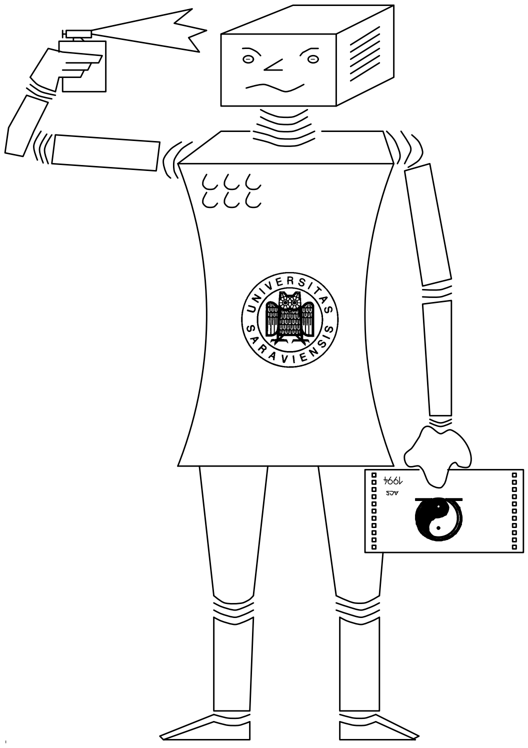
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: `http://js-sfbsun.cs.uni-sb.de/pub/www/`



Unification in an Extensional Lambda Calculus with Ordered Function Sorts and Constant Overloading

Patricia Johann and Michael Kohlhase

Published as: In Alan Bundy, editor, *Automated Deduction — CADE-12*, Proceedings of the 12th International Conference on Automated Deduction, pages 371–385, Nancy, France, 1994. Springer Verlag, Berlin, Germany. LNAI 814.

Unification in an Extensional Lambda Calculus with Ordered Function Sorts and Constant Overloading

Patricia Johann* and Michael Kohlhase**
Fachbereich Informatik
Universität des Saarlandes
66123 Saarbrücken, Germany
{*pjohann, kohlhase*}*@cs.uni-sb.de*

Abstract. We develop an order-sorted higher-order calculus suitable for automatic theorem proving applications by extending the extensional simply typed lambda calculus with a higher-order ordered sort concept and constant overloading. Huet's well-known techniques for unifying simply typed lambda terms are generalized to arrive at a complete transformation-based unification algorithm for this sorted calculus. Consideration of an order-sorted logic with functional base sorts and arbitrary term declarations was originally proposed by the second author in a 1991 paper; we give here a corrected calculus which supports constant rather than arbitrary term declarations, as well as a corrected unification algorithm, and prove in this setting results corresponding to those claimed there.

1 Introduction

In the quest for calculi best suited for automating logic, the introduction of sorts has been one of the most promising developments. Sorts, which are intended to capture for automated deduction purposes the kinds of meta-level taxonomic distinctions that humans naturally assume structure the universe, can be employed to syntactically distinguish objects of different classes. The essential idea behind sorted logics is to assign sorts to objects and to restrict the ranges of variables to particular sorts, so that unintended inferences, which then violate the constraints imposed by this sort information, are disallowed. These techniques have been seen to dramatically reduce the search space associated with deduction in first-order systems ([Wal88], [Coh89], [Sch89]).

On the other hand, the inherently higher-order nature of many problems whose solutions one would like to deduce automatically has sparked an increasing interest in higher-order deduction. The behavior of sorted higher-order calculi, which boast both the expressiveness of higher-order logics and the efficiency of sorted calculi, is thus a natural topic of investigation. In this paper, we develop precisely such a calculus — an order-sorted lambda calculus supporting functional base sorts and constant

* On leave from the Department of Mathematics and Computer Science, Hobart and William Smith Colleges, Geneva, NY 14456, *johann@hws.bitnet*. This material is based on work supported by the National Science Foundation, Grant No. INT-9224443.

** Supported by the Deutsche Forschungsgemeinschaft (SFB 314).

overloading — as well as a complete unification algorithm for this calculus, which is suitable for use in an automated deduction setting. Calculi intended for actual mathematical deduction will no doubt support constant — if not arbitrary term — declarations (see Example 3.7); by incorporating constant declarations into our calculus, we treat deduction issues common to all mathematically useful extensional order-sorted higher-order logics supporting functional base sorts.

Although Huet proposed the study of a simple sorted lambda calculus in an appendix to [Hue72], the development of order-sorted higher-order calculi for use in deduction systems has only in recent years been pursued ([Koh92], [NQ92], [Pfe92]). There has, however, been considerable interest in order-sorted higher-order logic from the point of view of higher-order algebraic specifications, the theory of functional programming languages, and object-oriented programming ([Car88], [BL90], [Qia90], [CG91], [Pie91]).

In unsorted logics, the knowledge that an object is a member of a certain class of objects is expressed using unary predicates. This leads to a multitude of unit clauses in deductions, each of which carries only taxonomic information and contributes to a severe explosion of the search space. In sorted logics, predicates are replaced by sorts carrying precisely the same taxonomic information, so that their attendant unit clauses are also eliminated and the search space is correspondingly pruned. The incorporation of sort information is perhaps even more natural for higher-order than for first-order logics: type information in higher-order logics can be regarded as coding very coarse distinctions between disjoint classes of objects, so that sorts merely refine an already present structure. But more importantly, the benefits of sorts for restricting search spaces in higher-order deduction will necessarily be more pronounced than in first-order systems, since the sort hierarchy propagates into the higher-order structure of the logics.

Sorting the universe of individuals in higher-order logics gives rise to new classes of functions, namely those whose domains and codomains are (denoted by) the sorts. But in addition to sorting function universes in such a first-order manner, classes of functions defined by domains and codomains can themselves be divided into subclasses since functions are explicit objects of higher-order logics. Functional base sorts, *i.e.*, base sorts that denote classes of functions, are thus permitted. Syntactically, each sort A comes with a type, a codomain sort $\gamma(A)$, and — if of functional type — also with a domain sort $\delta(A)$. Partial orders on the set of sorts, capturing inclusion relations among the various classes of objects, are induced by covariance in the codomain sort via subsort declarations. But in the presence of functional base sorts an additional mechanism for inducing subsort information is needed: since any function of sort A is a function with domain $\delta(A)$ and codomain $\gamma(A)$, a functional sort A must always be a subsort of the sort $\delta(A) \to \gamma(A)$.

The calculus presented here supports constructs for restricting the ranges of variables to, and assigning constants membership in, certain classes of objects. Depending on the partial order induced on the sorts, certain classes of terms built from these atoms then become the objects of study — the partial order restricts the class of models for the calculus, so that terms must meet certain conditions to denote meaningful objects, *i.e.*, to be well-sorted. Notions of β- and η-reduction generalizing the corresponding reductions in the simply typed lambda calculus are defined on the class of well-sorted terms. The former is a straightforward adaptation of typed β-

reduction, but the delicate interaction between extensionality and partially ordered sorts necessitates care in defining the latter. If X is a term of functional sort A, for example, and x is a variable whose range is restricted to the subsort B of $\delta(A)$, then $\lambda x.Xx$ denotes the restriction of the function (denoted by) X to the domain (specified by) B. In order to properly model extentionality by η-reduction, B must therefore be precisely the (maximal) domain of X in order for $\lambda x.Xx$ to η-reduce to X — otherwise X would be equal to a proper restriction of itself.

A similar subtle interplay between extensionality and functional base sorts renders the natural generalization of Huet's ([Hue75]) classical method for unification of simply typed lambda terms inadequate in our setting. Nevertheless, a more liberal notion of partial binding, which in particular does not require the bindings to be η-expanded, does suffice for incrementally approximating answer substitutions for arbitrary unification problems modulo $\beta\eta$-equality on well-sorted terms.

