sigma\to\omega)\to\omega\}} |A|{:}\mathcal{T}(K)$ in Curry's type assignment system ([13], Ch. 15, pp. 203-223).
2. If N proves $\Gamma \vdash_\Sigma M{:}A$, then $\mathcal{T}(\Gamma) \vdash_{\mathcal{T}(\Sigma),\{\pi_\sigma|\pi_\sigma:\omega\to(\sigma\to\omega)\to\omega\}} |M|{:}\mathcal{T}(A)$ in Curry's type assignment system (*ibid.*, 1).

PROOF

By induction on the structure of the proof of $\Gamma \vdash_\Sigma A{:}K$ and $\Gamma \vdash_\Sigma M{:}A$. We have only to notice that any derivation of $\vdash_\Sigma \Gamma, x{:}A$ context contains as a subderivation a derivation of $\Gamma \vdash_\Sigma A{:}$ Type. □

The translation has been carried out in such a way that the extra combinatorial complexity in the λΠ-calculus terms due to the presence of type labels is not lost. We then have the following result:

LEMMA 2.4 (REDUCTION UNDER TRANSLATION)

1. If $A \to_{\beta(\eta)} A'$, then $|A| \to^*_{\beta(\eta)} |A'|$;
2. If $M \to_{\beta(\eta)} M'$, then $|M| \to^*_{\beta(\eta)} |M'|$.

(We use $\to^*_{\beta(\eta)}$ to denote the transitive closure of the reduction $\to_{\beta(\eta)}$ of the simply-typed terms, just as for the λΠ-calculus, as stated on p. 4.)
PROOF

By induction on the derivation of $A \to A'$ and $M \to M'$. □

Now since the Curry typable terms are strongly normalizing, so too are the terms of the λΠ-calculus. For more information concerning these arguments the reader is referred to [12].

Moreover, it can be easily seen that $|A| \to^*_{\beta(\eta)} \mathit{erase}(A)$, and $|M| \to^*_{\beta(\eta)} \mathit{erase}(M)$, where $\mathit{erase}(M)$ denotes the term obtained from M by removing the type labels from the λ-abstractions, so that we obtain the following corollary, which characterizes the terms that are definable in the λΠ-calculus:

COROLLARY 2.5 (DEFINABILITY CHARACTERIZATION (PREDICATIVITY))

If N proves $\Gamma \vdash_\Sigma M{:}A$, then $erase(M)$ can be typed in Curry's type assignment system. □

The translation of the $\lambda\Pi$-calculus into simple types described above is of crucial importance for the definition of a unification algorithm for the $\lambda\Pi$-calculus. We shall return to this point in Section 2.4.

Finally, we note that it is possible to define a notion of *long $\beta\eta$-normal form.* Suppose that the object

$$\lambda x_1{:}A_1 \ldots \lambda x_m{:}A_m \,.\, @M_1 \ldots M_p \tag{20}$$

is in $\beta\eta$-normal form and that

$$@ : \Pi y_1{:}B_1 \ldots \Pi y_q{:}B_q \,.\, B,$$

where $q > p$. The *long $\beta\eta$-normal form* of the object (20) is the object

$$\lambda x_q{:}B_q \ldots \lambda x_{p+1}{:}B_{p+1} \,.\, \lambda x_1{:}A_1 \ldots \lambda x_m{:}A_m \,.\, @M_1 \ldots M_p x_{p+1} \ldots x_q \,.$$

Long $\beta\eta$-normal forms in dependent type theory are studied carefully in [8] (see also [12]). This notion is required for Section 3.8.

2.2 A Syntactic Category of Contexts and Substitutions

We define a syntactic category in which the objects are contexts and the morphisms are substitutions. Such a category is a *contextual category with products*, a notion defined by Cartmell [3], [4] and developed by Streicher [28]. Contextual categories are presented in Appendix A. A survey of categorical descriptions of dependent types is provided in [16]. We adopt contextual categories because they are both adequate for the purposes of this paper and elementary.

DEFINITION 2.6 (SUBSTITUTIONS)

Let the contexts Γ and Δ be given by $\Gamma \equiv x_1{:}A_1, \ldots, x_m{:}A_m(x_1, \ldots, x_{m-1})$ and $\Delta \equiv y_1{:}B_1, \ldots, y_n{:}B_n(y_1, \ldots, y_{n-1})$.

A *substitution* is a morphism $\rho : \Gamma \longrightarrow \Delta$ in which we call Δ the *domain context* and Γ the *range context*[5], given by a finite tuple of *substitution pairs*:

$$\rho \equiv \langle \langle M_1, y_1 \rangle, \ldots, \langle M_n, y_n \rangle \rangle,$$

such that N proves $\Gamma \vdash_\Sigma M_i{:}B_i[M_1/y_1, \ldots, M_{i-1}/y_{i-1}]$ for $1 \leq i \leq n$.[6] [7] □

[5] Alternatively, the *codomain context.*

[6] We take tuples of *pairs* here because this is both standard in the presentation of unification and pedagogically helpful. However, when discussing the algebraic properties of substitutions this can be cumbersome, and in such cases we drop the variables in this definition, writing $\rho \equiv \langle M_1 \ldots M_n \rangle$.

[7] Cartmell [3], [4] calls substitutions *realizations*, in the tradition of categorical logic, which is appropriate in the setting of [3], [4].

DEFINITION 2.7 (APPLICATION OF SUBSTITUTIONS)

Suppose that M and A are well-typed in the context Δ. The application of the substitution

$$\rho(\equiv \langle\langle M_1, y_1\rangle, \ldots, \langle M_n, y_n\rangle\rangle) : \Gamma \longrightarrow \Delta,$$

where $\Gamma \equiv x_1 : A_1, \ldots, x_m : A_m$ and $\Delta \equiv y_1 : B_1, \ldots, y_n : B_n$, to M and A is given by, respectively,

$$M\rho =_{\text{def}} \text{NF}((\lambda y_1 : B_1 \ldots \lambda y_n : B_n \,.\, M)M_1 \ldots M_n)$$

and

$$A\rho =_{\text{def}} \text{NF}((\lambda y_1 : B_1 \ldots \lambda y_n : B_n \,.\, A)M_1 \ldots M_n). \quad \square$$

LEMMA 2.8 (APPLICATION OF SUBSTITUTIONS)

If $\rho : \Gamma \longrightarrow \Delta$ is a substitution, and if N proves $\Delta \vdash_\Sigma M : A$, then N proves $\Gamma \vdash_\Sigma M\rho : A\rho$.
PROOF

Straightforward. $\square$

The definition of substitution taken here is slightly different from that taken in [14]. The reason for this is that the presence of dependent types makes the definition taken in [14] rather awkward because it is difficult to define the range context in a simple manner. The categorical definition is the natural one.

We define the syntactic category $C(\Sigma)$ in which the objects are valid $\lambda\Pi$-calculus contexts over a given valid signature Σ, and in which the morphisms are substitutions. This category corresponds exactly to the syntax of the $\lambda\Pi$-calculus over Σ, as determined by the rules of Table 1. For the remainder of this section we identify expressions up to $\alpha\beta\eta$-equality.

DEFINITION 2.9 (THE CATEGORY $C(\Sigma)$)

Let Σ be a given valid $\lambda\Pi$-calculus signature, and let $\Gamma \equiv x_1 : A_1, \ldots, x_m : A_m$ and $\Delta \equiv y_1 : B_1, \ldots, y_n : B_n$ be valid $\lambda\Pi$-calculus contexts over Σ.

- The objects of $C(\Sigma)$ are valid $\lambda\Pi$-calculus contexts over Σ.
- The morphisms of $C(\Sigma)$ are substitutions $\langle M_1, \ldots, M_n\rangle : \Gamma \longrightarrow \Delta$, where Γ and Δ are contexts.
  - The identity morphism on the object $\Gamma \equiv x_1 : A_1, \ldots, x_m : A_m$ is the substitution $\langle x_1, \ldots, x_m\rangle$.
  - Let Γ and Δ and the substitution $\langle M_1, \ldots, M_n\rangle$ be defined as above. Let $\Theta \equiv z_1 : C_1, \ldots, z_p : C_p$ be a context, and let $\langle N_1, \ldots, N_p\rangle$ be a morphism $\Delta \longrightarrow \Theta$. Composition in $C(\Sigma)$ is defined by

$$\langle M_1, \ldots, M_n\rangle \circ \langle N_1, \ldots, N_p\rangle =_{\text{def}}$$

$$\langle N_1[M_1/y_1,\ldots,M_n/y_n],\ldots,N_p[M_1/y_1,\ldots,M_n/y_n]\rangle\,.$$

□

The reader is referred to the work of Cartmell [3], [4] and Streicher [28] for the proofs of the next two theorems.[8]

THEOREM 2.10 ($C(\Sigma)$ IS A CONTEXTUAL CATEGORY)

The category $C(\Sigma)$ is structured as a contextual category as follows:

- The least element ϵ is the empty context, $\langle\rangle$;
- Let $\Gamma \equiv x_1 : A_1,\ldots,x_m : A_m$ and $\Delta \equiv \Gamma, x_{m+1} : A_{m+1},\ldots,x_{m+n} : A_{m+n}$ be contexts. The canonical morphism $p(\Delta,\Gamma) : \Delta \longrightarrow \Gamma$ is defined by $p(\Delta,\Gamma) \equiv_{\text{def}} \langle x_1,\ldots,x_m\rangle$;
- Let $\Gamma \equiv x_1{:}A_1,\ldots,x_m{:}A_m$, $\Gamma' \equiv y_1{:}B_1,\ldots,y_n{:}B_n$, and $\Delta \equiv y_1{:}B_1,\ldots,y_n : B_n, y_{n+1} : B_{n+1},\ldots,y_{n+p} : B_{n+p}$ and suppose that $f \equiv \langle M_1,\ldots,M_n\rangle : \Gamma \longrightarrow \Gamma'$ is a substitution for Γ' with respect to Γ, then:
  - $f^*\Delta \equiv_{\text{def}} x_1{:}A_1,\ldots,x_m{:}A_m, y_{n+1}{:}B_{n+1}[M_1/y_1,\ldots,M_n/y_n],\ldots, y_{n+p}{:}B_{n+p}[M_1/y_1,\ldots,M_n/y_n]$;
  - $q(f,\Delta) \equiv_{\text{def}} \langle M_1,\ldots,M_n,y_{n+1},\ldots,y_{n+p}\rangle$. □

THEOREM 2.11 ($C(\Sigma)$ IS A CONTEXTUAL CATEGORY WITH PRODUCTS)

Let Σ be a valid $\lambda\Pi$-signature. In the presence of β-equality the category $C(\Sigma)$ is structured as a contextual category with weak products, and in the presence of $\beta\eta$-equality as a contextual category with products, as follows:
If $\Gamma \equiv x_1{:}A_1,\ldots,x_m{:}A_m, x{:}A, y{:}B$ is a valid context over Σ, then:

- $\Pi(\Gamma) \equiv_{\text{def}} x_1{:}A_1,\ldots,x_m{:}A_m, x : \Pi x{:}A\,.\,B$;
- $\text{eval}_\Gamma \equiv_{\text{def}} \langle x_1,\ldots,x_m,x,yx\rangle : \Delta, x{:}A, y{:}\Pi x{:}A\,.\,B \longrightarrow \Delta$, where $\Delta \equiv x_1{:}A_1,\ldots,x_m{:}A_m$. □

The next theorem summarizes the equivalence of the logical and algebraic descriptions of the syntax of the $\lambda\Pi$-calculus; the reader is referred to [3], [4] and [28] for its proof.[9]

THEOREM 2.12 (CORRECTNESS OF $C(\Sigma)$)

Let Σ be a valid $\lambda\Pi$-signature. Let $\Gamma \equiv x_1{:}A_1,\ldots,x_m{:}A_m$ and $\Delta \equiv y_1{:}B_1,\ldots,y_n{:}B_n$ be valid Σ-$\lambda\Pi$-contexts. The tuple $\langle M_1,\ldots,M_n\rangle : \Gamma \longrightarrow \Delta$ is a morphism in $C(\Sigma)$ if and only if N proves $\Gamma \vdash_\Sigma M_i{:}B_i[M_1/y_1,\ldots,M_{i-1}/y_{i-1}]$ for each $1 \le i \le n$. □

[8] [3] and [4] are the basic references for contextual categories; the proofs given in [28] are of similar theorems for the Calculus of Constructions [5]; the proofs for the $\lambda\Pi$-calculus are similar and simpler.

[9] Again, [3] and [4] are the basic references for contextual categories; the proof given in [28] is of a similar theorem for the Calculus of Constructions [5]; the proof for the $\lambda\Pi$-calculus is similar and simpler.

2.3 Unifiers

Now that we have defined substitutions for the $\lambda\Pi$-calculus we are able to give a definition of *unifier* for the $\lambda\Pi$-calculus.

DEFINITION 2.13 (UNIFIERS)

Suppose that N proves $\Delta \vdash_\Sigma M:A$ and N proves $\Delta \vdash_\Sigma N:B$. The substitution $\rho : \Gamma \longrightarrow \Delta$ is a *unifier* of $M{:}A$ and $N{:}B$ if $A\rho =_{\beta(\eta)} B\rho$ and $M\rho =_{\beta(\eta)} N\rho$.

We let $\mathcal{U}_\Delta(M{:}A, N{:}B)$ denote the set of all unifiers of the terms $M{:}A$ and $N{:}B$. We omit the subscript Δ where no confusion can arise by doing so. □

It is interesting to consider what algebraic structure, in the contextual category $C(\Sigma)$, can be considered to characterize unifiers.

Suppose that N proves $\Delta \vdash_\Sigma M{:}A$ and N proves $\Delta \vdash_\Sigma N{:}B$, then we have the maps

$$\sigma \equiv \langle y_1, \ldots, y_n, M\rangle : \Delta \longrightarrow \Delta, y{:}A$$

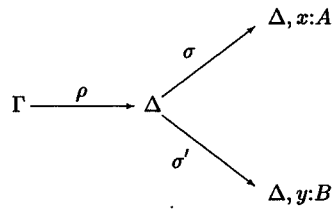
and

$$\sigma' \equiv \langle y_1, \ldots, y_n, N\rangle : \Delta \longrightarrow \Delta, y{:}B$$

in the category $C(\Sigma)$. If

$$\rho \equiv \langle P_1, \ldots, P_n\rangle : \Gamma \longrightarrow \Delta$$

is a unifier of $M{:}A$ and $N{:}B$, then we have that, under suitable equality,



is a diagram in $C(\Sigma)$, and we have that

$$\langle P_1, \ldots, P_n, M\rho\rangle : \Gamma \longrightarrow \Delta, y{:}A\rho$$

and

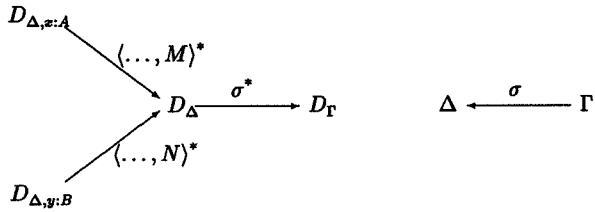
$$\langle P_1, \ldots, P_n, N\rho\rangle : \Gamma \longrightarrow \Delta, y{:}B\rho$$

are morphisms in the category $C(\Sigma)$, and that $M\rho = N\rho$ and $A\rho = B\rho$. Moreover, we have that N proves $\Gamma \vdash_\Sigma M\rho{:}A\rho$ and N proves $\Gamma \vdash_\Sigma N\rho{:}B\rho$.