As in the simply typed lambda calculus, the need for "guessing" partial bindings for pairs so called *flex-flex pairs* gives rise to a serious explosion of the search space, but unfortunately, this cannot be avoided without sacrificing the unification completeness of our algorithm. Huet resolved this difficulty in the simply typed lambda calculus by redefining the higher-order unification problem to a form sufficient for refutation purposes: flex-flex pairs are considered to *pre-unified*, or already solved. We conjecture that it is possible to define an appropriate notion of pre-unification in our setting as well, but warn that a naive modification of the standard methods is evidently insufficient for calculi supporting functional base sorts. Specifically, pre-unification only makes sense under regular signatures, and the existence of unifiers for flex-flex pairs depends heavily on the partial order on sorts under which unification is being considered.

Unification in an extensional order-sorted lambda calculus with functional base sorts was first investigated in [Koh92]. A calculus supporting functional base sorts and arbitrary term, rather than only constant, declarations is proposed there, but its presentation is flawed in serveral places. Our calculus can be seen as a subcalculus of the one proposed in [Koh92] which has been corrected to be well-defined and to properly incorporate extensionality (see the problematic clauses 4 and 5 of Definition 2.5, and Remark 2.10, there). The notion of partial binding developed here paves the way for remedying both the ill-defined unification transformations and the flawed completeness proof of [Koh92]. For a detailed treatment of our results and the issues surrounding them, the reader is referred to the full paper [JK93].

2 The Calculus

The set of *types* $\mathcal{T}$ is obtained by inductively closing a set of *base types* $\mathcal{T}_0$ under the operation $\alpha \to \beta$; assuming right-associativity of $\to$, the *length* of a type $\alpha \equiv \alpha_1 \to \alpha_2 \to \ldots \to \alpha_n$, denoted $length(\alpha)$, is $n-1$. Types are denoted by lower case Greek letters. In theorem proving applications we might have only two base types, o denoting truth-values and ι denoting the universe of individuals, with all other subdivisions of the universe being coded into sort distinctions among individuals, as described in the next subsection.

For each type $\alpha \in \mathcal{T}$, fix a countably infinite set of variables $x_\alpha, y_\alpha, z_\alpha, \ldots$ of type α and a countably infinite set of constants $a_\alpha, b_\alpha, c_\alpha, \ldots$ of type α. We assume that no two distinct variables or constants have the same type-erasure.

$\mathcal{LC}$ is the set of explicitly simply typed lambda terms over the variables and constants. We omit reference to the type of X when this will not lead to confusion. On $\mathcal{LC}$, $\beta\eta$-equality is generated by $\beta\eta$-reduction, denoted by $\xrightarrow{\beta\eta}$ and determined by the usual rules $(\lambda x.X)Y \xrightarrow{\beta} X[x := Y]$ and $\lambda x.Xx \xrightarrow{\eta} X$. $\beta\eta$-reduction is terminating and confluent (*i.e., convergent*) on $\mathcal{LC}$-terms.

The reflexive, transitive closure of a reduction relation $\xrightarrow{\nu}$ is denoted $\xrightarrow{\nu}\!\!\!\!\rightarrow$, and we write $=_\nu$ for the symmetric closure of $\xrightarrow{\nu}\!\!\!\!\rightarrow$. We write $X \equiv Y$ to indicate that two $\mathcal{LC}$-terms X and Y are identical up to renaming of bound variables. As is customary, we consider $\mathcal{LC}$-terms identical up to renaming of bound variables to be the same.

2.1 Order-sorted Structures

As described in the introduction, we capitalize on the fact that functions are explicit objects of higher-order logic by allowing classes of functions defined by domains and codomains to themselves be divided into subclasses. We thus postulate both functional base sorts — *i.e.,* base sorts that denote classes of functions — as well as non-functional base sorts.

Definition 2.1 A *sort system* is a quintuple $(\mathcal{S}_0, \mathcal{S}, \tau, \delta, \gamma)$ such that:

- $\mathcal{S}_0$ is a set of *base sorts* distinct from the set of type symbols. The set of *sorts* obtained by closing $\mathcal{S}_0$ under the operation $A \rightarrow B$ comprises $\mathcal{S}$.
- The *type function* τ is a mapping $\tau : \mathcal{S}_0 \rightarrow \mathcal{T}$. If $\tau(A) \in \mathcal{T}_0$, then A is said to be *non-functional*, and A is said to be *functional* otherwise; the set of non-functional (resp., functional) sorts is denoted by $\mathcal{S}^{nf}$ (resp., $\mathcal{S}^f$). For all $A \in \mathcal{S}^f$, we require that $\tau(A) = \tau(\delta(A)) \rightarrow \tau(\gamma(A))$, where the *domain sort function* δ is a map $\delta : \mathcal{S}_0^f \rightarrow \mathcal{S}$, the *codomain sort function* γ is a map $\gamma : \mathcal{S}_0 \rightarrow \mathcal{S}$ with $\gamma\mid_{\mathcal{S}^{nf}}$ the identity map, and the mappings δ and γ are extended to $\mathcal{S}$ by defining $\delta(A) = B$ and $\gamma(A) = C$ for $A \equiv B \rightarrow C \in \mathcal{S}$.

Sorts are denoted by upper case Roman letters. If the context is clear, we abbreviate by $\mathcal{S}$ the sort system $(\mathcal{S}_0, \mathcal{S}, \tau, \delta, \gamma)$. Since we are ultimately interested in sorted terms and their typed counterparts, we only consider sort systems for which τ is surjective. We further assume that for each $\alpha \in \mathcal{T}$ there exist only finitely many $A \in \mathcal{S}_0$ with $\tau(A) = \alpha$.

It will be useful to have some notational conventions for domain and codomain sorts. For any $A \in \mathcal{S}$, define the following notation: $\delta^0(A) \equiv A$, $\gamma^0(A) \equiv A$, and for $i \geq 1$, $\gamma^i(A) \equiv \gamma(\gamma^{i-1}(A))$, and $\delta^i(A) \equiv \delta(\gamma^{i-1}(A))$. Write $length(A)$ for the *length* of the sort A.

Example 2.2 Functional base sorts are useful in the study of elementary analysis, where we might postulate a non-functional base sort R denoting the reals and a functional base sort C with $\delta(C) = R$ and $\gamma(C) = R$ denoting the class of real-valued continuous functions on the reals. Since it is not possible to distinguish syntactically such continuous functions solely in terms of their domains and codomains, permitting functional base sorts indeed increases the expressiveness of a calculus.

While types represent disjoint classes of objects, certain kinds of orderings on sorts reflect permissible inclusion relations among classes of objects sorts denote. We capture a consistency condition which such orderings are required to satisfy by defining, for a sort system $\mathcal{S}$ and a pair of sorts A and B in $\mathcal{S}$ such that $\tau(A) = \tau(B)$, the set $\mathrm{Con}(A, B)$ of *subsort declarations* (for $\mathcal{S}$) to be the set $\{[A \leq B]\}$ if $A, B \in \mathcal{S}^{nf}$, and

$$\mathrm{Con}(\delta(A), \delta(B)) \cup \mathrm{Con}(\delta(B), \delta(A)) \cup \mathrm{Con}(\gamma(A), \gamma(B)) \cup \{[A \leq B]\}$$

if $A, B \in \mathcal{S}^f$. A *sort structure* (for $\mathcal{S}$) is any set of subsort declarations obtained by inductively adding sets of the form $\mathrm{Con}(A, B)$ to the empty set. Since each set $\mathrm{Con}(A, B)$ of subsort declarations is finite, sort structures are necessarily finite. For any sort structure Δ, we have $[A \leq B] \in \Delta$ iff $\mathrm{Con}(A, B) \subseteq \Delta$.

Any sort structure Δ induces an *inclusion ordering* $\leq_\Delta$ (or simply "$\leq$") on $\mathcal{S}$, inductively defined by the rules of Definition 2.3.

Definition 2.3 For any sort structure Δ, the *inclusion ordering determined by* Δ contains all judgements of the form $\Delta \vdash A \leq B$ provable by the following calculus:

$$\frac{[A \leq B] \in \Delta}{\Delta \vdash A \leq B} \qquad \frac{A \in \mathcal{S}^f}{\Delta \vdash A \leq \delta(A) \to \gamma(A)}$$

$$\frac{}{\Delta \vdash A \leq A} \qquad \frac{\Delta \vdash A \leq B}{\Delta \vdash C \to A \leq C \to B}$$

$$\frac{\Delta \vdash A \leq B \qquad \Delta \vdash B \leq C}{\Delta \vdash A \leq C}$$

Clearly we cannot insist that $\Delta \vdash A \leq B$ hold for any sorts A and B with a common domain sort C and codomain sorts satisfying $\Delta \vdash \gamma(A) \leq \gamma(B)$ (assuming, for example, a standard semantics). But if Δ is a sort structure for $\mathcal{S}$, and $\sim$ is the equivalence relation induced by $\leq$, then $A, B \in \mathcal{S}^f$, $\Delta \vdash A \leq B$ implies $\Delta \vdash \delta(A) \sim \delta(B)$ and $\Delta \vdash \gamma(A) \leq \gamma(B)$. In addition, for all $A, B \in \mathcal{S}$, $\Delta \vdash A \leq B$ implies $\tau(A) = \tau(B)$, so that any sort system $\mathcal{S}$ is the disjoint union of infinitely many subsets $\mathcal{S}_\alpha = \{A \in \mathcal{S} \mid \tau(A) = \alpha\}$ of sorts such that if $A \in \mathcal{S}_\alpha$ and $B \in \mathcal{S}_\beta$ with $\alpha \not\equiv \beta$, then A and B are incomparable with respect to $\leq$. Since $\mathcal{S}$ has only finitely many base sorts per type, each subset $\mathcal{S}_\alpha$ is finite. Decidability of the inclusion ordering determined by any sort structure thus follows from the next lemma, which is proved by induction on $length(\alpha)$.

Lemma 2.4 *For any type $\alpha \in \mathcal{T}$ and any sort structure Δ, if $\leq$ is the inclusion ordering determined by Δ, then the restriction $\leq_\alpha$ of $\leq$ to sorts of type α is effectively computable.*

Theorem 2.5 *The inclusion ordering determined by any sort structure Δ is decidable.*

It will be important that the signatures over which our well-sorted terms are built "respect function domains," *i.e.*, that for any term X and any sorts A and B such that X has sort A and also sort B, $\delta(A) \sim \delta(B)$ holds. The proof that signatures indeed satisfy this property (see Lemma 2.11) depends in part on the consistency conditions for sort structures and in part on the fact that constant declarations meet the sort condition of the fifth clause of Definition 2.7 below, given in terms of the equivalence relation Rdom, which we now define.

Definition 2.6 Given a sort structure Δ for $\mathcal{S}$ and a pair of sorts A and B in $\mathcal{S}$, A Rdom_Δ B holds if either $A, B \in \mathcal{S}^{nf}$ and $\tau(A) = \tau(B)$, or if $A, B \in \mathcal{S}^f$, $\Delta \vdash \delta(A) \sim \delta(B)$, and $\gamma(A)$ Rdom_Δ $\gamma(B)$.

We write "Rdom" for Rdom_Δ when Δ can be discerned from the context. Then A Rdom B implies $\tau(A) = \tau(B)$, and $\Delta \vdash A \leq B$ implies A Rdom B.

Definition 2.7 A *signature* Σ comprises *i*) a sort system $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}, \delta, \gamma, \tau)$, *ii*) a sort structure Δ (for $\mathcal{S}$), *iii*) a countably infinite set $Vars_A$ of *variables* $x_A, y_A, z_A, \ldots$ for each $A \in \mathcal{S}$, *iv*) a set $\mathcal{C}$ of typed constant symbols, and *v*) a set of *constant declarations* of the form $[c_\alpha :: A]$ for $c \in \mathcal{C}$ such that $\tau(A) = \alpha$. We assume that if $[c :: A]$ and $[c :: B]$ are constant declarations, then A Rdom B.

The requirement that $\tau(A) = \alpha$ for a constant declaration $[c_\alpha :: A]$ insures that sort assignments respect the types of constants. In a theorem proving context, any signature would have, for each $\alpha \in \mathcal{T}$, only finitely many constant declarations involving constants of type α. We will assume this restriction on signatures.

Any sorted variable can naturally be regarded as a typed variable by "forgetting" its sort information. Denoting the forgetful functor by $\overline{}$, we may regard the sorted variable x_A as the typed variable $\overline{x_A}$, *i.e.,* as $x_{\tau(A)}$. By prudently naming the variables, we can arrange that the forgetful functor is bijective on variables, thereby avoiding merely technical complications that could otherwise arise.

2.2 Term Structure

Definition 2.8 Let Σ be a signature with sort structure Δ. The set of *well-sorted* $\mathcal{LC}$*-terms* for Σ is determined inductively by the following inference rules:

$$\frac{x \in Vars_A}{\Sigma \vdash x : A}\ (var) \qquad \frac{\Sigma \vdash X : A \quad \Sigma \vdash Y : B \quad \Delta \vdash B \sim \delta(A)}{\Sigma \vdash XY : \gamma(A)}\ (app)$$

$$\frac{[c :: A] \in \Sigma}{\Sigma \vdash c : A}\ (const) \qquad \frac{x \in Vars_B \quad \Sigma \vdash X : A}{\Sigma \vdash \lambda x.X : B \to A}\ (abs)$$

$$\frac{\Sigma \vdash X : A \quad \Delta \vdash \delta(A) \sim B}{\Sigma \vdash \lambda x_B.Xx : A}\ (\eta) \qquad \frac{\Sigma \vdash X : B \quad \Delta \vdash B \leq A}{\Sigma \vdash X : A}\ (weaken)$$

Let $\mathcal{LC}_A(\Sigma) = \{X \mid \Sigma \vdash X : A\}$ and $\mathcal{LC}(\Sigma) = \bigcup_{A \in \mathcal{S}} \mathcal{LC}_A(\Sigma)$. For any $X \in \mathcal{LC}(\Sigma)$ write $\mathcal{S}_\Sigma(X)$ for $\{A \in \mathcal{S} \mid X \in \mathcal{LC}_A(\Sigma)\}$. Since the inclusion ordering determined by any sort structure Δ is transitive, we need never follow one application of the rule (weaken) by another in constructing sort derivations for well-sorted $\mathcal{LC}$-terms (henceforth called $\mathcal{LC}(\Sigma)$*-terms*). We consider $\mathcal{LC}(\Sigma)$-terms which are identical up to renaming of (sorted) variables to be the same, and omit sort information whenever possible.