This is in contrast to the situation in first-order logic, where the existence of *most general unifiers* means that unifiers are characterized by *coequalizers* in an appropriate syntactic category [25], [26]. Note that even if $\Delta, y{:}A$ and $\Delta, y{:}B$ were equal, the map ρ would be an equalizer rather than a coequalizer. This is

because the view of the λΠ-calculus as a contextual category does not give an explicit account of the entailment relation $\vdash_\Sigma$. Such an account of this relation can be given by viewing the λΠ-calculus as an *indexed category*[10] in which the objects of the base are contexts, with substitutions as morphisms, and the objects of the fibre over a given context are the types that are inhabited in that context: if $\sigma : \Gamma \longrightarrow \Delta$ is a substitution, then there is a map (also known as substitution) $\sigma^* : D_\Delta \longrightarrow D_\Gamma$, where D_Γ denotes the fibre over Γ: but note that σ^* is a map in the opposite direction on the fibres (we return to this point below). In this setting, unifiers (when defined precisely) are on the appropriate side of the diagram to be coequalizers although, of course, the need to unify types and the non-existence of most general unifiers means that unifiers fail to be characterized by coequalizers.

The reader might consider why, in the definition of a substitution $\rho : \Gamma \longrightarrow \Delta$, Definition 2.6, we refer to Δ as the domain context and Γ as the range context. To see why consider the diagram below (informally, in a suitable $C(\Sigma)$-indexed category). We assume that $\Delta \vdash_\Sigma M{:}A$, $\Delta \vdash_\Sigma N{:}B$, $\Gamma \vdash_\Sigma M\sigma{:}A\sigma$, $\Gamma \vdash_\Sigma N\sigma{:}B\sigma$ and that $M\sigma = N\sigma$ and $A\sigma = B\sigma$.



Although the unifier that we wish to calculate, σ, exists in the base category $C(\Sigma)$, the application of σ occurs via the map σ^*, a map in the opposite direction on the fibres.

With regard to the non-existence of most general unifiers, Rydeheard and Burstall [25] identify, in the setting of simple types, the notion of a *coequalizing set* and are able to recover certain of their theorems for coequalizing first-order most general unifiers.

It remains an open problem to give an adequate categorical characterization of higher-order unification.

2.4 Similarity and Syntax

The requirement that we must unify types means that in order to construct unifiers we shall need a slightly weaker notion of substitution. We develop a suitable notion. Elliott [9] encounters a similar problem.

Our unification algorithm, as presented in Section 3, uses a procedure *match* that is similar to the procedure *MATCH* of [14].

In Huet's *MATCH*, *imitation* and *projection* construct terms that are substituted for a variable that occurs at the head of a given term, and in this setting

[10] See Appendix A, Section A.3, Definition A.13.

the constructed term has the same type as the variable for which it is substituted.

In our setting the objects constructed by *imitation* and *projection* cannot be assumed to have the same type as the variables for which they are substituted. For example, suppose that we are faced, in the manner of Huet's algorithm [14], with a disagreement pair $\langle cM_1N_1, cM_2N_2\rangle$. Suppose further that the two objects cM_1N_1 and cM_2N_2 have that same type. Now, in the manner of Huet's algorithm [14], we simplify this disagreement pair to the two disagreement pairs $\langle M_1, M_2\rangle$ and $\langle N_1, N_2\rangle$ and the procedure *match* applies to each of these pairs. But N_1 and N_2 might have different types; for example, if the type of c is of the form $\Pi x\!:\!A\,.\,(dx) \to e$ then N_1 has type dM_1 and N_2 has type dM_2, for suitable A, d and e.

However, we can ensure that the type of an object that is constructed by *match* is *similar* to the type of the variable for which it is substituted, in the following sense:

DEFINITION 2.14 (SIMILAR TYPES)

Let the types A and B be given by the expressions:

$$A \equiv \Pi x_1\!:\!A_1 \ldots \Pi x_{n_1}\!:\!A_{n_1}.c_1M_1 \ldots M_{q_1}\,,$$

$$B \equiv \Pi y_1\!:\!B_1 \ldots \Pi y_{n_2}\!:\!B_{n_2}.c_2N_1 \ldots N_{q_2}\,,$$

where the kinds of c_1 and c_2 have respectively q_1 and q_2 Π-abstractions. The types A and B are said to be *similar*, written $A \sim B$, if $n_1 = n_2(= n)$, $c_1 = c_2(= c)$, $q_1 = q_2(= q)$[11] and for $1 \leq i \leq n$, $A_i \sim B_i$. □

We now define the notions of *pobject*, *ptype* and *pcontext* and *psubstitution* by allowing the replacement of free variables in objects, types and contexts by objects of similar, not equal, type. Recall the translations $\mathcal{T}$ and $|-|$ of Definition 2.2, of $\lambda\Pi$-calculus-terms into Curry–typable terms.

DEFINITION 2.15 (PCONTEXTS)

We define the notion of *pcontext* inductively as follows:

- (Valid) contexts are pcontexts;
- If $\Gamma \equiv \Gamma_1, x : A, \Gamma_2$ is a pcontext, if N, C and the pcontext Γ_1, Θ, where $\mathrm{Dom}(\Theta) \cap \mathrm{Dom}(\Gamma_2) = \emptyset$, are such that

  $$\mathcal{T}(\Gamma_1, \Theta) \vdash_{\mathcal{T}(\Sigma),\{\pi_\sigma : \omega \to (\sigma\to\omega)\to\omega\}} |N| : \mathcal{T}(C)$$

  is provable in Curry's type assignment system (*cf.* Lemmas 2.3 and 2.4) and if $C \sim A$, then $\Gamma_1, \Theta, \Gamma_2[N/x]$ is a pcontext. □

DEFINITION 2.16 (PSUBSTITUTIONS)

[11] Note that we have required only the given syntactic shapes; we have not required that the expressions be well-typed of kind Type (in a given context).

We define the notion of *psubstitution* inductively as follows:

- A substitution $\sigma : \Gamma \longrightarrow \Delta$ is a psubstitution;
- Let $\sigma : \Gamma_1, x:A, \Gamma_2 \longrightarrow \Delta$ be a psubstitution. If N, C and the pcontext Γ_1, Θ, where $\mathrm{Dom}(\Theta) \cap \mathrm{Dom}(\Gamma_1) = \emptyset$, are such that

  $$\mathcal{T}(\Gamma_1, \Theta) \vdash_{\mathcal{T}(\Sigma),\{\pi_\sigma | \pi_\sigma : \omega \to (\sigma \to \omega) \to \omega\}} |N| : \mathcal{T}(C)$$

  is provable in Curry's type assignment system and if $C \sim A$ then

  $$\sigma[N/x] : \Gamma_1, \Theta, \Gamma_2[N/x] \longrightarrow \Delta$$

  is a psubstitution from the *domain pcontext* Δ to the *range pcontext* $\Gamma_1, \Theta, \Gamma_2[N/x]$. □

We shall refer to the components of psubstitutions and pcontexts respectively as *pobjects* and *ptypes* and given a pobject M we shall write $\tau(M)$ for a ptype A such that, for some pcontext Γ,

$$\mathcal{T}(\Gamma) \vdash_{\mathcal{T}(\Sigma),\{\pi_\sigma | \pi_\sigma : \omega \to (\sigma \to \omega) \to \omega\}} |M| : \mathcal{T}(A)$$

is provable in Curry's type assignment system. Lemmas 2.3 and 2.4 will guarantee the existence of *head normal forms* for these entities: recall that a λ-term is said to be in head normal form if it is of the form

$$\lambda x_1 : A_1 \ldots x_m : A_m \,.\, @ M_1 \ldots M_n \,,$$

and note that this notion extends to the ptypes of the $\lambda\Pi$-calculus in the obvious way. We denote the head normal form of the expression U by $\mathrm{HNF}(U)$.

A psubstitution is said to be in head normal form if each of its components is in head normal form. A pcontext is said to be in head normal form if each of its components is in head normal form.

DEFINITION 2.17 (HEADINGS AND BARE HEADINGS; RIGID AND FLEXIBLE POBJECTS)

Consider the pobject M in head normal form given by:

$$M \equiv \lambda x_1 : A_1 \ldots \lambda x_m : A_m . @ M_1 \ldots M_p \,.$$

We call the *heading* of the pobject M the expression $\lambda x_1 : A_1 \ldots \lambda x_m : A_m \,.\, @$, and call @ the *head* of M. We call the expression $\lambda x_1 \ldots \lambda x_m \,.\, @$ (the heading of M with the type-labels deleted) the *bare heading* of M. If @ is a constant or one of the variables $x_1, \ldots, x_m$, then M is said to be *rigid*; otherwise M is said to be *flexible*. Let A be a ptype given by the expression: $\Pi x_1 : A_1 \ldots \Pi x_n : A_n \,.\, c N_1 \ldots N_q$. We call the *heading* of the ptype A the expression $\Pi x_1 : A_1 \ldots \Pi x_n : A_n \,.\, c$, and call c the *head* of A and $N_1 \ldots N_q$ the *matrix* of A. We call the expression $\Pi x_1 \ldots \Pi x_m \,.\, @$ (the heading of A with the type-labels deleted) the *bare heading* of A. □

The Church–Rosser and strong normalization properties fail for pobjects and ptypes, but the next lemma summarizes the properties that we need.

LEMMA 2.18 (HEAD NORMAL FORMS)

Every pobject (and every ptype) has a head normal form, and every head normal form of a given pobject has the same head.

PROOF SKETCH

The proof proceeds by translating the $\lambda\Pi$-calculus objects and types into Curry–typable terms via the translations $\mathcal{T}$ and $|-|$ of Definition 2.2, and is similar to Lemmas 2.3 and 2.4. Reductions in the Curry-typed terms are mimicked in the $\lambda\Pi$-calculus-terms. See also [11] and [12]. □

DEFINITION 2.19 (THE MEASURE π)

As in [14], let $\pi(M) \in \mathbb{N}$ denote the number of applications in the pobject M (in head normal form):

$$\pi(\lambda x_1 : A_1 \ldots \lambda x_n : A_n \,.\, @M_1 \ldots M_p) =_{\text{def}} p + \sum_{i=1}^{p} \pi(M_i)\,. \;\square$$

DEFINITION 2.20 (APPLICATION OF PSUBSTITUTIONS)

Let M be a pobject and A be a ptype. Let Δ be a pcontext such that

$$\mathcal{T}(\Delta) \vdash_{\mathcal{T}(\Sigma),\{\pi_\sigma : \omega \to (\sigma \to \omega) \to \omega\}} |M| : \mathcal{T}(B)$$

for some B, and

$$\mathcal{T}(\Delta) \vdash_{\mathcal{T}(\Sigma),\{\pi_\sigma : \omega \to (\sigma \to \omega) \to \omega\}} |A| : \omega$$

are provable in Curry's type assignment system. The application of the psubstitution

$$\rho(\equiv \langle\langle M_1, y_1\rangle, \ldots, \langle M_n, y_n\rangle\rangle) : \Gamma \longrightarrow \Delta\,,$$

where $\Gamma \equiv x_1 : A_1, \ldots, x_m : A_m$ and $\Delta \equiv y_1 : B_1, \ldots, y_n : B_n$, to M and A is given by, respectively,

$$M\rho =_{\text{def}} \text{HNF}((\lambda y_1 : B_1 \ldots \lambda y_n : B_n \,.\, M)M_1 \ldots M_n)$$

and

$$A\rho =_{\text{def}} \text{HNF}((\lambda y_1 : B_1 \ldots \lambda y_n : B_n \,.\, A)M_1 \ldots M_n). \;\square$$

It is a banality to check that for such a psubstitution, assertions of the form

$$\mathcal{T}(\Gamma) \vdash_{\mathcal{T}(\Sigma),\{\pi_\sigma : \omega \to (\sigma \to \omega) \to \omega\}} |M\rho| : \mathcal{T}(B\rho)$$

etc., are provable in Curry's type assignment system, *cf.* Lemma 2.8.

DEFINITION 2.21 (EQUIVALENCE OF PSUBSTITUTIONS)

Let $\sigma : \Gamma \longrightarrow \Delta$ and $\rho : \Gamma \longrightarrow \Delta$ be psubstitutions. We write $\sigma =_\Delta \rho$ if HNF(σ) = HNF(ρ), defined componentwise.[12] □

We need a lemma which says that rigid pobjects and ptypes keep their heading essentially unchanged under the application of substitutions.

LEMMA 2.22 (PROPERTIES OF PSUBSTITUTIONS)

1. A rigid pobject keeps its head and the number of abstractions in its heading (and in its bare heading) unchanged under the application of any psubstitution.
2. A ptype keeps its head and the number of abstractions in its heading (and in its bare heading) unchanged under the application of any psubstitution.

PROOF

Immediate from the definitions . □

DEFINITION 2.23 (COMPOSITION OF PSUBSTITUTIONS)

Suppose that the pcontexts Γ and Δ are respectively given by

$$\Gamma \equiv x_1{:}A_1, \ldots, x_m{:}A_m \text{ and } \Delta \equiv y_1{:}B_1, \ldots, y_n{:}B_n\,,$$

and that $\rho : \Gamma \longrightarrow \Delta$ is the psubstitution given by

$$\rho \equiv \langle\langle M_1, y_1\rangle, \ldots, \langle M_n, y_n\rangle\rangle\,,$$

and that $\sigma : \Delta \longrightarrow \Theta$ is the psubstitution given by

$$\sigma \equiv \langle\langle N_1, z_1\rangle, \ldots, \langle N_p, z_p\rangle\rangle$$

where Θ is the pcontext $\Theta \equiv z_1 : C_1, \ldots, z_p : C_p(z_1, \ldots, z_{p-1})$. The composite psubstitution,

$$\rho \circ \sigma : \Gamma \longrightarrow \Theta\,,$$

is the psubstitution given by

$$\rho \circ \sigma \equiv \langle\langle N_1[M_1/y_1, \ldots, M_n/y_n], z_1\rangle, \ldots, \langle N_p[M_1/y_1, \ldots, M_n/y_n], z_p\rangle\rangle\,. \,\square$$

DEFINITION 2.24 (ORDERING PSUBSTITUTIONS)

Let $\sigma : \Gamma \longrightarrow \Delta$ and $\rho : \Gamma \longrightarrow \Theta$ be psubstitutions. We say that σ is *less general than* ρ on Γ, and write $\sigma \leq_\Gamma \rho$, if and only if there exists a psubstitution $\eta : \Theta \longrightarrow \Delta$ such that $\sigma =_\Gamma \rho \circ \eta$.[13] □

[12] Although the subscript is technically unnecessary in this setting, we leave it in place (*cf.* [14]) because we believe it is of some pedagogic value in the presentation of many of the proofs of this paper.