If Σ is a signature with sort system $\mathcal{S}$ and sort structure Δ, and if $\sim$ is the equivalence relation determined by Δ, then $\mathcal{LC}_A(\Sigma) = \mathcal{LC}_B(\Sigma)$ whenever $A \sim B$. Passing to the quotient signature Σ' with respect to $\sim$, *i.e.*, to the signature with sort system $\mathcal{S}'$ equal to $\mathcal{S}/\sim$ obtained by replacing sorts in $\mathcal{S}$ by canonical $\sim$-equivalence class representatives, we arrive at a signature whose equivalence relation is trivial and such that $\mathcal{LC}_A(\Sigma') = \mathcal{LC}_A(\Sigma)$ for all sorts A. We may therefore assume that $\leq$ is a partial ordering for all signatures in the remainder of this paper. We also assume that we have ridded our sort structures of redundant subsort declarations of the form $[A \leq A]$, and that whenever $\Delta \vdash B \leq A$ for a sort structure Δ, $length(B) \leq length(A)$ holds. The latter assumption is without loss of generality under a standard semantics, and implies that $length(B) \leq length(A)$ if $\Delta \vdash B \leq A$.

A routine induction on sort derivations establishes that signatures are subterm closed, *i.e.*, that each subterm of a well-sorted term is again well-sorted.

In any signature Σ, if $x \in Vars_A$, then x has least sort A in Σ. But because of constant overloading, not every term will necessarily have a unique least sort. For an arbitrary term X, however, if $\Sigma \vdash X : A$ and $\Sigma \vdash X : B$ then $\tau(A) = \tau(B)$. As a result, the fact that Σ has only finitely many sorts per type implies that, for $X \in \mathcal{LC}(\Sigma)$, the set of sorts $\mathcal{S}_\Sigma(X)$ is finite. It also follows that if we consider the forgetful functor to be the identity on typed constants, then it can be extended to an injection (but not necessarily a bijection) from $\mathcal{LC}(\Sigma)$ into $\mathcal{LC}$. And if Σ is a signature with empty sort structure and exactly one sort A such that $\tau(A) = \alpha$ for each $\alpha \in \mathcal{T}_0$, then $\mathcal{LC}(\Sigma)$ is isomorphic to the fragment of $\mathcal{LC}$ containing only the finitely many constants per type appearing in constant declarations in Σ.

To prove computability of sort assignment for $\mathcal{LC}(\Sigma)$, we extend the function $\mathcal{S}_\Sigma(\cdot)$ on $\mathcal{LC}(\Sigma)$ to all of $\mathcal{LC}$. For $X \in \mathcal{LC}$ and Σ a signature, define $\mathcal{S}_\Sigma(X) = \{\mathcal{S}_\Sigma(Y) \mid Y \in \mathcal{LC}(\Sigma) \text{ and } \overline{Y} \equiv X\}$. Then $X \in \mathcal{LC} \setminus \mathcal{LC}(\Sigma)$ iff $\mathcal{S}_\Sigma(X) = \emptyset$. If there exists a $Y \in \mathcal{LC}(\Sigma)$ with $\overline{Y} \equiv X$, then it is unique; in this case, we say that $X \in \mathcal{LC}$ is *well-sorted* with respect to Σ.

Theorem 2.9 *For $X \in \mathcal{LC}$ and any signature Σ, $\mathcal{S}_\Sigma(X)$ is effectively computable.*

Proof. We will later observe that η-reduction on $\mathcal{LC}(\Sigma)$ is sort-preserving, and, assuming this, we take X to be in η-normal form. Induction on the structure of X completes the proof. □

Corollary 2.10 *For $X \in \mathcal{LC}$ and any signature Σ, it is decidable whether or not X is well-sorted with respect to Σ.*

As promised, we can prove (by induction according to the various cases for the derivations of $\Sigma \vdash X : A$ and $\Sigma \vdash X : B$) that

Lemma 2.11 *If $\Sigma \vdash X : A$ and $\Sigma \vdash X : B$, then A Rdom B. That is, any signature Σ respects function domains.*

Lemma 2.11 guarantees that for any term X and any sorts $A, B \in \mathcal{S}_\Sigma(X)$ we must have $\delta(A) = \delta(B)$. This unique domain sort for X is called its *supporting sort* and is denoted $supp(X)$. At first glance, requiring signatures to respect function domains appears to be a grave restriction on the expressiveness of a calculus, but functional extensionality itself relies heavily on the notion of implicitly specified domains of functions, which unique supporting sorts syntactically capture. Indeed, in mathematics, functions are assumed to have unique (implicitly specified) domains, and must therefore be distinguished from restrictions to subdomains: functions f and g are the same only if $fa = ga$ for all a in the common (implicitly specified) domain of f and g.

2.3 Order-sorted Reduction

As per the above discussion, η-expansion of the term X_A to $\lambda x_B.Xx$, which corresponds to restricting the function denoted by X to the sort denoted by B, should only again yield the original function if B represents the domain of the function denoted by X. This restriction is embodied in the order-sorted η-rule.

Definition 2.12 Let Σ be any signature. The following order-sorted reductions are defined for $\mathcal{LC}(\Sigma)$-terms:

- $(\lambda x.X)Y \xrightarrow{\beta} X[x := Y]$, and
- $\lambda x_B.Xx \xrightarrow{\eta} X$ if $x_B \notin FV(X)$ and $B \equiv supp(X)$.

The first rule above, assumed to happen without free variable capture, is called *(order-sorted) β-reduction*; the second is called *(order-sorted) η-reduction*. Since order-sorted $\beta\eta$-reduction generalizes ordinary typed $\beta\eta$-reduction, we write $\xrightarrow{\beta\eta}$ for order-sorted $\beta\eta$-reduction as well as for its typed version.

It is important to our program that the fundamental operations of our calculus do not allow the formation of ill-sorted terms from well-sorted ones. This ensures that our unification algorithm never has to handle ill-sorted terms. In fact, if $X \xrightarrow{\beta}\!\!\!\!\rightarrow Y$, then $\mathcal{S}_\Sigma(X) \subseteq \mathcal{S}_\Sigma(Y)$. A similar although slightly stronger result holds for η-reduction: if $X \xrightarrow{\eta}\!\!\!\!\rightarrow Y$, then $S_\Sigma(X) = S_\Sigma(Y)$.

Order-sorted $\beta\eta$-reduction is convergent. Termination is a direct consequence of the corresponding well-known result for the simply typed lambda calculus, and weak confluence — and, in light of termination, therefore confluence — follows from weak confluence of $\beta\eta$-reduction on $\mathcal{LC}$ together with the fact that $X \xrightarrow{\beta\eta} Y$ implies $supp(X) \equiv supp(Y)$. It thus makes sense to refer to *the* order-sorted $\beta\eta$-normal form of an $\mathcal{LC}(\Sigma)$-term, and *the* order-sorted long (*i.e.*, η-expanded) β-normal form of X, denoted $l\beta nf(X)$.

3 Order-sorted Higher-order Unification

When considering unification in the simply typed lambda calculus, it is customary to work modulo η-equality. We explicitly keep track of order-sorted η-equality, since the interaction between extensionality and sorts can be unexpectedly subtle. Fix an arbitrary signature Σ for use throughout the remainder of this paper.

3.1 Systems and Substitutions

We will represent unification problems by equational systems comprising the pairs of $\mathcal{LC}(\Sigma)$-terms to be simultaneously unified, and use transformations of such systems as our main tool for solving the unification problems they represent.

A *pair* is a two-element multiset of $\mathcal{LC}(\Sigma)$-terms. A *system* is a finite set Γ of pairs. A pair is *η-trivial* (or simply *trivial*) if its elements are η-equal, and *Σ-valid* if its elements are $\beta\eta$-equal; a system is *Σ-valid* if each of its pairs is Σ-valid. As usual, we write $\Gamma, \langle X, Y\rangle$ instead of $\Gamma \cup \{\langle X, Y\rangle\}$, but since Γ may or may not also contain $\langle X, Y\rangle$, such a decomposition is ambiguous. We use the notation $\Gamma; \langle X, Y\rangle$ to abbreviate $\Gamma \cup \{\langle X, Y\rangle\}$ when $\langle X, Y\rangle$ is *not* a pair in Γ. A pair $\langle X, Y\rangle$ is *solved in Γ* if it is either trivial, or for some $x \in Vars_A$, $X \xrightarrow{\eta}\!\!\!\!\rightarrow x$, $A \in \mathcal{S}_\Sigma(Y)$ and there are no occurrences of x in Γ other than the one indicated. In this case, x is said to be *solved in Γ*. If each pair in Γ is solved in Γ, then Γ is a *solved system*.

A *substitution* is a finitely supported map from variables to $\mathcal{LC}(\Sigma)$; a substitution θ induces a mapping on terms, which we also denote by θ. We write substitution application as juxtaposition, so that θX is the application of the substitution θ to the term X, and by $D(\theta)$ and $I(\theta)$ we denote the set of variables in the domain of θ and the set of variables introduced by θ, respectively. A substitution θ is *well-sorted* if for every $x \in Vars_A$, $A \in \mathcal{S}_\Sigma(\theta x)$. It follows that if $X \in \mathcal{LC}_A(\Sigma)$ and θ is well-sorted, then $\theta X \in \mathcal{LC}_A(\Sigma)$ as well. That the set of well-sorted substitutions is closed under composition is not hard to prove.

We can extend equalities on $\mathcal{LC}(\Sigma)$ to (well-sorted) substitutions in the usual manner: Let $=_*$ be an equational theory on $\mathcal{LC}(\Sigma)$, W be a set of variables, and θ and θ' be substitutions. Then $\theta =_* \theta'[W]$ means that for every variable in $x \in W$, $\theta x =_* \theta' x$. The subsumption relation $\theta' \leq_* \theta[W]$ holds provided there exists a substitution ρ such that $\theta =_* \rho\theta'[W]$. If W is the set of all variables, we drop the notation "$[W]$." If $=_*$ is the empty equational theory we write "$\equiv$" and "$\leq$" for the induced equality and subsumption ordering on substitutions.

We can extend substitutions on $\mathcal{LC}(\Sigma)$ to mappings on systems $\Gamma \equiv \{\langle X_i, Y_i\rangle \mid i \leq n\}$ by defining $\sigma\Gamma$ to be the system $\{\langle \sigma X_i, \sigma Y_i\rangle \mid i \leq n\}$. The normal form $l\beta nf(\Gamma)$, all of whose unsolved pairs comprise terms in long β-normal form, is defined si milarly. If all terms in the unsolved pairs of Γ are in long β-normal form, we say that Γ is *in long β-normal form*. We write $FV(X)$ for the set of free variables occurring in the $\mathcal{LC}(\Sigma)$-term X and $FV(\Gamma)$ for the free variables occurring in the terms in the system Γ.

A well-sorted substitution θ is a *Σ-unifier* of a system Γ if $\theta\Gamma$ is Σ-valid. If σ is a Σ-unifier of Γ with the properties that $D(\sigma) \subseteq FV(\Gamma)$ and that for any Σ-unifier θ of Γ, $\sigma \leq_{\beta\eta} \theta$ holds, then σ is said to be a *most general Σ-unifier* of Γ. A system Γ is *Σ-unifiable* if there exists some Σ-unifier of Γ. An idempotent well-sorted substitution θ is a *normalized Σ-unifier* of a system Γ if *i)* $D(\theta) \subseteq FV(\Gamma)$, *ii)* θ is a Σ-unifier of Γ, and *iii)* for all unsolved variables x in Γ, θx is in long β-normal form. Write $U_\Sigma(\Gamma)$ for the set of all normalized Σ-unifiers of Γ. It is clear that every well-sorted substitution θ is $\beta\eta$-equal to a well-sorted substitution θ' with $D(\theta) = D(\theta')$ and $\theta' x$ in long β-normal form for each $x \in D(\theta)$. Such a substitution θ' is said to be *in long β-normal form*. Thus for any Σ-unifier θ of a system Γ, there exists a $\theta' \in U_\Sigma(\Gamma)$ such that $\theta' =_{\beta\eta} \theta[FV(\Gamma)]$. In particular, every Σ-unifiable system has a normalized Σ-unifier. For technical reasons, normalized Σ-unifiers will be important in what follows. Note that we relax the standard requirement that normalized substitutions map all variables to normal forms, and allow solved variables to be bound arbitrarily. This is justified in Lemma 3.2 below.

The remainder of this section explores the relationship between systems and their unifiers. If Γ is a solved system whose non-trivial pairs are $\langle X_1, Y_1\rangle, ..., \langle X_n, Y_n\rangle$ with $X_i \xrightarrow{\eta}\!\!\!\rightarrow x_i$ for $i = 1, ..., n$, then these pairs determine an idempotent well-sorted substitution $\sigma_\Gamma = \{x_1 \mapsto Y_1, ..., x_n \mapsto Y_n\}$, although such a pair $\langle X, Y\rangle$ with $X \xrightarrow{\eta}\!\!\!\rightarrow x \in Vars_A$ and $Y \xrightarrow{\eta}\!\!\!\rightarrow y \in Vars_A$ requires a choice as to which of x and y is to be in the domain of the substitution. We assume that a uniform way exists for making this choice, and so refer to *the* well-sorted substitution determined by a solved system. Conversely, idempotent well-sorted substitutions can be represented by solved systems without trivial pairs. If σ is such a substitution, write $[\sigma]$ for any solved system which represents it. Any system Γ can be written as $\Gamma'; [\sigma]$ where $[\sigma]$ is the set of solved pairs in Γ. We call $[\sigma]$ the *solved part* of Γ.

Transformation-based unification methods attempt to reduce systems to be unified to solved systems which represent their unifiers. The fundamental connection between solved systems and Σ-unifiers is that solved systems represent their own solutions:

Lemma 3.1 *If $\Gamma \equiv \langle X_1, Y_1\rangle, ..., \langle X_n, Y_n\rangle$ is a solved system, then σ_Γ is a most general Σ-unifier for Γ. In fact, for any Σ-unifier θ of Γ, $\theta =_{\beta\eta} \theta\sigma_\Gamma$.*

In general, however, a system Γ will not have a single most general Σ-unifier. The next lemma shows that we need not be concerned with solved pairs when computing Σ-unifiers. This is consistent with the intuition that the solved part of a system is merely a record of an answer substitution being constructed.

Lemma 3.2 *Suppose Γ is a Σ-unifiable system with solved part $[\sigma]$ and unsolved part Γ'. If θ is a Σ-unifier of Γ, then for every Σ-unifier ρ of Γ' such that $D(\rho) \subseteq FV(\Gamma')$ and $\rho \leq_{\beta\eta} \theta[FV(\Gamma')]$, $\rho\sigma$ is a Σ-unifier of Γ and $\rho\sigma \leq_{\beta\eta} \theta[FV(\Gamma)]$.*

3.2 The Unification Algorithm

One of the key steps for sorted higher-order unification is solving the following problem: given a term $X \equiv \lambda x_1...x_k.hU_1...U_n \in \mathcal{LC}_A(\Sigma)$ in long β-normal form, find a term $G \in \mathcal{LC}_A(\Sigma)$ with head h which can be instantiated to yield X. This is a generalization of a problem in $\mathcal{LC}$ which Huet ([Hue75]) resolved by describing a set of *partial bindings* in long β-normal form capable of approximating any $\mathcal{LC}$-term by instantiation. While Huet-style partial bindings suffice for approximating arbitrary $\mathcal{LC}(\Sigma)$-terms — although not necessarily with bindings of the appropriate sorts — in our setting, we cannot require that partial bindings be η-expanded without sacrificing completeness of our Σ-unification algorithm (see Example 3.6). Below, a variable will be called *fresh* if it does not appear in any term in the current context.