[13] As in Definition 2.21, although the subscript is technically unnecessary in this setting, we leave it in place (*cf.* [14]) because we believe it is of some pedagogic value in the presentation of many of the proofs of this paper.

3 An Existence Algorithm

3.1 Outline

Wherever it is possible we shall follow the structure, style and notation of Huet's presentation of the unification algorithm for the simply-typed λ-calculus, which is found in [14]. The reason for this approach is that many of the proofs and arguments needed in our work are very similar to those needed in [14] and it is therefore useful to be totally explicit about this so that our necessary additional work is clearly seen.

As in [14], we are interested in the existence of unifiers for $M:A$ and $N:B$. To this end, we construct a tree, called a *matching tree*, for $M:A$ and $N:B$. We need the definitions below.

DEFINITION 3.1 (DISAGREEMENT SETS AND DISAGREEMENT PAIRS)

A *disagreement set* is any finite set of pairs of pobjects of similar ptype (called *disagreement pairs*),

$$\{\langle M_i, N_i\rangle | \tau(M_i) \sim \tau(N_i)\ ,\ 1 \leq i \leq n\}.$$

Pairs $\langle M, N\rangle$ in which both M and N are rigid will be referred to as *rigid-rigid* pairs, and those in which both M and N are flexible will be referred to as *flexible-flexible* pairs.

A *simplified disagreement set* is a disagreement set that contains only flexible-rigid pairs and flexible-flexible pairs and contains at least one flexible-rigid pair.

The psubstitution σ is said to be a unifier of the disagreement set $\mathcal{D}$ if for all pairs $\langle M, N\rangle \in \mathcal{D}$, $M\sigma = N\sigma$. We let $\mathcal{U}(\mathcal{D})$ denote the set of unifiers of the disagreement set $\mathcal{D}$. □

DEFINITION 3.2 (MATCHING TREES)

Matching trees are trees whose nodes and arcs are labelled as follows:

- Nodes are labelled either with a pair in which the first component is either a simplified disagreement set or **succeed** or **fail** and the second component of which is a pcontext;
- Arcs are labelled with psubstitutions as follows: if $\mathcal{N}$ and $\mathcal{N}'$ are nodes and if there is an arc from $\mathcal{N}$ to $\mathcal{N}'$ then the arc is labelled by a psubstitution $\sigma : \Gamma' \longrightarrow \Gamma$ where Γ is the pcontext that labels $\mathcal{N}$ and Γ' is the pcontext that labels $\mathcal{N}'$.

Nodes labelled with either **succeed** or **fail** are called *terminal* and those labelled with simplified disagreement sets are call *non-terminal.* □

DEFINITION 3.3 (CURRENT PCONTEXT)

Suppose that a node $\mathcal{N}$ of a matching tree $\mathcal{M}$ is labelled by a pair in which the second component is the pcontext Γ. We say that Γ is the *current pcontext* for the node $\mathcal{N}$. □

3.2 Construction of a Matching Tree

The construction of a matching tree $\mathcal{M}$ for two terms $M:A$ and $N:B$ (which are well-typed in some context Δ) is based on the three functions described below:

- *reduce*, which takes as arguments two ptypes (initially A and B) and returns either a disagreement set or **fail**;
- *simplify*, which takes as argument a disagreement set, and returns a *simplified disagreement set* or **succeed** or **fail**;
- *match*, which takes as arguments a flexible pobject M, a rigid pobject N of similar ptype, a pcontext Δ and a disagreement set $\mathcal{D}_{\mathcal{N}}$; it returns a finite set $\mathcal{S}$ of psubstitutions, a pcontext and a disagreement set.

The initial node in $\mathcal{M}$ is $simplify(reduce(A,B) \cup \{\langle M,N\rangle\})$. Let $\mathcal{N}$ be the current node in $\mathcal{M}$. If $\mathcal{N}$ is labelled with (a pair in which the first component is) **succeed** or **fail** then $\mathcal{N}$ has no successor; otherwise, choose a pair $\langle M',N'\rangle$ in the set which labels $\mathcal{N}$ such that N' is rigid, and let the set of psubstitutions $\mathcal{S} = match(M',N',\Delta,\mathcal{D}_{\mathcal{N}})$, with some abuse of notation. If $\mathcal{S}=\emptyset$, replace the label of $\mathcal{N}$ by **fail**; otherwise, for every $\sigma \in \mathcal{S}$, grow an arc, labelled with σ, from $\mathcal{N}$ to the new node $\mathcal{N}'$ labelled with

$$\mathcal{D}_{\mathcal{N}'} = simplify(\{\langle M'\sigma, N'\sigma\rangle | \langle M',N'\rangle \in \mathcal{D}_{\mathcal{N}}\} \cup reduce(H_1,H_2)),$$

where $\mathcal{D}_{\mathcal{N}}$ denotes the simplified disagreement set which labels the node $\mathcal{N}$ and where H_1 and H_2 are certain ptypes generated by *match*.

3.3 The Procedure *reduce*

The procedure *reduce* implements[14] the function *reduce* described in Section 5.3.2 which is used to reduce the problem of unifying two $\lambda\Pi$-calculus ptypes (in head normal form) to that of unifying a finite set of pairs of $\lambda\Pi$-calculus pobjects (where in each pair the pobjects have similar ptype). Thus, given a pair of ptypes A and B, *reduce* returns a disagreement set $\mathcal{D}$.

Let the head normal forms of $M:A$ and $N:B$ be given by

$$\lambda x_1{:}A_1 \ldots \lambda x_{n_1}{:}A_{n_1}.@_1 M_1 \ldots M_{p_1} : \Pi x_1{:}A_1 \ldots \Pi x_{n_1}{:}A_{n_1}.c_1 M'_1 \ldots M'_{r_1}$$

and

$$\lambda y_1{:}B_1 \ldots \lambda y_{n_2}{:}B_{n_2}.@_2 N_1 \ldots N_{p_2} : \Pi y_1{:}B_1 \ldots \Pi y_{n_2}{:}B_{n_2}.c_2 N'_1 \ldots N'_{r_2}$$

respectively, where the kinds of c_1 and c_2 are given by

$$\Pi w_1{:}D_1 \ldots \Pi w_{r_1}{:}D_{r_1}.\mathrm{Type} \qquad \text{and} \qquad \Pi z_1{:}E_1 \ldots \Pi z_{r_2}{:}E_{r_2}.\mathrm{Type}$$

respectively.

[14] As an algorithm.

The procedure *reduce* first checks that the ptypes are similar, that is that $n_1 = n_2$ and $c_1 = c_2$ (and so $r_1 = r_2$, since $c_1 M'_1 \ldots M'_{r_1}$ and $c_2 N'_1 \ldots N'_{r_2}$ must both be of kind Type). If they are not then there is no hope of unification, *e.g.*,

$$M : \Pi x{:}C \,.\, \Pi y{:}C \,.\, cz \quad \text{and} \quad N : \Pi x{:}C \,.\, dz$$

are not unifiable. It then proceeds, recursively, to reduce each pair of ptypes arising from the headings of A and B, taking care to maintain all variable bindings; it then adds to the (putative) disagreement set, $\mathcal{D}$, each pair of pobjects arising from the matrices of A and B, again taking care to maintain all variable bindings, *cf.* the example in Section 1.

---

reduce(A, B)

Step 1

If A and B are not similar ($A \not\sim B$) then return **fail**; otherwise let $c = c_1 (= c_2)$, $r = r_1 (= r_2)$ and $n = n_1 (= n_2)$.

Step 2

If $n = 0$ then perform Step 3; otherwise, for $i = 1$ to n make the assignment:

$$\mathcal{D} := \mathcal{D} \cup (\lambda x_1{:}A_1 \ldots \lambda x_{i-1}{:}A_{i-1}, \lambda y_1{:}B_1 \ldots \lambda y_{i-1}{:}B_{i-1})(\textit{reduce}(A_i, B_i)),$$

where $(\lambda x_1{:}A_1 \ldots \lambda x_n{:}A_n, \lambda y_1{:}B_1 \ldots \lambda y_n{:}B_n)(\textit{reduce}(A, B))$ denotes that all of the disagreement pairs added to $\mathcal{D}$ as a result of the application of *reduce* to A and B must be λ-abstracted with respect to $x_1, \ldots, x_n$ in the first argument, and with respect to $y_1, \ldots, y_n$ in the second argument.

Step 3

For $j = 1$ to r, make the assignment:

$$\mathcal{D} := \mathcal{D} \cup \{(\lambda x_1{:}A_1 \ldots \lambda x_n{:}A_n . M'_j, \lambda y_1{:}B_1 \ldots \lambda y_n{:}B_n . N'_j)\}.$$

Step 4

Return $\mathcal{D}$. □

For example, given the pair of objects and (similar) types

$$M : \Pi x{:}A \,.\, cP \quad \text{and} \quad N : \Pi y{:}B \,.\, cQ\,,$$

where c is a constant, *reduce* returns the set

$$\{(\lambda x{:}A \,.\, P, \lambda y{:}B \,.\, Q)\} \cup \textit{reduce}(A, B)\,.$$

PROPOSITION 3.4 (TERMINATION OF REDUCE)

The procedure *reduce* terminates, and returns a disagreement set, $\mathcal{D}$.

PROOF

A simple inductive argument. □

PROPOSITION 3.5 (FAITHFULNESS OF REDUCE)

If $\sigma : \Gamma \longrightarrow \Delta$ unifies $M:A$ and $N:B$ and if $\mathcal{D} = \mathit{reduce}(A,B) \cup \{\langle M,N\rangle\}$, then for all $\langle M',N'\rangle \in \mathcal{D}$, $M'\sigma = N'\sigma$.
PROOF

The proof is by an analysis of the structure of the set $\mathcal{D}$. If σ unifies $M:A$ and $N:B$ then we must have that $A\sigma = B\sigma$, so that we must have, for $1 \leq i \leq n$, that

$$A_i\sigma = B_i\sigma$$

and, for $1 \leq j \leq r$, that

$$\lambda x_1{:}(A_1\sigma)\ldots\lambda x_n{:}(A_n\sigma).(M'_j\sigma) = \lambda y_1{:}(B_1\sigma)\ldots\lambda y_n{:}(B_n\sigma).(N'_j\sigma)\,.$$

Now, each pair (A_i,B_i), on which reduce is called, has the same general form as the pair (A,B), so that if A_i and B_i are unified by σ, then each of the pairs of ptypes arising from the headings of A_i and B_i must be unified by σ, *et seq.*. This decomposition, along with the addition of the pairs of ptypes arising from the matrices of A and B, is exactly that performed by *reduce*, so that the constructed disagreement set, $\mathcal{D}$, contains exactly those pairs which must be unifed by σ if the types A and B are unified by σ. Therefore $M'\sigma = N'\sigma$ for all $\langle M',N'\rangle \in \mathcal{D}$. □

Before proving the adequacy of *reduce*, we introduce a function r that assigns to each pair of pobjects in $\mathit{reduce}(A,B)$ (provided this is not **fail**) a unique level, as follows:

DEFINITION 3.6 (THE FUNCTION r)

Suppose that A and B are ptypes, with their head normal forms given respectively by $A \equiv \Pi x_1 : A_1 \ldots \Pi x_m : A_m \,.\, cM_1 \ldots M_n$ and $B \equiv \Pi y_1{:}B_1 \ldots \Pi y_m{:}B_m \,.\, cN_1 \ldots N_n$. Suppose further that $A_1 \equiv \Pi x_{11}{:}A_{11} \ldots \Pi x_{1m_1}{:}A_{1m_1} \,.\, cM_{11} \ldots M_{1n_1}$ and that $B_1 \equiv \Pi y_{11}{:}B_{11} \ldots \Pi y_{1m_1}{:}B_{1m_1} \,.\, cN_{11} \ldots N_{1n_1}$; and that, similarly,

$$A_{ij} \equiv \Pi x_{ij1}{:}A_{ij1} \ldots \Pi x_{ijm_{ij}}{:}A_{ijm_{ij}} \,.\, cM_{ij1} \ldots M_{ijn_{ij}}$$

and

$$B_{ij} \equiv \Pi y_{ij1}{:}B_{ij1} \ldots \Pi y_{ijm_{ij}}{:}B_{ijm_{ij}} \,.\, cN_{ij1} \ldots N_{ijn_{ij}}\,,$$

etc.. The function r assigns the level $0k$ to each of the pairs $\langle M_k,N_k\rangle$ $(1 \leq k \leq n)$; r assigns the level $1k$ to each of the pairs $\langle M_{1k},N_{1k}\rangle$ $(1 \leq k \leq n_1)$; r assigns the level ijk to each of the pairs $\langle M_{ijk},N_{ijk}\rangle$ $(1 \leq k \leq n_1)$, *etc.*. □

PROPOSITION 3.7 (ADEQUACY OF REDUCE)

Suppose that N proves $\Delta \vdash_\Sigma M:A$ and N proves $\Delta \vdash_\Sigma N:B$.

1. Let $\sigma : \Gamma \longrightarrow \Delta$ be a substitution and let $\mathcal{D} = \mathit{reduce}(A,B) \cup \{\langle M,N\rangle\}$. If for all $\langle M',N'\rangle \in \mathcal{D}$, $M'\sigma = N'\sigma$, and if $M\sigma = N\sigma$, then σ unifies $M:A$ and $N:B$.

2. If *reduce* returns **fail**, then $\mathcal{U}_\Delta(M:A, N:B) = \emptyset$.

PROOF

1. We observe that the levels, lists of integers, assigned to the elements of $\mathcal{D}$ by the function r can be ordered ($\sqsupset_r$) as follows, for suitable integers m, n_m, n_{m-1}, $\ldots$, n_2, n[15]:

$$\underbrace{11\ldots111}_{\text{length } m} \sqsupset_r \underbrace{11\ldots112}_{\text{length } m} \sqsupset_r \ldots \sqsupset_r \underbrace{11\ldots11n_m}_{\text{length } m} \sqsupset_r$$

$$\underbrace{11\ldots11}_{\text{length } m-1} \sqsupset_r \underbrace{11\ldots12}_{\text{length } m-1} \sqsupset_r \ldots \sqsupset_r \underbrace{11\ldots1n_{m-1}}_{\text{length } m-1} \sqsupset_r \ldots$$

$$\underbrace{11}_{\text{length } 2} \sqsupset_r 12 \sqsupset_r \ldots \sqsupset_r 1(n_2-1) \sqsupset_r 1n_2 \sqsupset_r \underbrace{01}_{\text{length } 2} \sqsupset_r 02 \sqsupset_r \ldots \sqsupset_r 0(n-1) \sqsupset_r 0n \sqsupset_r 0\,,$$

with the pair $\langle M, N\rangle$ assigned the level 0.