Definition 3.3 If h is an atom such that either $h \in Vars_C$ or $[h :: C]$ is a constant declaration in Σ, then a *partial binding of sort A for head h* is any term of the form $G \equiv \lambda y_1...y_l.hV_1...V_m$, where *i)* $l = length(A)$, *ii)* $m = l + length(\tau(C)) - length(\tau(A)) \geq 0$, *iii)* $\Delta \vdash \gamma^m(C) \leq \gamma^l(A)$, *iv)* $y_j \in Vars_{\delta^j(A)}$ for $j = 1, ..., l$, and *v)* $V_i \equiv z_i y_1...y_l$ for $1 \leq i \leq m$, where $z_i \in Vars_{\delta^1(A) \to ... \to \delta^l(A) \to \delta^i(C)}$ is fresh.

For a given sort A and head h partial bindings need not exist due to conditions *ii)* and *iii)* of Definition 3.3, but because signatures respect function domains, when they do exist they are unique up to renaming of the variables z_i. If Σ is a signature without functional base sorts, then the partial bindings are η-expanded; in particular, if Σ is a signature with exactly one sort per (base) type, then the partial bindings are precisely those obtained for $\mathcal{LC}$. Writing $\mathcal{G}^h_A(\Sigma)$ for the set of partial bindings of sort A for head h, the fact that $\Sigma \vdash G : A$ for $G \in \mathcal{G}^h_A(\Sigma)$ justifies our terminology.

Call a partial binding $G \equiv \lambda y_1...y_l.hV_1...V_m$ a j^{th} *projection binding* if $h \equiv y_j$ and an *imitation binding* if $h \in FV(G) \cup \mathcal{C}$. The following transformations on which our algorithm is based are adapted from those of [Sny91].

Definition 3.4 The set $\Sigma\mathcal{T}$ comprises the following transformations on systems in long β-normal form (it is possible that $k = 0$ below).

- DECOMPOSE: For any atom h,

$$\Gamma; \langle \lambda x_1...x_k.hX_1...X_n, \lambda x_1...x_k.hU_1...U_n\rangle \Longrightarrow$$
$$\Gamma, \langle \lambda x_1...x_k.X_1, \lambda x_1...x_k.U_1\rangle, ..., \langle \lambda x_1...x_k.X_n, \lambda x_1...x_k.U_n\rangle.$$

- ELIMINATE: If $x \in Vars_A$, $x \notin \{x_1, ..., x_k\}$, $x \notin FV(\lambda x_1...x_k.X)$, and $\sigma = \{x \mapsto \lambda x_1...x_k.X\}$ is well-sorted, then

$$\Gamma; \langle \lambda x_1...x_k.xx_1...x_k, \lambda x_1...x_k.X\rangle \Longrightarrow \langle x, \lambda x_1...x_k.X\rangle, \sigma\Gamma.$$

- IMITATE: If $x \in Vars_A$, $h \in \mathcal{C}$ or $h \in FV(\lambda x_1...x_k.hU_1...U_m)$, $h \not\equiv x$, and $G \in \mathcal{G}^h_A(\Sigma)$ is an imitation binding, then

$$\Gamma; \langle \lambda x_1...x_k.xX_1...X_n, \lambda x_1...x_k.hU_1...U_m\rangle \Longrightarrow$$
$$\Gamma, \langle x, G\rangle, \langle \lambda x_1...x_k.xX_1...X_n, \lambda x_1...x_k.hU_1...U_m\rangle.$$

- j-PROJECT: If $x \in Vars_A$, h is a (possibly bound) atom and $G \in \mathcal{G}^h_A(\Sigma)$ is a j^{th} projection binding for some $j \in \{1, ..., n\}$ such that $head(X_j) \in \mathcal{C}$ implies $head(X_j) \equiv h$, then

$$\Gamma; \langle \lambda x_1...x_k.xX_1...X_n, \lambda x_1...x_k.hU_1...U_m\rangle \Longrightarrow$$
$$\Gamma, \langle x, G\rangle, \langle \lambda x_1...x_k.xX_1...X_n, \lambda x_1...x_k.hU_1...U_m\rangle.$$

- GUESS: If h is any atom, and x and y are free variables in $Vars_A$ and $Vars_B$, respectively, both distinct from h, and $G \in \mathcal{G}^h_A(\Sigma)$, then

$$\Gamma; \langle \lambda x_1...x_k.xX_1...X_n, \lambda x_1...x_k.yU_1...U_m\rangle \Longrightarrow$$
$$\Gamma, \langle x, G\rangle, \langle \lambda x_1...x_k.xX_1...X_n, \lambda x_1...x_k.yU_1...U_m\rangle.$$

As part of the transformations IMITATE, j-PROJECT, and GUESS, we immediately apply ELIMINATE to the new pair $\langle x, G\rangle$.

Our sort mechanism insures that applications of the transformations are such that all terms involved are well-sorted. We adopt the convention that no transformations may be done out of solved or trivial pairs, which accords with the intuition that the solved pairs in a system are merely recording an answer substitution as it is incrementally built up.

We emphasize that there is no deletion of trivial pairs in this presentation. This guarantees that if $\Gamma \Longrightarrow \Gamma'$, then $FV(\Gamma) \subseteq FV(\Gamma')$, so that when a fresh variable is chosen during a computation it is guaranteed to be new to the entire computation. This prevents us from having to manipulate the "protected sets of variables" typically found in completeness proofs in the literature, and respects the fundamental idea behind the use of transformations for describing algorithms, namely that the logic of the problem being considered can be abstracted from implementational issues.

Definition 3.5 The non-deterministic algorithm $\Sigma\mathcal{U}$ is the process of repeatedly

1. reducing all terms of the unsolved pairs in the system to long β-normal form and then applying some transformation in $\Sigma\mathcal{T}$ to an unsolved pair, and

2. returning a most general Σ-unifier if at any point in the computation the system becomes solved.

The choice of pair upon which Algorithm $\Sigma\mathcal{U}$ is to act, and the rule from $\Sigma\mathcal{T}$ to be applied, are non-deterministic. We illustrate use of Algorithm $\Sigma\mathcal{U}$:

Example 3.6 Let $[b :: \delta(A)]$ and $[c :: A]$ comprise the set of constant declarations in a signature Σ with a functional base sort A. Let $f \in Vars_A$, $x \in Vars_{\delta(A)}$, and $w \in Vars_{A\rightarrow\delta(A)}$, and consider the Σ-unifiable long β-normal form system $\Gamma \equiv \langle fx, cb\rangle, \langle wc, b\rangle$. Applying IMITATE with partial binding c to the first pair of Γ yields $\langle f, c\rangle, \langle cx, cb\rangle, \langle wc, b\rangle$. An application of DECOMPOSE results in $\langle f, c\rangle, \langle x, b\rangle, \langle wc, b\rangle$, and an application of IMITATE with binding $\lambda y.b$ for $y \in Vars_A$ to the third pair, followed by some β-reductions give the solved system $\Gamma' \equiv \langle f, c\rangle, \langle x, b\rangle, \langle w, \lambda y.b\rangle, \langle b, b\rangle$. We extract the well-sorted substitution $\sigma = \{f \mapsto c, x \mapsto b, w \mapsto \lambda y.b\}$, and anticipating Theorem 3.8, conclude that σ is a Σ-unifier of Γ' and hence of Γ. If we instead allow only η-expanded partial bindings, then the only possible IMITATE step binds f to a term of the form $\lambda y.c(zy)$ for a variable y and a fresh variable z of appropriate sorts. But then ELIMINATE cannot be performed on the pair $\langle f, \lambda y.c(zy)\rangle$ (as is required to complete the IMITATE step), since $\Sigma \not\vdash \lambda y.c(zy) : A$.

While unification in $\mathcal{LC}(\Sigma)$ is apparently more delicate than unification in $\mathcal{LC}$, the extra care pays off when sort information disallows certain undesirable unifications that would be possible in an unsorted calculus.

Example 3.7 Let Σ be a signature with base sorts D, I, and R, where the non-functional sort R denotes the real numbers, and the functional sorts D and I denote the strictly decreasing and strictly increasing functions on the reals, respectively. Suppose further that $\delta(D) = \delta(I) = R$ and $\gamma(D) = \gamma(I) = R$. Finally, let $[n :: D \rightarrow I]$ and $[4 :: R]$ comprise the set of constant declarations of Σ, where n denotes the "negation functor" mapping each function F to $-F$, and 4 denotes the real number four.

Let $x \in Vars_R$, $f \in Vars_I$, and $g \in Vars_D$, and consider the unification problem given by the pairs $\langle f4, ngx\rangle, \langle gx, 4\rangle$. It is not hard to see that an application of IMITATE to the pair $\langle f4, ngx\rangle$ is the only possibility for computation. Letting z be fresh from $Vars_D$, we have that $nz \in \mathcal{G}^n_I(\Sigma)$, and so can apply IMITATE with this binding for f to get $\langle f, nz\rangle, \langle nz4, ngx\rangle, \langle gx, 4\rangle$. Similarly, we conclude that only DECOMPOSE applies here, resulting in $\langle f, nz\rangle, \langle z, g\rangle, \langle x, 4\rangle, \langle gx, 4\rangle$. Two applications of ELIMINATE yield $\langle f, ng\rangle, \langle z, g\rangle, \langle x, 4\rangle, \langle g4, 4\rangle$, all of whose pairs, save the last — unsolvable — one, are solved. The only alternative to eliminating z above is applying GUESS to $\langle z, g\rangle$ in the second derived system, but this makes no progress toward a solution. Anticipating Theorem 3.13, we conclude that the original system is unsolvable, in accordance with the facts that neither the identity function nor the function which is constantly four is strictly decreasing.

Of course, if D were to denote the (not strictly) decreasing real-valued functions on the reals, then we would expect $\langle g4, 4\rangle$ to be solvable by binding g to $\lambda y.4$. A calculus allowing arbitrary term declarations finds a middle road between the typed calculus, which permits too many bindings, and one supporting only constant declarations, which permits too few: declaring $\lambda y.4$ to be of sort D when $y \in Vars_R$, Γ yields precisely the desired solutions.

3.3 Soundness and Completeness of the Algorithm

The proof that our transformations are sound is not appreciably different from the proof for the corresponding transformations for unification in $\mathcal{LC}$.

Theorem 3.8 *(Soundness) If $\Gamma \Longrightarrow \Gamma'$, then for any well-sorted substitution θ, θ is a Σ-unifier of Γ if it is a Σ-unifier of Γ'.*

Thus if Algorithm $\Sigma\mathcal{U}$ is run on initial system Γ and returns a well-sorted substitution θ, then θ is indeed a Σ-unifier of Γ. Our main result (Theorem 3.13) is a converse. We require a few technical lemmas, the first of which is proved by induction on the derivation of $\Sigma \vdash Y : A$.

Lemma 3.9 *If $Y \equiv \lambda x_1...x_p.hU_1...U_q \in \mathcal{LC}_A(\Sigma)$ is in $\beta\eta$-normal form, then either $h \in Vars_C$ or $[h :: C]$ is a constant declaration in Σ for some sort C such that $length(A) + length(\tau(C)) - length(\tau(A)) \geq 0$ and $\Delta \vdash \gamma^q(C) \leq \gamma^p(A)$.*

Lemma 3.10 *If $X \equiv \lambda x_1...x_k.hU_1...U_n \in \mathcal{LC}_A(\Sigma)$ is in long β-normal form, then there exist a partial binding $G \in \mathcal{G}^h_A(\Sigma)$ and a well-sorted substitution ρ in long β-normal form such that $D(\rho)$ is precisely the set of fresh variables in G, ρz has smaller depth than X for each $z \in D(\rho)$, and $\rho G =_{\beta\eta} X$.*

Proof. Let $Y \equiv \lambda x_1...x_p.hU'_1...U'_q$ be the $\beta\eta$-normal form of X, where $U_i \xrightarrow{\eta} U'_i$ for $i = 1,...,q$, $p \leq k$, and $n = q + (k - p)$. Let C be the sort whose existence is guaranteed by Lemma 3.9, $m = length(A) + length(\tau(C)) - length(\tau(A)) \geq 0$, and $G \equiv \lambda x_1...x_l.hV_1...V_m \in \mathcal{G}^h_A(\Sigma)$, where $V_i = z_i x_1...x_l$ for fresh variables z_i, $i = 1,...,m$. Then $l \leq length(\tau(A)) = k$ and $n = length(\tau(C))$, so that $m = l + n - k = l + q - p$. Since $\Sigma \vdash Y : A$, we must have $p \leq l \leq k$. The substitution ρ mapping z_i to $\lambda x_1...x_l.U_i$ for $i = 1,...,q$, and z_i to $\lambda x_1...x_l.x_{p-q+i}$ for $i = q+1,...,m$ is well-sorted, has domain consisting precisely of the set of fresh variables in G, and has the property that ρz has smaller depth than X for each $z \in D(\rho)$. It is well-defined because $m - q = l - p \geq 0$, and indeed $\rho(G) =_\beta \lambda x_1...x_l.hU_1...U_q x_{p+1}...x_l =_\eta \lambda x_1...x_p.hU'_1...U'_q =_\eta X$. □

Note that with the Huet-style partial bindings, it would not necessarily be possible to find G of sort A and a substitution ρ as required:

Example 3.11 If Σ is a signature with a constant declaration $[c :: A]$ for a functional base sort A, then $\Sigma \vdash \lambda x.cx : A$ using (const) followed by an application of (η). Any Huet-style partial binding that might approximate the long β-normal form $\lambda x.cx$ must be of the form $\lambda x.c(zx)$ where z is a fresh variable of an appropriate sort, but there is no derivation of $\Sigma \vdash \lambda x.c(zx) : A$. Under our definition, however, $G \equiv c$ is itself a partial binding of sort A for head h, and ρ can be taken to be the identity substitution.

The measure μ defined by $\mu(\Gamma, \theta) = \langle \mu_1(\Gamma, \theta), \mu_2(\Gamma) \rangle$, where $\mu_1(\Gamma, \theta)$ is the multiset of the depths of the θ-bindings of unsolved variables in Γ which are also in $D(\theta)$, and $\mu_2(\Gamma)$ is the multiset of depths of terms in Γ, will provide the basis for proving termination of Algorithm $\Sigma\mathcal{U}$.

Lemma 3.12 *Let $\theta \in U_\Sigma(\Gamma)$ and let $\langle X, Y\rangle$ be an unsolved pair in a system Γ in long β-normal form. Then there exist a system Γ' and a substitution θ' such that $\Gamma \Longrightarrow \Gamma'$, $\theta \equiv \theta'[FV(\Gamma)]$, $\theta' \in U_\Sigma(\Gamma')$, and $\mu(\Gamma', \theta') < \mu(\Gamma, \theta)$.*

Proof. If $head(X) \equiv head(Y) \notin D(\theta)$, then since $\langle X, Y\rangle$ is not trivial, DECOMPOSE must apply and we must have $\theta \in U_\Sigma(\Gamma')$. Also, $\mu(\Gamma', \theta) < \mu(\Gamma, \theta)$ since $\mu_1(\Gamma', \theta) \leq \mu_1(\Gamma, \theta)$ and $\mu_2(\Gamma') < \mu_2(\Gamma)$.