For example, suppose we have that $A \equiv \Pi x_1 : A_1 \,.\, cM_1$ and $B \equiv \Pi y_1 : B_1 \,.\, cN_1$, where $A_1 \equiv \Pi x_{11} : A_{11} \,.\, dM_{11}$ and $B_1 \equiv \Pi y_{11} : B_{11} \,.\, dN_{11}$, $A_{11} \equiv eM_{111}$ and $B_{11} \equiv eN_{111}$, then the ordering can be considered to be

$$\langle M_{111}, N_{111}\rangle \sqsupset_r \langle M_{11}, N_{11}\rangle \sqsupset_r \langle M_1, N_1\rangle\,.$$

We can consider this order to be a finite segment of the natural numbers with their usual ordering, the highest number being labelled by $11\ldots111$ and the lowest being labelled by 0.

The result now follows by a simple inductive argument on the structure of A and B.

If the pairs at all levels down to and including level $j+1$ have been unified, then the components of the pair at level j have unified ptypes, but this pair is also unified, *etc.*.

2. The procedure *reduce*(A, B), for some pair (A, B) occurring in the pair of ptypes to be reduced, returns **fail** just in case A and B are not similar, so that either $n_1 \neq n_2$ or $c_1 \neq c_2$, and so by Lemma 2.22 $M:A$ and $N:B$ are not unifiable. □

Henceforth we shall consider $\sqsupset_r$ to be an order on disagreement pairs and drop the subscript r, so that we write $\langle M', N'\rangle \sqsupset \langle M'', N''\rangle$, *etc.*. Also, we shall write $\langle M', N'\rangle \sqsupset \langle M'', N''\rangle$ for $\langle M', N'\rangle \sqsupset \ldots \sqsupset \langle M'', N''\rangle$, *etc.*.

3.4 The Procedure *simplify*

The purpose of the procedure *simplify* is to remove from the input disagreement set any rigid-rigid pairs, and to check for nodes containing only flexible-flexible pairs. Let $\mathcal{D}$ be the disagreement set input to *simplify*.

[15] We refrain from providing the full, formal definition of the ordering $\sqsupset_r$. Its formal definition is both tedious and unnecessary for the remainder of the paper. We believe that the sketch below provides an adequate illustration of the definition, from which the interested reader might reconstruct the formal definition if he so desires.

simplify($\mathcal{D}$)

Step 1

If there does not exist a pair $\langle M, N\rangle \in \mathcal{D}$ such that M and N are both rigid then perform Step 2; otherwise write

$$\mathcal{D} = \{\langle M, N\rangle\} \uplus \mathcal{D}'$$

where

$$M = \lambda x_1 : A_1 \ldots \lambda x_{n_1} : A_{n_1} . @_1 M_1 \ldots M_{p_1}$$

and

$$N = \lambda y_1 : B_1 \ldots \lambda y_{n_2} : B_{n_2} . @_2 N_1 \ldots N_{p_2}.$$

We now check that the bare heading of M is equal (up to α-equality of the λ-terms) to the bare heading of N[16]:

if $@_1 \neq @_2$, then return **fail**, otherwise let $p = p_1 (= p_2)$, (note that M and N have similar types so that if $@_1 = @_2$ then we must have that $n_1 = n_2$ $(= n)$ and $p_1 = p_2$ $(= p)$), and make the assignment:

$$\mathcal{D} := \{\langle \tilde{M}_i, \tilde{N}_i\rangle | 1 \leq i \leq p\} \uplus \mathcal{D}'$$

where $\tilde{M}_i = \lambda x_1 : A_1 \ldots \lambda x_n : A_n . M_i$ and $\tilde{N}_i = \lambda y_1 : B_1 \ldots \lambda y_n : B_n . N_i$. Extend the order $\sqsupset$ to $\mathcal{D}$ by making the assignment

$$\sqsupset := \sqsupset \cup \{\langle \tilde{M}_i, \tilde{N}_i\rangle \sqsupset \langle \tilde{M}_j, \tilde{N}_j\rangle \mid 1 \leq i < j \leq p\}.$$

Repeat Step 1.

Step 2

Replace in $\mathcal{D}$ each pair $\langle M, N\rangle$ where M is rigid and N is flexible by the pair $\langle N, M\rangle$.

Step 3 If there exists $\langle M, N\rangle \in \mathcal{D}$ such that N is rigid, then return the set $\mathcal{D}$; otherwise return **succeed**. □

DEFINITION 3.8 (THE MEASURE δ)

Let $\mathcal{D} = \{\langle M_i, N_i\rangle | 1 \leq i \leq n\}$, and define the measure $\delta(\mathcal{D}) \in \mathbb{N}$ by

$$\delta(\mathcal{D}) = n + \sum_{i=1}^{n} \pi(M_i)$$

where π is the measure given in Definition 2.19. □

Let $\mathcal{D}$ be a disagreement set that is input to *simplify*. We show that *simplify* terminates after a finite number of steps, and returns a simplified disagreement set possessing the properties described by the three results given below.

[16] If the bare headings are not so equal, there is no hope of unification.

PROPOSITION 3.9 (CORRECTNESS OF AND TERMINATION OF SIMPLIFY)

Let $\mathcal{D}$ be a disagreement set.

1. If $simplify(\mathcal{D}) =$ **fail**, then $\mathcal{U}(\mathcal{D}) = \emptyset$.
2. If $simplify(\mathcal{D}) = \mathcal{D}'$, where $\mathcal{D}'$ is a disagreement set, then $\mathcal{U}(\mathcal{D}) = \mathcal{U}(\mathcal{D}')$.

Furthermore, the procedure terminates after a finite number of applications of Step 1.

PROOF

The proof of Part 1 is essentially the same as that of the corresponding result in [14] (Lemma 3.4). Let $\mathcal{D}_k$, for some $k \geq 0$, be the set obtained after k successful applications of Step 1. We assume that $\mathcal{U}(\mathcal{D}_k) = \mathcal{U}(\mathcal{D})$, which is immediately true for $k = 0$. If there exists a rigid-rigid pair in $\mathcal{D}_k$, let $\langle M, N \rangle$ be the pair selected. If M and N have distinct heads then the algorithm stops and returns the answer **fail**; but if M and N have distinct bare headings, we know, by Lemma 2.22, that $\mathcal{U}(\mathcal{D}_k) = \emptyset$[17], and therefore that $\mathcal{U}(\mathcal{D}) = \emptyset$, which proves Part 1.

The proof of Part 2 also is essentially the same as that of the corresponding result in [14] (Lemma 3.6). Suppose that M and N do not have distinct headings, and that $\sigma : \Gamma \longrightarrow \Delta$ is a unifying psubstitution. Assume that for all $v \in \mathrm{FV}(M) \cup \mathrm{FV}(N)$ and for all $1 \leq i \leq n$, $x_i \notin \mathrm{FV}(v\sigma)$ and $y_i \notin \mathrm{FV}(v\sigma)$, (otherwise we can simply rename the $x(y)_i$s by α-conversion), so that we have

$$M\sigma = \lambda x_1{:}(A_1\sigma) \ldots \lambda x_n{:}(A_n\sigma).@(M_1)\sigma \ldots (M_p\sigma)$$

and

$$N\sigma = \lambda y_1{:}(B_1\sigma) \ldots \lambda y_n{:}(B_n\sigma).@(N_1\sigma) \ldots (N_p\sigma).$$

Now, $M\sigma = N\sigma$ if and only if for all $1 \leq i \leq p$,

$$\lambda x_1{:}(A_1\sigma) \ldots \lambda x_n{:}(A_n\sigma).(M_i\sigma) = \lambda y_1{:}(B_1\sigma) \ldots \lambda y_n{:}(B_n\sigma).(N_i\sigma),$$

that is, if and only if $\tilde{M}_i\sigma = \tilde{N}_i\sigma$, for each i. It is immediate that $\mathcal{D}_{k+1}$ differs from $\mathcal{D}_k$ only in that the pair $\langle M, N \rangle$ is replaced by the set of pairs $\{\langle \tilde{M}_i, \tilde{N}_i \rangle \mid 1 \leq i \leq p\}$. Therefore $\mathcal{U}(\mathcal{D}_{k+1}) = \mathcal{U}(\mathcal{D}_k) = \mathcal{U}(\mathcal{D})$. We note further that the procedure terminates after a finite number of applications of Step 1, since

$$\delta(\mathcal{D}_{k+1}) = \delta(\mathcal{D}_k) - 1.$$

If **fail** has not been returned, we proceed to Step 2 with a set $\mathcal{D}_k$ such that $\mathcal{U}(\mathcal{D}_k) = \mathcal{U}(\mathcal{D})$, and in Step 2 the only operation we perform is to swap the order of some of the pairs in $\mathcal{D}_k$, which has no effect on unification. This proves Part 2. □

We must now prove that disagreement sets in which all pairs are flexible-flexible are always unifiable. This is significantly more difficult than the corresponding result in [14] (Lemma 3.5), and indeed we must delay the proof of this fact until we have proved the correctness of *match*. However, we begin by proving the result for pairs in which the components are of common type, *cf.* [14] (Lemma 3.5, Part 2).

[17] The bare headings can be distinct only if either the heads are distinct or the numbers of abstractions differ. A similar argument is used in [9].

LEMMA 3.10 (FLEXIBLE-FLEXIBLE UNIFIABILITY FOR COMMON TYPES)

If $\mathcal{D}$ is (a subset of) a disagreement set in which each pair is flexible-flexible and such that for each pair the components of that pair are of common type then $\mathcal{U}(\mathcal{D}) \neq \emptyset$.

PROOF

We proceed in a manner similar to the elementary construction given in the proof of Lemma 3.5 of [14].

Suppose the that pair $\langle Q, R\rangle$ is such that the head normal forms Q and R have the same ptype (up to α-conversion) and that they are given by

$$Q = \lambda u_1 : A_1 \ldots \lambda u_{n_1} : A_{n_1} . f Q_1 \ldots Q_{r_1}$$

and

$$R = \lambda v_1 : A_1 \ldots \lambda v_{n_2} : A_{n_2} . g R_1 \ldots R_{r_2} ,$$

where, with the A'_is and A''_is denoting the ptypes arising from the ptypes of f and g,

$$\begin{aligned} \tau(f) = F \;\; = \;\; & \Pi u'_1 : A'_1 \ldots \Pi u'_{r_1} : A'_{r_1} \\ & \Pi u'_{r_1+1} : A'_{r_1+1} \ldots \Pi u'_{s_1} : A'_{s_1} . c P_1 \ldots P_r \end{aligned}$$

and

$$\begin{aligned} \tau(g) = G \;\; = \;\; & \Pi v'_1 : A''_1 \ldots \Pi v'_{r_2} : A''_{r_2} \\ & \Pi v'_{r_2+1} : A''_{r_2+1} \ldots \Pi v'_{s_2} : A''_{s_2} . c P_1 \ldots P_r . \end{aligned}$$

Suppose that

$$\Delta \equiv \Delta_1, f : F, \Delta_2, g : G, \Delta_3$$

is the current pcontext. We assume, without loss of generality, that f precedes g in Δ — of course, there is a trivial case in which they are equal.

Define $\zeta_\Delta(Q, R)$ to be the psubstitution

$$\zeta_\Delta(Q, R) \equiv_{\text{def}} \langle \ldots, \langle M_f, f\rangle, \ldots, \langle M_g, g\rangle, \ldots \rangle ,$$

where M_f is given by

$$M_f \equiv_{\text{def}} \lambda w_1 : A'_1 \ldots \lambda w_{s_1} : A'_{s_1} [w_1 / u'_1, \ldots, w_{s_1-1} / u'_{s_1-1}] . w_f ,$$

for w_f a distinct canonical variable of appropriate ptype (we assume that for a given ptype, just one such choice of variable is available), and

$$M_g \equiv_{\text{def}} \lambda w'_1 : A''_1 \ldots \lambda w'_{s_2} : A''_{s_2} [w'_1 / v'_1, \ldots, w'_{s_2-1} / v'_{s_2-1}] . w_g ,$$

for w_g a distinct canonical variable of appropriate ptype.

$\zeta_\Delta(Q, R)$ unifies Q and R: to see this, consider the application of $\zeta_\Delta(Q, R)$ to Q and R:

$$Q\zeta_\Delta(Q, R) = \lambda u_1 : A_1 \ldots \lambda u_{n_1} : A_{n_1} . \lambda w_{r_1+1} : A'_{r_1+1} \ldots \lambda w_{s_1} : A'_{s_1} . w_f$$

and

$$R\zeta_\Delta(Q,R) = \lambda v_1{:}A_1 \ldots \lambda v_{n_2}{:}A_{n_2}.\lambda w'_{r_2+1}{:}A''_{r_2+1} \ldots \lambda w'_{s_2}{:}A''_{s_2}\,.\,w_g\,.$$

Here we assume that the u'_is and v'_is do not clash with w_f and w_g. Since Q and R have the same ptype, we must have that $n_1 + s_1 - r_1 = n_2 + s_2 - r_2$, and $\tau(w_f) = \tau(w_g)$, so that we have $w_f = w_g (= w$, say). Therefore $Q\zeta_\Delta(Q,R) = R\zeta_\Delta(Q,R)$.

It remains to note that if the domain pcontext Δ of the psubstitution $\zeta_\Delta(Q,R)$ is given by

$$\Delta_1, f{:}F, \Delta_2, g{:}G, \Delta_3\,,$$

then the range pcontext of the psubstitution is given by

$$\Delta_1, w{:}cP_1 \ldots P_r, \Delta_2[M_f/f], \Delta_3[M_f/f, M_g/g]\,.$$

It is a straightforward matter to verify that $\zeta_\Delta(Q,R)$ is well-defined.

In order to unify a set of *flexible-flexible* pairs, such that for each pair the components of the pair are of common type, we simply compose psubstitutions of the above form. □

3.5 The Procedure *match*

Following [14], the procedure *match* is presented in an informal algorithmic style. The procedure *match* is not very different from the procedure *MATCH* of [14]. However, we must take care to maintain the current pcontext — as *match* introduces new variables — and the disagreement set — as match introduces new pairs of types that we must unify. Such new pairs arise because *match* does not construct an object whose type is the same as the variable it is to replace, as in the procedure *MATCH* of [14], but yields an object whose type is merely *similar* to that of the variable it is to replace. Our procedure *reduce* guarantees only similarity of types, *not* equality. We must take care to extend the order $\sqsupset$ to these new pairs. This ordering allows us to demonstrate that flexible-flexible pairs are always unifiable.