Otherwise, at least one of X and Y has an unsolved variable $x \in D(\theta) \cap Vars_A$ of Γ as its head; assume X does. Then since θ is well-sorted, $\Sigma \vdash \theta x : A$, and θx is in long β-normal form since θ is normalized. Suppose $\theta x \equiv \lambda x_1 ... x_k . h U_1 ... U_n$. By Lemma 3.10, there exist $G \in \mathcal{G}^h_A(\Sigma)$ and a well-sorted substitution ρ in long β-normal form satisfying the conclusions of that lemma. Thus if $head(Y) \notin D(\theta)$ and $h \equiv head(Y)$, then IMITATE applies, if $head(Y) \notin D(\theta)$ and $h \not\equiv head(Y)$, then j-PROJECT applies for some j, and if $head(Y) \in D(\theta)$, then GUESS applies. Taking $\theta' = \theta \cup \rho$, we have that $\theta \equiv \theta'[FV(\Gamma)]$, $\theta \in U_\Sigma(\Gamma')$ since $\theta \in U_\Sigma(\Gamma)$ and ρ is in long β-normal form, and $D(\rho)$ is exactly the set of fresh variables in G. Moreover, $\mu_1(\Gamma', \theta') < \mu_1(\Gamma, \theta)$: x is removed from the set of unsolved variables in Γ which appear in $D(\theta)$, and is replaced by the set of fresh variables of G, but for each such variable z, $\theta' z \equiv \rho z$ is smaller than θx. Thus $\mu(\Gamma', \theta') < \mu(\Gamma, \theta)$.

Observe that if $head(X) \equiv head(Y) \notin D(\theta)$ does not hold, but $X \xrightarrow{\eta}\!\!\!\!\!\rightarrow x \in Vars_A$, x is not free in Y, and $\Sigma \vdash Y : A$, then ELIMINATE applies. In this case, we can take θ' to be θ by noting that $\mu_1(\Gamma', \theta) < \mu_1(\Gamma, \theta)$. □

The proof of Lemma 3.12 shows that it is possible to restrict DECOMPOSE to apply only when $head(X) \equiv head(Y) \notin D(\theta)$, although there is no way of encoding this restriction into the transformations. If we call a transformation prescribed by Lemma 3.12 a *μ-prescribed* transformation, then each application of a μ-prescribed transformation decreases the well-founded measure μ. The previous lemma guarantees that if Γ is a Σ-unifiable system in long β-normal form to which no μ-prescribed transformation in $\Sigma\mathcal{T}$ applies, then Γ is solved.

Theorem 3.13 *Let θ be a Σ-unifier of Γ. Then there exists a computation of Algorithm $\Sigma\mathcal{U}$ on Γ producing a Σ-unifier σ of Γ such that $\sigma \leq_{\beta\eta} \theta[FV(\Gamma)]$.*

Proof. Since every Σ-unifier of Γ is pointwise $\beta\eta$-equal on $FV(\Gamma)$ to some $\theta' \in U_\Sigma(\Gamma)$, we prove the theorem under the added hypothesis that $\theta \in U_\Sigma(\Gamma)$.

If Γ is not in long β-normal form, then perform reductions until a system in long β-normal form results. Note that if θ Σ-unifies Γ, then θ also Σ-unifies $l\beta nf(\Gamma)$, and that this reduction is a $\Sigma\mathcal{U}$ step. We may therefore assume without loss of generality in the remainder of this proof that Γ is in long β-normal form. We induct on the length of the longest sequence of μ-prescribed sequence of transformations available out of Γ.

If no μ-prescribed transformation from $\Sigma\mathcal{T}$ applies to Γ, then Γ is solved so we may return a most general Σ-unifier σ of Γ whose existence is guaranteed by Lemma 3.1. This action is a step of Algorithm $\Sigma\mathcal{U}$, and $\sigma \leq_{\beta\eta} \theta$. If some μ-prescribed transformation from $\Sigma\mathcal{T}$ applies to Γ yielding a system Γ' and a substitution θ' satisfying the conclusion of Lemma 3.12, then applying this transformation is a $\Sigma\mathcal{U}$ step. By the induction hypothesis, there is a computation of $\Sigma\mathcal{U}$ on Γ' producing a Σ-unifier δ of Γ' such that $\delta \leq_{\beta\eta} \theta'[FV(\Gamma')]$. It follows from Lemma 3.8 that δ is a

Σ-unifier of Γ, and since $FV(\Gamma) \subseteq FV(\Gamma')$, $\delta \leq_{\beta\eta} \theta'[FV(\Gamma)]$. But $\theta' \equiv \theta[FV(\Gamma)]$, so that $\delta \leq_{\beta\eta} \theta[FV(\Gamma)]$. □

Since we have not made any assumption about the order in which transformations from $\Sigma\mathcal{T}$ are performed, and since any application of ELIMINATE to a system reduces the measure μ, we infer that the strategy of eager variable elimination is complete for unification in our calculus. It is unknown whether eager variable elimination is complete for an arbitrary calculus and equational theory, even if both are first-order.

References

[BL90] K. B. Bruce and G. Longo. A Modest Model of Records, Inheritance, and Bounded Quantification. *Information and Computation* 87, pp. 196 – 240, 1990.

[Car88] L. Cardelli. A Semantics of Multiple Inheritance. *Information and Computation* 76, pp. 138 – 164, 1988.

[CG91] P.-L. Curien and G. Ghelli. Subtyping + Extensionality: Confluence of $\beta\eta$top Reduction in $F_{\leq}$. In *Proc. TACS '91*, Springer-Verlag LNCS 526, pp. 731 – 749, 1991.

[Coh89] A. G. Cohn. Taxonomic Reasoning with Many-sorted Logics. *Artificial Intelligence Review* 3, pp. 89 – 128, 1989.

[Hue72] G. Huet. Constrained Resolution: A Complete Method for Higher Order Logic. Dissertation, Case Western Reserve University, 1972.

[Hue75] G. Huet. A Unification Algorithm for Typed λ-Calculus. *Theoretical Computer Science* 1, pp. 27 – 57, 1975.

[JK93] P. Johann and M. Kohlhase. Unification in an Extensional Lambda Calculus with Ordered Function Sorts and Constant Overloading. Technical Report SR-93-14, Universität des Saarlandes, 1993.

[Koh92] M. Kohlhase. An Order-sorted Version of Type Theory. In *Proc. LPAR '92*, Springer-Verlag LNAI 624, pp. 421 – 432, 1992.

[NQ92] T. Nipkow and Z. Qian. Reduction and Unification in Lambda Calculi with Subtypes. In *Proc. CADE '92*, Springer-Verlag LNAI 607, pp. 66 – 78, 1992.

[Pfe92] F. Pfenning. Intersection Types for a Logical Framework. POP-Report, Carnegie-Mellon University, 1992.

[Pie91] B. C. Pierce. Programming with Intersection Types and Bounded Polymorphism. Dissertation, Carnegie Mellon University, 1991.

[Qia90] Z. Qian. Higher-order Order-sorted Algebras. In *Proc. Algebraic & Logic Programming '90*, Springer-Verlag LNCS 463, pp. 86 – 100, 1990.

[Sch89] M. Schmidt-Schauß. Computational Aspects of an Order-sorted Logic with Term Declarations. Springer-Verlag LNAI 395, 1989.

[Sny91] W. Snyder. A Proof Theory for General Unification. Birkhäuser Boston, 1991.

[Wal88] C. Walther. Many-sorted Unification. *Journal of the ACM* 35, pp. 1 – 17, 1988.