---

match$(M, N, \Delta, \mathcal{D}_\mathcal{N})$
Let $\mathcal{D}_\mathcal{N}$ be the reduced disagreement set labelling the current node $\mathcal{N}$ in the matching tree $\mathcal{M}$, and let the current pcontext be Δ. Let M be a flexible pobject and N be a rigid pobject such that $\langle M, N\rangle \in \mathcal{D}$ (and so $\tau(M) \sim \tau(N)$). We take the given pobjects M and N to have the following head normal forms:

$$M \equiv \lambda x_1{:}A_1 \ldots \lambda x_m{:}A_m.fM_1 \ldots M_p$$

$$N \equiv \lambda y_1{:}B_1 \ldots \lambda y_n{:}B_n.@N_1 \ldots N_q\,,$$

where m, n, p, $q \geq 0$, and where the ptypes of f and @ are given by:

$$f : \Pi u_1{:}E_1 \ldots \Pi u_p{:}E_p.\Pi u_{p+1}{:}E_{p+1} \ldots \Pi u_r{:}E_r\,.\,cP_1 \ldots P_l$$

and

$$@ : \Pi w_1 : F_1 \ldots \Pi w_q : F_q . \Pi w_{q+1} : F_{q+1} \ldots \Pi w_s : F_s . cQ_1 \ldots Q_l ,$$

where c is a constant and where $cP_1 \ldots P_l$ and $cQ_1 \ldots Q_l$ are not of higher kind.

The purpose of the procedure *match* is to adjust the heading of the flexible pobject M, and to this end we shall consider psubstitutions for f of a pobject with a heading of the form $\lambda z_1 : C_1 \ldots \lambda z_t : C_t . @'$, for some t. The head $@'$ will be either $@$, in which case the most general such psubstitution will be given by the rule of *imitation*, (Case 1, below), or z_k for some $1 \leq k \leq t$, in which case the most general such psubstitution will be given by the rule of *projection*, (Case 2, below). In *imitation*, we attempt to substitute for f a pobject with the same head as N. In *projection*, we attempt to project f on to one of its arguments.

Since the number of abstractions in a pobject cannot be reduced by psubstitution, and since the number of abstractions in the heading of rigid pobjects is unchanged under psubstitution, we must require that $m \leq n$. So if $m > n$ return $\mathcal{S} = \emptyset$; otherwise $n - m \geq 0$.[18]

Case 1 : imitation

The *imitation* construction applies where either $@$ is a constant, or where $@ = y_i$ for some $m < i \leq n$. Following [14], we distinguish two cases.

1. $n - m > 0$. In this case the pobject which we substitute for f is given uniquely by:

$$\begin{array}{rcl} M_f & \equiv_{\text{def}} & \lambda v_1 : E_1 \ldots \lambda v_p : E_p[v_1/u_1, \ldots, v_{p-1}/u_{p-1}] \\ & & \lambda y_{m+1} : B_{m+1} \ldots \lambda y_n : B_n . @ N_1' \ldots N_q' \end{array}$$

where for $1 \leq i \leq q$, $N_i' = h_i v_1 \ldots v_p y_{m+1} \ldots y_n$ with each h_i and v_i distinct and such that each h_i and each v_i is not in $\mathrm{Dom}(\Delta)$.

For each h_i, $1 \leq i \leq q$, we require that

$$\begin{array}{rcl} \tau(h_i) & = & \Pi v_1 : E_1 \ldots \Pi v_p : E_p[v_1/u_1, \ldots, v_{p-1}/u_{p-1}] \\ & & \Pi y_{m+1} : B_{m+1} \ldots \Pi y_n : B_n . \tau(N_i) . \end{array}$$

We must maintain the condition that $H_1 = \tau(M_f)$ and $H_2 = \tau(f)$ are to be unified. We have

$$\begin{array}{rcl} H_1 & = & \Pi v_1 : E_1 \ldots \Pi v_p : E_p[v_1/u_1, \ldots, v_{p-1}/u_{p-1}] \, \Pi y_{m+1} : B_{m+1} \ldots \Pi y_n : B_n \\ & & \Pi w_{q+1} : F_{q+1} \ldots \Pi w_s : F_s . \\ & & cQ_1 \ldots Q_l \, [N_1'/w_1, \ldots, N_q'/w_q] \end{array}$$

and

$$\begin{array}{rcl} H_2 & = & \Pi v_1 : E_1 \ldots \Pi v_k : E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \\ & & \Pi u_{k+1} : E_{k+1}[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \ldots u_r : E_r[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] . \\ & & (cP_1 \ldots P_l)[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] . \end{array}$$

[18] The applicability of this argument rests on the *similarity* of the types of M and N. In [14] it is the case that their types are equal, which is a stronger condition.

We make the assignment

$$\mathcal{D}_{\mathcal{N}'} := \mathcal{D}'_{\mathcal{N}} \cup \mathit{reduce}(H_1, H_2)\,,$$

where $\mathcal{D}'_{\mathcal{N}}$ is obtained by applying the returned psubstitution

$$\langle \ldots \langle M_f, f\rangle \ldots\rangle : \Gamma \to \Delta\,,$$

to the input disagreement set $\mathcal{D}_{\mathcal{N}}$, and where the range pcontext Γ is constructed as described below. Finally, we extend the order $\sqsupset$ to the disagreement set $\mathcal{D}_{\mathcal{N}'}$ by making the assignment $\sqsupset := \sqsupset' \cup \sqsupset_{r'}$, where the order $\sqsupset_{r'}$ is defined on $\mathit{reduce}(H_1, H_2)$ just as in Definition 3.6 and Proposition 3.7 and where the order $\sqsupset'$ is just the order $\sqsupset$ on $\mathcal{D}_{\mathcal{N}}$ under the returned psubstitution.

2. $n - m = 0$. In this case it must be that @ is a constant. The pobject M_f substituted for f is of the form:

$$M_f \equiv_{\text{def}} \lambda v_1 : E_1 \ldots \lambda v_k : E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}].@N'_1 \ldots N'_{q-p+k}\,,$$

provided that $\max(0, p-q) \leq k \leq p$, and where $N_i = h_i v_1 \ldots v_k$, for $1 \leq i \leq q-(p-k)$, for suitable h_is. To ensure type-compatibility we necessarily require that

$$\tau(h_i) = \Pi v_1 : E_1 \ldots \Pi v_k : E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}]\,.\,\tau(N_i)\,.$$

We must maintain the condition that

$$H_1 = \tau(\lambda v_1 : E_1 \ldots \lambda v_k : E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}].@N'_1 \ldots N'_{q-p+k})$$

and $H_2 = \tau(f)$ are to be unified. We have

$$\begin{aligned} H_1 \;=\; & \Pi v_1 : E_1 \ldots \Pi v_k : E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}]\, \Pi w_{q-p+k+1} : F_{q-p+k+1} \cdots \\ & \Pi w_s : F_s\,.\,cQ_1 \ldots Q_l\,[N'_1/w_1, \ldots, N'_{q-p+k}/w_{q-p+k}] \end{aligned}$$

and

$$\begin{aligned} H_2 \;=\; & \Pi v_1 : E_1 \ldots \Pi v_k : E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \\ & \Pi u_{k+1} : E_{k+1}[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \ldots u_r : E_r[v_1/u_1, \ldots, v_{k-1}/u_{k-1}]\,. \\ & (cP_1 \ldots P_l)[v_1/u_1, \ldots, v_{k-1}/u_{k-1}]\,. \end{aligned}$$

We verify that $k + s - (q-p+k) = r$ (from $m = n$ and the similarity of the types of M and N we have that $s - q = r - p$), and make the assignment

$$\mathcal{D}_{\mathcal{N}'} := \mathcal{D}'_{\mathcal{N}} \cup \mathit{reduce}(H_1, H_2)\,,$$

where $\mathcal{D}'_{\mathcal{N}}$ is obtained by applying the returned psubstitution

$$\langle \ldots \langle M_f, f\rangle \ldots\rangle : \Gamma \to \Delta\,,$$

to the input disagreement set $\mathcal{D}_{\mathcal{N}}$, and where the range pcontext Γ is constructed as described below. Finally, we extend the order $\sqsupset$ to the disagreement set $\mathcal{D}_{\mathcal{N}'}$ by making the assignment $\sqsupset := \sqsupset' \cup \sqsupset_{r'}$, where the order $\sqsupset_{r'}$ is defined on $\mathit{reduce}(H_1, H_2)$ just as in Definition 3.6 and Proposition 3.7 and where the order $\sqsupset'$ is just the order $\sqsupset$ on $\mathcal{D}_{\mathcal{N}}$ under the returned psubstitution.

Case 2 : projection
Following [14], we distinguish three cases for the *heading* of the object that we may substitute for f. We take the union of their solutions.

1. $\lambda v_1{:}E_1 \ldots \lambda v_k{:}E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \,.\, v_i$, for $1 \leq k \leq p$ and $1 \leq i \leq k$.
   In this case the pobject that we substitute for f is given by:

$$M_f \equiv_{\text{def}} \lambda v_1{:}E_1 \ldots \lambda v_k{:}E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}].v_i N'_1 \ldots N'_{s'} \,,$$

   where for $1 \leq i \leq s'$, $N'_i = h_i v_1 \ldots v_k$, with each h_i and v_i distinct and such that each h_i and v_i is not in $\mathrm{Dom}(\Delta)$, and where $\tau(h_j)$ is determined as described below.
   We have that $E_i = \Pi z_1{:}G_1 \ldots \Pi z_{s''}{:}G_{s''} \,.\, dR_1 \ldots R_{l'}$, say. We must maintain the condition that $H_1 = \tau(M_f)$ and $H_2 = \tau(f)$ are to be unified.
   We have that

$$\begin{aligned} H_1 \;=\; & \Pi v_1{:}E_1 \ldots \Pi v_k{:}E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \\ & \Pi z_{s''-s'+1}{:}G_{s''-s'+1}[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \ldots \Pi z_{s''}{:}G_{s''}[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \,. \\ & (dR_1 \ldots R_{l'})[N'_1/z_1, \ldots, N'_{s'}/z_{s'}][v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \end{aligned}$$

   and

$$\begin{aligned} H_2 \;=\; & \Pi v_1{:}E_1 \ldots \Pi v_k{:}E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \\ & \Pi u_{k+1}{:}E_{k+1}[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \ldots u_r{:}E_r[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \,. \\ & (cP_1 \ldots P_l)[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \,. \end{aligned}$$

   For it to be possible for a solution to exist, we must require (for similarity) that

$$k + s'' - s' = r \quad \text{and} \quad d = c \;,$$

   so that we must have that $l = l'$. So in this case we take

$$\tau(h_i) = \Pi v_1{:}E_1 \ldots \Pi v_k{:}E_k[v_1/u_1, \ldots, v_{k-1}/u_{k-1}] \,.\, G_i \,,$$

   for $1 \leq i \leq s'$, and make the assignment

$$\mathcal{D}_{\mathcal{N}'} := \mathcal{D}'_{\mathcal{N}} \cup \mathit{reduce}(H_1, H_2) \,,$$

   where $\mathcal{D}'_{\mathcal{N}}$ is obtained by applying the returned psubstitution

$$\langle \ldots \langle M_f, f \rangle \ldots \rangle : \Gamma \to \Delta \,,$$

   to the input disagreement set $\mathcal{D}_{\mathcal{N}}$, and where the range pcontext Γ is constructed as described below. Finally, we extend the order $\sqsupset$ to the disagreement set $\mathcal{D}_{\mathcal{N}'}$ by making the assignment $\sqsupset := \sqsupset' \cup \sqsupset_{r'}$, where the order $\sqsupset_{r'}$ is defined on $\mathit{reduce}(H_1, H_2)$ just as in Definition 3.6 and Proposition 3.7 and where the order $\sqsupset'$ is just the order $\sqsupset$ on $\mathcal{D}_{\mathcal{N}}$ under the returned psubstitution.

2. $\lambda v_1 : E_1 \ldots \lambda v_p : E_p[v_1/u_1, \ldots, v_{p-1}/u_{p-1}].\lambda y_{m+1} : B_{m+1} \ldots \lambda y_{m+k} : B_{m+k} \cdot v_i$, for $1 \leq k \leq n-m$ and $1 \leq i \leq p$. Just as in [14], this case is almost identical to the previous one.

   In this case the pobject that we substitute for f is given by:

$$M_f \equiv_{\text{def}} \lambda v_1 : E_1 \ldots \lambda v_p : E_p[v_1/u_1, \ldots, v_{p-1}/u_{p-1}] \\ \lambda y_{m+1} : B_{m+1} \ldots \lambda y_{m+k} : B_{m+k} \cdot v_i N'_1 \ldots N'_{s'},$$

   where for $1 \leq i \leq s'$, $N'_i = h_i v_1 \ldots v_k$, with each h_i and v_i distinct and such that each h_i and v_i is not in $\text{Dom}(\Delta)$, and where $\tau(h_j)$ and s' are determined in a manner similar to that of the previous case. We extend the disagreement set $\mathcal{D}_{\mathcal{N}}$ and the order $\sqsupset$ in a manner similar to that of the previous case.

3. $\lambda v_1 : E_1 \ldots \lambda v_p : E_p[v_1/u_1, \ldots, v_{p-1}/u_{p-1}].\lambda y_{m+1} : B_{m+1} \ldots \lambda y_{m+k} : B_{m+k} \cdot y_{m+i}$, for $1 \leq k \leq n-m$ and $1 \leq i \leq k$.

   Following the substitution of the pobject for f, we are required to unify

$$\lambda x_1 : A_1 \ldots \lambda x_m : A_m \lambda y_{m+1} : B_{m+1} \ldots \lambda y_{m+k} : B_{m+k} . y_{m+i} : B_{m+i}(\ldots)$$

   with

$$\lambda y_1 : B_1 \ldots \lambda y_m : B_m . \lambda y_{m+1} : B_{m+1} \ldots \lambda y_n : B_n . @(\ldots)$$

   and this will be rejected by *simplify* unless $k = n - m$ and $@ = y_{m+i}$, and this case has already been considered in Part 1 of the imitation case .

This concludes the description of the *imitation* and *projection* cases of the procedure *match*. In the setting of simply-typed terms, [14] provides an analysis ofthe number of solutions calculated in the case of each of *imitation* and *projection*. Although the analysis of [14] does not apply directly to our setting, we conjecture that similar methods can be used for the $\lambda\Pi$-calculus.

It now only remains to construct the image pcontext of the psubstitution determined by the two cases given above.

In each of *imitation* and *projection* an object M_f is substituted for the variable f. Let $\mathcal{N}$ be the current selected node and let Δ be the current pcontext for $\mathcal{N}$, so that Δ has the form

$$\Delta \equiv \Delta_1, f : A, \Delta_2,$$

where A denotes the ptype of f.

Suppose that all of the free variables of the object M_f are contained in the pcontext Δ_1, Θ, where Θ contains just the new variables introduced by *match*, then the range pcontext of the psubstitution σ given by $\sigma \equiv \langle \ldots, \langle M_f, f \rangle, \ldots \rangle$ is

$$\Gamma \equiv \Delta_1, \Theta, \Delta_2[M_f/f] .$$

It follows immediately that the definition of a psubstitution is satisfied. □

We now proceed to prove the correctness of the procedure *match*. The proof is similar to that of the corresponding result of [14], although since our psubstitutions are (ordered) tuples there are some differences. Just as in [14], we introduce a complexity measure for psubstitutions. The measure is defined on the head normal forms of the pobjects constructed.

DEFINITION 3.11 (THE MEASURE θ)

Let the psubstitution $\xi : \Gamma \longrightarrow \Delta$ be given by

$$\xi \equiv \langle\langle M_1, y_1\rangle, \ldots, \langle M_i, y_i\rangle, \ldots, \langle M_r, y_r\rangle\rangle$$

then the measure $\theta(\xi) \in \mathbb{N}$ is given by

$$\theta(\xi) = r + \sum_{i=1}^{r} \pi(M_i),$$

where the measure π is given by Definition 2.19. □

PROPOSITION 3.12 (CORRECTNESS OF MATCH)

Suppose that the terms M and N form the pair $\langle M, N\rangle$ which occurs at the node $\mathcal{N}$ which has current pcontext Δ. Suppose further that M is flexible and that N is rigid. If there exists $\rho : \Gamma \longrightarrow \Delta \in \mathcal{U}(\{\langle M, N\rangle\})$, then $match(M, N, \Delta, \mathcal{D}_{\mathcal{N}})$ returns a non-empty set $\mathcal{S}$ in which there is a psubstitution σ such that $\rho \leq_\Delta \sigma$, and we can find a psubstitution η such that $\rho =_\Delta \eta \circ \sigma$ with $\theta(\eta) < \theta(\rho)$.
PROOF

The proof is very similar to that of the corresponding result in [14]. Let x_f be the head of M, and let @ be that of N. If ρ unifies M and N, then there must exist in (the tuple) ρ a pair pertaining to x_f. Suppose then that

$$x_f\rho = \lambda z_1 : B_1 \ldots \lambda z_k : B_k . @' N_1' \ldots N_l' (= M_{x_f}),$$

that is ρ is of the form

$$\langle \ldots, \langle M_{x_f}, x_f\rangle, \ldots\rangle .$$

As with the proof of the corresponding result in [14], there are two cases to consider.

1. $@'$ is @ or some z_i, $1 \leq i \leq k$. In *match* we considered all possible cases of such headings, so that there must exist in $\mathcal{S} = match(M, N, \Delta, \mathcal{D}_{\mathcal{N}})$ a psubstitution σ with a component

$$\langle \lambda w_1 : B_1 \ldots \lambda w_k : B_k . @'' N_1'' \ldots N_l'', x_f\rangle$$

where $@'' = @'[w_1/z_1, \ldots, w_k/z_k]$ and $N_j'' = h_j w_1 \ldots w_k$ for $1 \leq j \leq l$; such a $\sigma : \Theta \longrightarrow \Delta$ is given by

$$\langle\langle x_1, x_1\rangle, \ldots, \langle \lambda w_1 : B_1 \ldots \lambda w_k : B_k . @'' N_1'' \ldots N_l'', x_f\rangle, \ldots, \langle x_n, x_n\rangle\rangle,$$

where the pcontext Θ is constructed from the pcontext Δ by removing the component labelled by f and adding suitable components labelled by $h_1, \ldots, h_l$.

Let $\eta : \Gamma \longrightarrow \Theta$ be the psubstitution given by

$$\eta \equiv \langle\langle M_1, x_1\rangle, \ldots, \langle P_1, h_1\rangle, \ldots, \langle P_l, h_l\rangle, \ldots, \langle M_n, x_n\rangle\rangle,$$

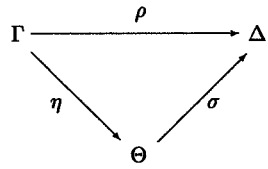
where $P_i = \lambda z_1{:}B_1 \ldots \lambda z_k{:}B_k.N_i'$, $1 \leq i \leq l$, and where we assume that ρ is given by

$$\langle\langle M_1, x_1\rangle, \ldots, \langle M_{x_f}, x_f\rangle, \ldots, \langle M_n, x_n\rangle\rangle\,.$$

We note that it is a straightforward calculation to show that σ and η are well-defined, that $\rho =_\Delta \eta \circ \sigma$, and that

$$\begin{aligned}\theta(\eta) &= l - 1 + \textstyle\sum_{j=1}^{f-1}\pi(M_j) + \sum_{j=f+1}^{l}\pi(M_j) + \\ &\quad\; l + \textstyle\sum_{i=1}^{l}\pi(P_i) \\ &= \theta(\rho) - 1 < \theta(\rho)\,.\end{aligned}$$

This construction may be elucidated by the following commuting diagram of psubstitutions:



Note that this diagram is of psubstitutions and so is not in the syntactic category $C(\Sigma)$ of Definition 2.9.

2. $@'$ is neither $@$ nor some z_i. In this case we must have that

$$M\rho = \lambda x_1{:}A_1 \ldots \lambda x_n{:}A_n.@'' \ldots\,,$$

where, for $1 \leq j \leq n$, $@''$ is distinct from $@$, x_j and z_j, so that $M\rho \neq N\rho$, and so this case can never arise. As in [14], this proves that the rules of *imitation* and *projection* are sufficient for the existence algorithm. □

We are now able to prove the unifiability of simplified disagreement sets in which all the pairs are flexible-flexible.

PROPOSITION 3.13 (FLEXIBLE-FLEXIBLE UNIFIABILITY)

Let $simplify(\mathcal{D})$ be a simplified disagreement set that labels some node $\mathcal{N}$ of a matching tree $\mathcal{M}$ for $M{:}A$ and $N{:}B$ (well-typed in some context). If $simplify(\mathcal{D})$ = **succeed**, then $\mathcal{U}(\mathcal{D}) \neq \emptyset$.

PROOF

We must show that if *simplify* returns **succeed** then the input disagreement set is always unifiable.

simplify returns **succeed** when all pairs in the disagreement set are flexible-flexible, so we must show that such a set is always unifiable. We cannot simply apply the elementary construction of [14] to flexible-flexible pairs since the components any given pair may have types which are only similar, not equal. However the ordering $\sqsupset$ on $\mathcal{D}$ allows us to construct a suitable substitution.

The first step is to note that we extended the order $\sqsupset$ to all of the elements of $\mathcal{D}$ by extending its definition each time we added new disagreement pairs to $\mathcal{D}$, in the procedures *simplify* and *match*, so that $\sqsupset$ was preserved on the existing, although modified, pairs.

We now observe that the following property of $\sqsupset$ is preserved by the procedures *simplify* and *match*: if the pair $\langle M', N' \rangle$ is such that each pair $\langle M'', N'' \rangle$ such that $\langle M'', N'' \rangle \sqsupset \langle M', N' \rangle$ is unifiable, then $\tau(M')$ and $\tau(N')$ are unifiable.

The result now follows by a simple inductive argument on $\sqsupset$. It is easy to see, by induction on the structure of ptypes, that in the pair that is assigned the highest level the two components are of common ptype. Therefore the pair at this level may be unified by the psubstitution ξ_1 which is constructed as in Lemma 3.10

So ξ_1 unifies the pair of highest level in the induced ordering $\sqsupset$. But now, under ξ_1, the components of the pair of level one lower in $\sqsupset$ than those unified by ξ_1 are of common ptype (ξ_1 unifies their ptypes), and so we may apply a similar construction to unify them; of course the pcontext has changed, it is now the range pcontext of the previous psubstitution.

By repeating this process we unify all the pairs of $\mathcal{D}$ and obtain a psubstitution

$$\xi_{\mathcal{D}} =_{\text{def}} \xi_d \circ \ldots \circ \xi_1 ,$$

where d is the number of levels in $\mathcal{D}$, which unifies the **succeed** node labelled with $\mathcal{D}$. Finally, we note that by construction all pairs of similar types represented by $\mathcal{D}$ are unified, so that $\xi_{\mathcal{D}}$ is indeed a substitution. □

3.6 Correctness of the Unification Algorithm

We now prove the correctness of the unification algorithm. This means that we prove a *soundness* theorem, that if a matching tree for two terms $M : A$ and $N : B$ has a success node, then $M : A$ and $N : B$ are unifiable, and a *completeness* theorem, that if $M : A$ and $N : B$ are unifiable, then every matching tree for $M : A$ and $N : B$ has a success node. In this setting, as opposed to that of [14], we must take account of our procedure *reduce* and of the more complex procedure *match*.

THEOREM 3.14 (SOUNDNESS OF THE UNIFICATION ALGORITHM)

Let $\mathcal{M}$ be a matching tree for the pair of terms $M : A$ and $N : B$. For any success node $\mathcal{N}$ in $\mathcal{M}$,

$$\zeta_p \circ \sigma_{\mathcal{N}_p} : \Gamma \longrightarrow \Delta$$

unifies $M : A$ and $N : B$, where Δ is given by the construction given in *match* and where $\sigma_{\mathcal{N}_p}$ and ζ_p are given by the constructions of the proof given below.

PROOF

As usual, the proof is very similar to that of the corresponding result in [14]. Suppose that we have constructed a matching tree for $\mathit{reduce}(A, B) \cup \{\langle M, N \rangle\}$ which possesses a success node on some branch:

$$\mathcal{N}_0 \xrightarrow{\sigma_1} \mathcal{N}_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_p} \mathcal{N}_p, \quad p \geq 0,$$

such that $\mathcal{D}_{\mathcal{N}_p} = \textbf{succeed}$. Let ζ_p be defined as a psubstitution which unifies $\mathcal{D}_{\mathcal{N}_p}$, as constructed in (the proof of) Proposition 3.13:

$$\zeta_p =_{\text{def}} \xi_{\mathcal{D}_{\mathcal{N}_p}}$$

$$\xi_i =_{\text{def}} \xi_{i+1} \circ \sigma_{i+1}, \quad 0 \leq i < p,$$

and prove by descending induction on i that ξ_i unifies $\mathcal{D}_{\mathcal{N}_i}$. The base case of this induction is contained in Proposition 3.13; for the induction step, assume that ξ_{i+1} unifies $\mathcal{D}_{\mathcal{N}_{i+1}}$, and let

$$\overline{\mathcal{D}_{\mathcal{N}_{i+1}}} = \{\langle M'\sigma_{i+1}, N'\sigma_{i+1}\rangle \mid \langle M', N'\rangle \in \mathcal{D}_{\mathcal{N}_i}\} \cup \mathit{reduce}(H_1, H_2)$$

By Proposition 3.9 (Part 2) and the induction hypothesis we have that since

$$\mathcal{D}_{\mathcal{N}_{i+1}} = \mathit{simplify}(\overline{\mathcal{D}_{\mathcal{N}_{i+1}}}),$$

ξ_{i+1} unifies $\overline{\mathcal{D}_{\mathcal{N}_{i+1}}}$; that is, for every $\langle M', N'\rangle \in \mathcal{D}_{\mathcal{N}_i}$, and every $\langle M'', N''\rangle \in \mathit{reduce}(H_1, H_2)$

$$M'\sigma_{i+1}\xi_{i+1} = N'\sigma_{i+1}\xi_{i+1} \qquad \text{and} \qquad M''\sigma_{i+1}\xi_{i+1} = N''\sigma_{i+1}\xi_{i+1}.$$

Therefore ξ_i unifies $\mathcal{N}_i$. This completes the induction step.

In particular we now get that $\xi_0 = \xi_p \circ \sigma_p \circ \ldots \circ \sigma_1$ unifies

$$\mathcal{D}_{\mathcal{N}_0} = \mathit{simplify}(\mathit{reduce}(A, B) \cup \{\langle M, N\rangle\})$$

so that by Proposition 3.9 (Part 2), ξ_0, which is a substitution, not merely a psubstitution, unifies

$$\mathit{reduce}(A, B) \cup \{\langle M, N\rangle\}.$$

By setting $\sigma_{\mathcal{N}_p} =_{\text{def}} \sigma_p \circ \ldots \circ \sigma_1$ and applying Proposition 3.7, the result follows. □

THEOREM 3.15 (COMPLETENESS OF THE UNIFICATION ALGORITHM)

If $\mathcal{U}(M : A, N : B) \neq \emptyset$, then every matching tree for the pair of terms $M : A$ and $N : B$ possesses a success node.

PROOF

Again, the proof is very similar to that of the corresponding result in [14]. We assume that $M : A$ and $N : B$ are unifiable by the psubstitution $\rho : \Gamma \longrightarrow \Delta$, and consider an arbitrary matching tree for

$$\mathit{reduce}(A, B) \cup \{\langle M, N\rangle\}.$$

As in [14], we construct a branch

$$\mathcal{N}_0 \xrightarrow{\sigma_1} \mathcal{N}_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_i} \mathcal{N}_i$$

of this tree, together with a sequence of psubstitutions $\xi_0, \ldots, \xi_i$ such that for all $i \geq 0$, $\xi_i \in \mathcal{U}(\mathcal{D}_{\mathcal{N}_i})$ or $\mathcal{D}_{\mathcal{N}_i} = \textbf{succeed}$, and of strictly decreasing complexity. As in [14], this is proved by ascending induction on i. The base case of the induction is immediate, taking $\xi_0 = \rho$, since if

$$\mathcal{D}_{\mathcal{N}_0} = \mathit{simplify}(\mathit{reduce}(A, B) \cup \{\langle M, N\rangle\}) \neq \textbf{succeed}$$

then by Proposition 3.9 (Part 2), $\rho \in \mathcal{U}(\mathcal{D}_{\mathcal{N}_0})$. For the induction step we assume that $\xi_i \in \mathcal{U}(\mathcal{D}_{\mathcal{N}_i})$. Either $\mathcal{D}_{\mathcal{N}_i} = \textbf{succeed}$, in which case we stop the construction, or we are considering in $\mathcal{D}_{\mathcal{N}_i}$ a pair $\langle M', N'\rangle$ such that N' is rigid. By Proposition 3.9 (Part 1) $\mathcal{D}_{\mathcal{N}_i}$ cannot be **fail** since by the induction hypothesis $\xi_i \in \mathcal{U}(\mathcal{D}_{\mathcal{N}_i})$. By Proposition 3.12, $\mathit{match}(M_1, M_2, \Delta_{\mathcal{N}_i}, \mathcal{D}_{\mathcal{N}_i})$, where $\Delta_{\mathcal{N}_i}$ is the current pcontext for $\mathcal{N}_i$, returns some σ such that there exists η with

$$\xi_i =_{\Delta_{\mathcal{N}_i}} \eta \circ \sigma,$$

so that there exists in the tree a branch $\mathcal{N}_i \xrightarrow{\sigma} \mathcal{N}'$. Choose $\xi_{i+1} = \eta$, $\sigma_{i+1} = \sigma$ and $\mathcal{N}_{i+1} = \mathcal{N}'$. By construction, $\mathcal{D}_{\mathcal{N}_{i+1}} = \mathit{simplify}(\overline{\mathcal{D}_{\mathcal{N}_{i+1}}})$, where

$$\overline{\mathcal{D}_{\mathcal{N}_{i+1}}} = \{\langle M'\sigma, N'\sigma\rangle \mid \langle M', N'\rangle \in \mathcal{D}_{\mathcal{N}_i}\} \cup \mathit{reduce}(H_1, H_2).$$

By the induction hypothesis, $M'\xi_i = N'\xi_i$ for all $\langle M', N'\rangle \in \mathcal{D}_{\mathcal{N}_i}$, so that

$$M'\sigma_{i+1}\xi_{i+1} = N'\sigma_{i+1}\xi_{i+1},$$

since $\Delta_{\mathcal{N}_i}$ contains all the free variables of M' and N'. Since ξ_{i+1} unifies each such $\langle M'\sigma_{i+1}, N'\sigma_{i+1}\rangle$ it follows that $\sigma_{i+1}\xi_{i+1}$ unifies $\mathit{reduce}(H_1, H_2)$. This shows that ξ_{i+1} unifies $\overline{\mathcal{D}_{\mathcal{N}_{i+1}}}$, so that by Proposition 3.9 (Part 2), either $\mathcal{D}_{\mathcal{N}_{i+1}} = \textbf{succeed}$ or ξ_i unifies $\mathcal{D}_{\mathcal{N}_{i+1}}$. By Proposition 3.12, we have that $\theta(\xi_{i+1}) < \theta(\xi_i)$. This concludes the induction step. Therefore our construction yields a **succeed** node of complexity at most $\theta(\rho)$ and the result follows by Proposition 3.7. □

3.7 Complete Unification

Following [14], we investigate the extent to which we are able to enumerate all unifiers of two terms $M{:}A$ and $N{:}B$, which are well-typed in some given context Δ, using our existence algorithm. As in [14], we aim to strengthen the completeness theorem. In fact we have already done all of the necessary work, and indeed the arguments of Section 4.3 of [14] go through almost unaltered.

DEFINITION 3.16 (COMPLETE SETS OF UNIFIERS)

Suppose that N proves $\Delta \vdash_\Sigma M{:}A$ and N proves $\Delta \vdash_\Sigma N{:}B$. A *complete set of unifiers* of $M{:}A$ and $N{:}B$ on Δ is any set $\mathcal{S}$ of psubstitutions such that:

1. $\mathcal{S} \subseteq \mathcal{U}(M{:}A, N{:}B)$;
2. For all $\rho \in \mathcal{U}(M{:}A, N{:}B)$, there exists $\sigma \in \mathcal{S}$ such that $\rho \leq_\Delta \sigma$. □

Now, in [14], Huet defines the notion of a *complete matching tree.* A complete matching tree is distinguished from a matching tree in that the third argument given to his procedure $MATCH$ at each node $\mathcal{N}$ is given by a recursive construction which ensures that this set contains all the new variables introduced on the branch leading to $\mathcal{N}$. In our case however, by construction, the pcontext given as the third argument to *match* already has this property, so that we need no further definitions. Indeed, the proof of Theorem 4.4 of [14] now goes straight through.

THEOREM 3.17 (COMPLETE UNIFICATION)

Let $\mathcal{M}$ be any complete matching tree for the pair $M{:}A$ and $N{:}B$ on Δ. For any $\sigma \in \mathcal{U}(M{:}A, N{:}B)$ there exists a **succeed** node $\mathcal{N}$ in $\mathcal{M}$ such that $\sigma \leq_\Delta \sigma_\mathcal{N}$.
PROOF

As for the corresponding result in [14], we adapt the proof of our earlier completeness theorem. To this end we add the condition $\rho =_\Delta \xi_i \circ \sigma_{\mathcal{N}_i}$ to the induction hypothesis.

The base case is trivially obtained; $\sigma_{\mathcal{N}_0}$ is the identity, $\xi_0 = \rho$. For the induction step, we have that $\xi_i =_{\Delta_{\mathcal{N}_i}} \xi_{i+1} \circ \sigma_{i+1}$, and we note that, since this property is true of $\sigma_1, \sigma_2, \ldots, \sigma_i$, for all of the pairs $\langle M, x\rangle$ of $\sigma_{\mathcal{N}_i}$, $\mathrm{FV}(M) \subset \mathrm{Dom}(\Delta_{\mathcal{N}_i})$. This implies that

$$\xi_i \circ \sigma_{\mathcal{N}_i} =_{\Delta_{\mathcal{N}_i}} \xi_{i+1} \circ \sigma_{i+1} \circ \sigma_{\mathcal{N}_i} = \xi_{i+1} \circ \sigma_{\mathcal{N}_{i+1}},$$

and therefore, since $\mathrm{Dom}(\Delta) \subset \mathrm{Dom}(\Delta_{\mathcal{N}_i})$, that $\rho =_\Delta \xi_{i+1} \circ \sigma_{\mathcal{N}_{i+1}}$. This concludes the induction step. Therefore for the node $\mathcal{N}_p$ labelled with **succeed**, we have that $\rho =_\Delta \xi_p \circ \sigma_{\mathcal{N}_p}$. Hence result. □

3.8 η-reduction

Just as in [14], the addition of η-reduction enables us to simplify the unification algorithm. In particular the procedure *match* is greatly simplified. We consider expressions in long $\beta\eta$-normal form, as defined in Section 2.1. A relevant study of long $\beta\eta$-normal forms appears in [8].

Just as in [14], the following simplifications in *match* are possible:

- We have in *match* that $m = n$, so that we can eliminate Case 1 in *imitation* and Case 2 in *projection*;
- In Case 2 in *imitation* and in Case 1 in *projection* we can restrict to the solution $k = p$.

We note that in this case we need to consider *long head normal forms* in Lemma 2.18.

4 Summary

In this paper we have shown how to extend the unification algorithm for the simply-typed λ-calculus to the $\lambda\Pi$-calculus. Further work based on this paper falls into six main groups.

1. A study of complete-enumeration algorithms. In contrast to [14], the work of Jensen and Pietrzykowski [17] provides an algorithm which enumerates all of the unifiers of a pair of simply-typed λ-terms. This algorithm enumerates all of the unifiers instead of reporting that unifiers exist (with the possibility of constructing certain instances). This is achieved by taking approximation rules of *elimination*, *iteration* and *identification* in addition to the rules of *imitation* and *projection*. The reader is referred to [17] for a detailed explanation of these rules.

   If we take this algorithm as our basis, we conjecture that it is possible to obtain an algorithm for the complete enumeration of all of the unifiers of a pair of well-typed $\lambda\Pi$-terms in a remarkably simple manner. We outline the steps which are necessary in order to enumerate all of the unifiers of a pair of terms which are well-typed in some given context.

   (a) Suppose the well-typed terms are given by $\Delta \vdash_\Sigma M{:}A$ and $\Delta \vdash_\Sigma N{:}B$.

   (b) Apply the procedure *reduce* to the given pair of well-typed terms.

   (c) Enumerate all unifiers of the pairs of objects resulting from the reduction of A and B and by utilizing the ordering given in the proof of Proposition 3.7 (Part 2) and the modified Jensen and Pietrzykowski algorithm; thereby enumerate all unifiers of A and B.

   (d) Each time a unifying substitution, σ, for A and B is calculated, apply it to the objects M and N, and proceed to apply the modified Jensen and Pietrzykowski algorithm to the resulting pair of terms (which now have the same type); thereby enumerate all the unifiers of $M\sigma$ and $N\sigma$.

   (e) If ρ unifies $M\sigma$ and $N\sigma$, then $\sigma\rho$ unifies $M{:}A$ and $N{:}B$.

   Although of little practical value, the existence of such a procedure is important if we wish to state a sufficiently strong completeness theorem for $\lambda\Pi$-calculus search and computation procedures [23].

2. A study of decidable subproblems (*cf.* [20], [21]).

3. A study of heuristic improvments to the algorithm, such as certain suitable heuristic versions of the *occurs-check* [14], [24] (*cf.* [20], [21]), for efficiency.

4. A study of the integration of proof-search and unification, possibly via the notion of *reduction ordering* (a generalization of the occurs-check) of [23], [24] (*cf.* [20], [21]).

5. Extensions of the algorithm to cope with more complicated systems of dependent types.

6. An algebraic study of unification in *Generalized Algebraic Theories* [4], perhaps in the manner suggested by the work of Burstall, Rydeheard and Stell [25], [26]. An indexed category- or fibration-based approach should be used.

5 Acknowledgments

This paper is based on Chapter 5 of the author's Ph.D. thesis [23], which was supervised by Gordon Plotkin and Furio Honsell and examined by Alan Bundy and Gérard Huet.

The author is grateful to Martín Abadi, Alan Bundy, Furio Honsell, Gérard Huet, Dale Miller, Gordon Plotkin and Lincoln Wallen for their help and advice and to the referees for identifying several minor errors.

The author is currently supported by U.K. Science and Engineering Research Council grant GR/E 58588, "Logical and Semantical Frameworks"; and by European Community ESPRIT Basic Research Action 3245, "Logical Frameworks: Design, Implementation and Experiment".

References

[1] Avron, A., Honsell, F., Mason, I. *Using Typed Lambda Calculus to Implement Formal Systems on a Machine.* Report ECS-LFCS-87-31, Laboratory for Foundations of Computer Science, University of Edinburgh, 1987.

[2] Barendregt, H. *Introduction to Generalized Type Systems.* Journal of Functional Programming 1(2): pp. 125–154, April 1991.

[3] Cartmell, J. *Generalized algebraic theories and contextual categories.* D.Phil. thesis. University of Oxford, 1978.

[4] Cartmell, J. *Generalized Algebraic Theories and Contextual Categories.* Annals of Pure and Applied Logic 32(1986), pp. 209-243.

[5] Coquand, Th. *Une théorie des constructions.* Thèse de Troisième Cycle. Université de Paris VII, 1985.

[6] Coquand, Th. *An Algorithm for testing conversion in type theory.* In: G. Huet and G. Plotkin (editors), *Logical Frameworks.* Cambridge University Press, 1991. pp. 255-279.

[7] van Daalen, D. T. *The Language Theory of AUTOMATH.* Ph.D. thesis. Technical University of Eindhoven, 1980.

[8] Dowek, Gilles *Démonstration Automatique dans le Calcul des Constructions.* Thèse de Doctorat, Université de Paris 7, 1991.

[9] Elliott, Conal M. *Extensions and Applications of Higher-order Unification.* Ph.D. thesis, School of Computer Science, Carnegie Mellon University, 1990. Available as: Technical Report CMU-CS-90-134.

[10] Harper, R. *An Equational Formulation of* LF. University of Edinburgh report ECS-LFCS-88-67, 1987.

[11] Harper, R., Honsell, F., Plotkin, G. *A Framework for Defining Logics.* Proc. Second Annual Symposium on Logic in Computer Science, pp. 194–204. IEEE, 1987.

[12] Harper, R., Honsell, F., Plotkin, G. *A Framework for Defining Logics.* To appear in the J. Assoc. Comp. Mach.. Available as: University of Edinburgh report ECS-LFCS-91-162, 1991.

[13] Hindley, J.R., Seldin, J.P. *Introduction to Combinators and λ-Calculus.* Cambridge University Press, 1986.

[14] Huet, G. *A Unification Algorithm for Typed λ-calculus.* Theor. Comp. Sci., 1975.

[15] Huet, G. *Résolution d'Équations dans les Languages d'Ordre* $1, 2, \ldots, \omega$. Thèse de Doctorat d'Etat. Université de Paris VII, 1976.

[16] Jacobs, B.P.F. *Categorical Type Theory.* Doctoral thesis, Katholieke Universiteit Nijmegan, 1991.

[17] Jensen, D. Pietrzykowski, T. *Mechanizing ω-order Type Theory Through Unification.* Th. Comp. Sci. 3(1976), pp. 123-171.

[18] Meyer, A. *What is a Model of the Lambda Calculus ?* Information and Control 52(1982), pp. 87-122.

[19] Pavlović, Duško *Predicates and Fibrations.* Doctoral thesis, Rijksuniversiteit Utrecht, 1990.

[20] Pfenning, Frank *Logic Programming in the LF Logical Framework.* In: G. Huet and G. Plotkin (editors), *Logical Frameworks.* Cambridge University Press, 1991. pp. 149-181.

[21] Pfenning, Frank *Unification and anti-unification in the Calculus of Constructions.* In: Proc. Sixth Annual IEEE Symposium on Logic in Computer Science, pp. 74-85. IEEE Computer Society Press, July 1991.

[22] Moggi, E. *A category-theoretic account of program modules.* Math. Struct. in Comp. Science (1991), vol. 1, pp. 103–139.

[23] Pym, D.J. *Proofs, Search and Computation in General Logic.* Ph.D. thesis, Department of Computer Science, University of Edinburgh, 1990. Available as: University of Edinburgh report CST-69-90, 1990. (Also published as LFCS report ECS-LFCS-90-125.)

[24] Pym, D., Wallen, L. *Proof-search in the λΠ-calculus.* In: G. Huet and G. Plotkin (editors), *Logical Frameworks.* Cambridge University Press, 1991. pp. 309-340.

[25] Rydeheard, D., Burstall, R.M. *The Unification of Terms: A Category-Theoretic Algorithm.* Report CSR-192-85, Department of Computer Science, University of Edinburgh, 1985.

[26] Rydeheard, D., Stell, J. *Foundations of Equational Deduction: A Categorical Treatment of Equational Proofs and Unification Algorithms.* In: Category Theory and Computer Science, Pitt, D.H., Poigne, A., Rydeheard, D.E. (editors). Lecture Notes in Computer Science 283, Springer-Verlag, 1987.

[27] Salvesen, A. Manuscript. University of Edinburgh, 1989.

[28] Streicher, T. *Correctness and Completeness of a Semantics of the Calculus of Constructions.* Ph.D. thesis, University of Passau, 1988.

A Contextual Categories and Indexed Categories

A.1 Contextual Categories

We present definition and basic theory of contextual categories as developed by Cartmell [3], [4] and further developed and presented by Streicher [28]. For this we shall need to define the notion of a *tree structure* on a set.

DEFINITION A.1 (TREE STRUCTURE)

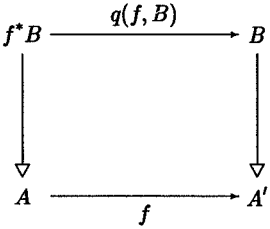
Let N be a set (of nodes). A *tree structure* on N is a pair $\lhd = \langle p, l\rangle$ where $p : N \longrightarrow N$ (the predecessor function) and $l : N \longrightarrow \mathbb{N}$ (the level function) are such that:

1. For all $x, y \in N$, if $l(x) = l(y) = 0$ then $x = y$ and $p(x) = x$;
2. For all $x \in N$ and $n \in \mathbb{N}$, if $l(x) = n + 1$ then $l(p(x)) = n$. □

DEFINITION A.2 (CONTEXTUAL CATEGORIES)

A *contextual category* consists of:

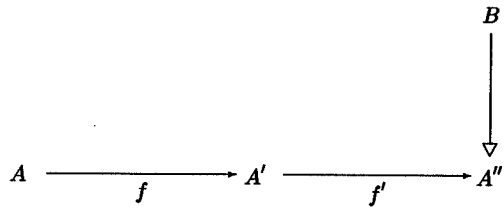
1. A category $\mathcal{C}$ with terminal object ϵ;
2. A tree structure, $\lhd = \langle p, l\rangle$, on the objects of $\mathcal{C}$ such that the terminal object ϵ is the unique least element, under the ordering determined by l, of the tree. We write $A \lhd B$ if the objects A and B of $\mathcal{C}$ are such that $p(B) = A$;
3. For all A, $B \in |\mathcal{C}|$ such that $A \lhd B$, a morphism $p(B) : B \longrightarrow A$ in $\mathcal{C}$. This morphism will also be written as $B \longrightarrow\!\!\!\!\triangleright\, A$;
4. For all A, $A' \in |\mathcal{C}|$, for all $f : A \longrightarrow A'$ in $\mathcal{C}$ and for all $B \in |\mathcal{C}|$ such that $A' \lhd B$, an object $f^*B \in |\mathcal{C}|$ and a morphism $q(f, B) : f^*B \longrightarrow B$ such that the diagram



is a pullback diagram in $\mathcal{C}$.

We also require two coherence conditions.

- For all A, $B \in |\mathcal{C}|$ such that $A \triangleleft B$, $1_A^* B = B$ and $q(1_A, B) = 1_B$, where 1_A is the identity morphism on A.
- If



in $\mathcal{C}$, then $(f \circ f')^* B = f^*(f'^* B)$ and $q(f \circ f', B) = q(f, f'^* B) \circ q(f', B)$. □

If $g : B \longrightarrow B'$ is another morphism over A', then we write $f^* g$ for the unique morphism m such that $m \circ q(f, B') = q(f, B) \circ g$ and $p(f^* B) = m \circ p(f^* B')$.

DEFINITION A.3 (THE RELATION $\blacktriangleleft$)

$A \blacktriangleleft B$ if $A \triangleleft \ldots \triangleleft B$. □

The morphisms of Part 3 of Definition A.2.2 can be composed to give a morphism $p(B, A)$, where $A \blacktriangleleft B$.

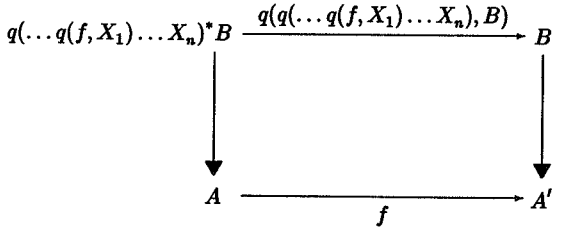
DEFINITION A.4 (MULTI-STEP p)

If $A \blacktriangleleft B$ in the contextual category $\mathcal{C}$, then we define the morphism $p(B, A) : B \longrightarrow A$ in $\mathcal{C}$, which will also be written as $B \longrightarrow\!\!\blacktriangleright A$, by

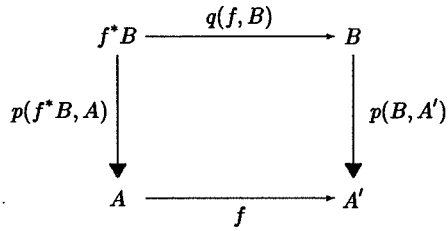
$$p(B, A) = p(B) \circ p(X_n) \circ \ldots \circ p(X_1)$$

where $X_1, \ldots, X_n$ is the unique sequence of objects of $\mathcal{C}$ such that $A \triangleleft X_1 \triangleleft \ldots \triangleleft X_n \triangleleft B$ in $\mathcal{C}$. In the case $A = B$, $p(B, A) = 1_A$. □

We can obtain a pullback for any morphism of the form $p(B, A)$ along any morphism with codomain A by repeatedly forming the pullback along each $p(X_i)$. Thus, if $A' \blacktriangleleft B$ in $\mathcal{C}$ and if $f : A \longrightarrow A'$ is a morphism in $\mathcal{C}$ then we have a pullback for the morphism $p(B, A')$ along f given by the diagram



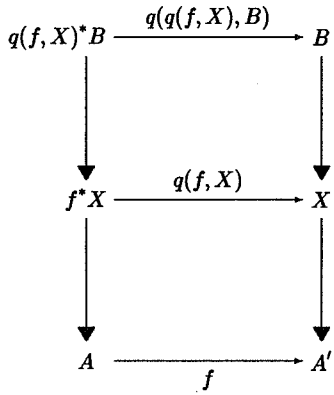
We extend the * and q notation to these new pullback diagrams, so that for $f : A \longrightarrow A'$ and $A' \blacktriangleleft B$



is the canonical pullback diagram.

LEMMA A.5 (COMPOSITION OF PULLBACKS)

If $f : A \longrightarrow A'$ and $A' \blacktriangleleft X \blacktriangleleft B$ in the contextual category $\mathcal{C}$ then $f^*B = q(f,X)^*B$ and $q(f,B) = q(q((f,X),B))$, *i.e.*,



□

A set-theoretic example of a contextual category is the category $\mathcal{F}$.

DEFINITION A.6 (THE CONTEXTUAL CATEGORY OF FAMILIES OF SETS, $\mathcal{F}$)

The category $\mathcal{F}$ of families of sets is defined as a construction on the category of (small) sets and functions, $\mathcal{S}$, as follows:

- Objects, level and denotation, DEN:
  - ϵ is the unique context of level 0, and $\mathrm{DEN}(\epsilon) = \{\emptyset\}$;

– A context A, of level $n+1$, is a pair $\langle B, F\rangle$ where B is of level n and

$$F : \mathrm{DEN}(B) \longrightarrow \mathcal{S}$$

is a family of sets indexed by the elements of DEN(B) and DEN(A) is defined by

$$\mathrm{DEN}(A) = \mathrm{DEN}(\langle B, F\rangle) = \{\langle x, y\rangle \mid x \in \mathrm{DEN}(B) \text{ and } y \in F(x)\};$$

- Morphisms:
  – If A and B are objects of $\mathcal{F}$ then the morphisms from A to B are the functions from DEN(A) to DEN(B) (which are morphisms in $\mathcal{S}$). □

The reader is referred to the work of Cartmell [3], [4] or Streicher [28] for the details of the straightforward proof of the next proposition.

PROPOSITION A.7 ($\mathcal{F}$ IS A CONTEXTUAL CATEGORY)

The category $\mathcal{F}$ is structured as a contextual category as follows:

- The unique terminal object is ϵ;
- If $A = \langle B, F\rangle$ is an object of $\mathcal{F}$ then $p(A)$ is the projection on the first component, *i.e.*, for $x \in \mathrm{DEN}(B)$ and $y \in F(x)$, $p(A)(\langle x, y\rangle) = x$. The level function is that given by Definition A.2.6. Thus the tree structure is determined;
- If $A = \langle B, F\rangle$ is an object of $\mathcal{F}$ and if $f : C \longrightarrow B$ is a morphism then $f^*A = \langle C, F \circ f\rangle$;
- If $A = \langle B, F\rangle$ is an object of $\mathcal{F}$ and if $f : C \longrightarrow B$ is a morphism then $q(f, A)$ is the function from $\mathrm{DEN}(f^*A)$ to DEN(A) such that for all $x \in \mathrm{DEN}(C)$ and $y \in F(f(x))$, $q(f, A)(\langle x, y\rangle) = \langle f(x), y\rangle$. □

The structure preserving morphisms (or homomorphisms) between contextual categories are *contextual functors.*

DEFINITION A.8 (CONTEXTUAL FUNCTORS)

Let $\mathcal{C}$ and $\mathcal{C}'$ be contextual categories. A *contextual functor* $F : \mathcal{C} \longrightarrow \mathcal{C}'$ is a functor $F : \mathcal{C} \longrightarrow \mathcal{C}'$ such that:

1. $F(\epsilon) = \epsilon$, and if $A \triangleleft B$ in $\mathcal{C}$, then $F(A) \triangleleft F(B)$ in $\mathcal{C}'$;
2. For all objects A of $\mathcal{C}$, $F(p(A)) = p(F(A))$;
3. For all f and B such that f^*B is defined in $\mathcal{C}$, $F(f^*B) = F(f)^*F(B)$ and $F(q(f, B)) = q(F(f), F(B))$. □

A.2 Contextual Categories with Products

The definition of a contextual category allows us to consider families of types, but does not include any notion of the "product of a family of types".

The Appropriate type-theoretic intuition for the product $B(x)$ of a family of types indexed by $x \in A$ is as the type of all functions f whose domain is the type A and whose value $f(a)$ on an object $a \in A$ is an object of $B(a)$.

With this in mind, we proceed to define the notion of a *contextual category with products.*

The content of the next definition is due to Cartmell [3], but the notation $sections_{\mathcal{C}}(B)$ is due to Streicher [28]. We adopt Streicher's more helpful notation. This definition provides a simple mechanism for imposing a simple coherence condition in the definition of a contextual category with products.

DEFINITION A.9 (THE SET $sections_{\mathcal{C}}(B)$)

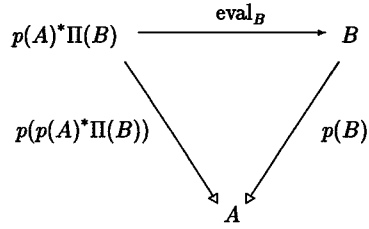
Let $\mathcal{C}$ be a contextual category and let A, B be objects of $\mathcal{C}$ such that $A \lhd B$.

$$sections_{\mathcal{C}}(B) =_{\text{def}} \{f \in \hom_{\mathcal{C}}(A, B) \mid p(B) \circ f = 1_A\} . \square$$

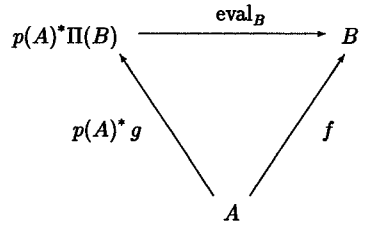
DEFINITION A.10 (CONTEXTUAL CATEGORIES WITH PRODUCTS)

Let $\mathcal{C}$ be a contextual category. $\mathcal{C}$ is a *contextual category with products* if it satisfies:

1. If $C \lhd A \lhd B$ in $\mathcal{C}$ there is an object $\Pi(B)$ of $\mathcal{C}$ such that $C \lhd \Pi(B)$, and a morphism $\text{eval}_B : p(A)^*\Pi(B) \longrightarrow B$ such that the diagram



commutes, and such that $\Pi(B)$ and eval_B have the property that for every morphism $f \in sections_{\mathcal{C}}(B)$ there is a unique morphism $g \in sections_{\mathcal{C}}(\Pi(B))$ such that the diagram



commutes. Following Streicher [28], we write $g = curry(f)$ and $f = uncurry(g)$;

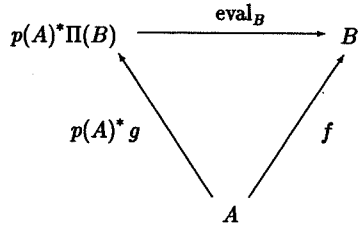
2. If $m : P \longrightarrow Q$ is a morphism in $\mathcal{C}$ and $Q \triangleleft A \triangleleft B$, then
   - $m^*\Pi(B) = \Pi(m^*B)$ and
   - $m^*\mathrm{eval}_B = \mathrm{eval}_{m^*B}$. □

This clause is the appropriate formulation of the *Beck-Chevalley condition.*[19] □

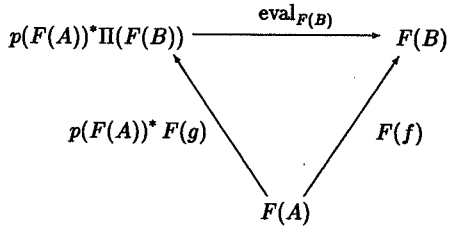
We define the notion of a contextual functor which preserves products.

DEFINITION A.11 (PRODUCT-PRESERVING CONTEXTUAL FUNCTORS)

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be contextual categories with products and let $F : \mathcal{C}_1 \longrightarrow \mathcal{C}_2$ be a contextual functor. F is said to *preserve products* if whenever



is a product diagram in $\mathcal{C}_1$,



is a product diagram in $\mathcal{C}_2$. □

The reader is referred to [3] and [28] for the proof of the next proposition.

PROPOSITION A.12 ($\mathcal{F}$ IS A CONTEXTUAL CATEGORY WITH PRODUCTS)

The category $\mathcal{F}$ is structured as a contextual category with products as follows:
Let A be an object of $\mathcal{F}$ such that $B = \langle\langle A, F\rangle, G\rangle$;

[19] In fact, it is possible to reformulate the statement of this condition so that it more closely resembles the usual presentation of the Beck-Chevalley condition, see [28].

- We define the object $\Pi(B)$ by $\Pi(B) =_{\text{def}} \langle A, P\rangle$, where $P(x)$ is defined by:

$$P(x) \;=_{\text{def}}\; \{f : F(x) \longrightarrow \bigcup\{G(\langle x, y\rangle) | y \in F(x)\} | \text{ for all } y \in F(x), f(y) \in G(\langle x, y\rangle)\};$$

- The morphism $\text{eval}_B : p(A)^*\Pi(B) \longrightarrow B$ is defined by $\text{eval}_B(\langle x, y\rangle, f\rangle) = \langle x, f(y)\rangle$ for each $x \in \text{DEN}(A)$, $y \in F(x)$ and $f \in P(x)$. □

If in the definition of a contextual category with products we drop the requirement that g be unique, then we speak of a *contextual category with weak products.*

A.3 Indexed Categories

Following [22], we give a simplified notion of indexed category, which is adequate for our purposes. For a general discussion of fibred categories, including fibrations, indexed categories and contextual categories together with various equivalences between these structures, the reader is referred to [16], [19] and [28]. Let **Cat** denote the category of small categories and functors.

DEFINITION A.13 (INDEXED CATEGORIES)

Let B be a small category, the *base category.* A *B-indexed category* is a functor $F : B^{\text{op}} \longrightarrow \mathbf{Cat}$:

$$Y \xrightarrow{f} X \quad \mapsto \quad F(X) \xrightarrow{F(f)} F(Y).$$

We refer to $F(X)$ as the *fibre over* X. □

The reader is also referred to [22] for a suitable definition of the 2-category of indexed categories over a given base category.