A Combinator-based Order-sorted Higher-order Unification Algorithm

Patricia Johann[*]
Fachbereich Informatik
Universität des Saarlandes
66123 Saarbrücken, Germany
pjohann@cs.uni-sb.de

Abstract

This paper develops a sound and complete transformation-based algorithm for unification in an extensional order-sorted combinatory logic supporting constant overloading and a higher-order sort concept. Appropriate notions of order-sorted weak equality and extensionality — reflecting order-sorted $\beta\eta$-equality in the corresponding lambda calculus given by Johann and Kohlhase — are defined, and the typed combinator-based higher-order unification techniques of Dougherty are modified to accommodate unification with respect to the theory they generate. The algorithm presented here can thus be viewed as a combinatory logic counterpart to that of Johann and Kohlhase, as well as a refinement of that of Dougherty, and provides evidence that combinatory logic is well-suited to serve as a framework for incorporating order-sorted higher-order reasoning into deduction systems aiming to capitalize on both the expressiveness of extensional higher-order logic and the efficiency of order-sorted calculi.

1 Introduction

Despite intensive investigation of equational reasoning (see, for example, the surveys [DJ90], [Klo91], [Pla93]) and the existence of powerful first-order deduction systems ([OS89], [Sti90], [LSBB92], [Lus92]), the inherently higher-order nature of many problems whose solutions one would like to deduce automatically has sparked a growing interest in higher-order deduction ([ALMP84], [Gor85], [Pau90], [Mil91]).

On the other hand, the fact that human reasoning naturally assumes an intrinsically structured universe, in which one typically wants to make assertions about every object in a certain class rather than about every object in the entire domain of discourse, often aids us in efficiently drawing logical conclusions. Making use of taxonomic distinctions in an automated deduction setting requires capturing the meta-level knowledge associated with reasoning in a structured universe by means of a purely syntactic calculus of formal reasoning. The incorporation of sort information into deduction calculi, the formal

[*]On leave from the Department of Mathematics and Computer Science, Hobart and William Smith Colleges, Geneva, NY 14456. This material is based upon work supported by the National Science Foundation under Grant No. INT-9224443.

mechanism for accomplishing this task, has been seen to dramatically reduce the search space associated with deduction in first-order logic ([Obe62], [Wal87], [Wal88], [Coh89], [Sch89]). Once employed, for example, to express inclusion relations among various classes of objects, or to record the fact that certain functions always return values in certain of those classes, sort information can be used to detect inferences which violate the constraints it imposes. Sort information is in fact now considered so crucial to the efficient automation of logical deduction that it is "common place [*sic*] to assert that without it no realistic applications of a deduction system are possible" ([OS89]).

In this light, the investigation of sorted higher-order calculi, which date back to Herbrand ([Her71]) and boast both the expressiveness of typed higher-order logic and the efficiency of sorted calculi, is only natural. The fact that type information can be regarded as coding very coarse taxonomic distinctions between disjoint classes of objects — so that sorts and ordering relations on them merely refine an already present structure — perhaps makes it even more so. But current sorted deduction systems typically make rather limited use of sort information, employing it primarily to determine which terms may, as governed by the constraints it imposes, be substituted for which variables in a deduction step. For this reason, sort information is built into deduction systems primarily by means of (pre-)unification algorithms rich enough to accommodate the relevant sorted calculi.

To date, all sorted higher-order calculi intended for use in deduction systems have been developed in terms of the lambda calculus ([NQ92], [Pfe92], [JK93], [KP93]), their unification algorithms being adaptations of Huet's classical algorithm for unification of simply typed lambda terms ([Hue73], [Hue75]). But recent inroads using a formulation of typed higher-order unification problems in terms of combinatory logic ([Dou93]), together with the aforementioned successes in order-sorted equational unification, suggest that an algebraic approach to unification in sorted higher-order calculi is also feasible. Indeed, this paper develops a sound and complete transformation-based algorithm for unification modulo an extensional order-sorted combinatory logic supporting constant overloading and a higher-order sort concept. This order-sorted higher-order unification algorithm may be viewed simultaneously as both a combinatory logic counterpart to the order-sorted lambda calculus-based algorithm of [JK93], and a refinement of the typed combinator-based algorithm of [Dou93]. Its development provides further support for a thesis advanced in [Joh91], [DJ92], and [Dou93], namely that combinatory logic can provide a computational framework for deduction in higher-order logic and its more expressive extensions.

The main features of this paper are as follows. Section 2 begins with a description of the sorted signatures over which our terms are built, as well as of the combinatory logic in which we are interested. We capitalize on the fact that functions are explicit objects of higher-order logic by allowing classes of functions defined by domains and codomains themselves to be divided into subclasses. Our signatures, precisely those of [JK93], thus support functional base sorts, *i.e.*, base sorts denoting classes of functions, in addition to base sorts denoting classes of individuals. Syntactically, each sort A comes with a type $\tau(A)$, a codomain sort $\gamma(A)$, and — if functional — 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 under 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 indeed a function with domain $\delta(A)$ and codomain $\gamma(A)$, a functional sort A must always be a subsort of the sort $\delta(A) \rightarrow \gamma(A)$.

The calculus presented here supports constructs for restricting the ranges of variables to, and assigning constants — including the various redex constants I, K, and S — membership in, certain classes of objects. Depending on the partial order induced on the sorts, certain classes of terms built from these atoms by function application then become the objects of study — the partial order restricts the models of the calculus so that terms must meet certain conditions to denote meaningful objects, *i.e.*, to be well-sorted. For example, the application of the functional term M to the argument term N is allowed only if there exist sorts A and B such that M has sort A, N has sort B, and B is a subsort of $\delta(A)$. The sort of the application term MN is defined to be $\gamma(A)$.

Next, notions of weak reduction and extensional equality which appropriately generalize the corresponding notions in typed combinatory logic, are defined. Together these generate the equality induced on well-sorted combinatory logic terms by sorted $\beta\eta$-equality, as originally defined on their lambda calculus counterparts in [JK93]. Despite the fact that there is no known convergent rewriting relation on *terms* generating this induced equality, our combinatory logic framework allows us to define an "almost convergent" reduction relation on *systems* which does indeed capture it. This reduction relation is then lifted to transformations on systems serving as the basis of our unification algorithm. The unification transformations given in Section 3 include one for narrowing at the heads of terms, one accommodating sorted extensionality, and a generalized decomposition transformation designed to handle paramodulating into a head variable (thereby changing the sort of a term, since the sorts of combinatory logic terms are determined by the sorts of their heads). Because systems are normalized with respect to the reduction relation before being submitted to the unification transformations, our algorithm can be seen as a "normalized narrowing" algorithm. It is proved complete at the end of Section 3, and a discussion follows in Section 4.

By using essentially equational, rather than higher-order methods, in our computations, we avoid some of the technical difficulties which arise in the treatment of order-sorted higher-order unification in [JK93]. In particular, our terms contain no bound variables and need not be kept in η-expanded form. But more importantly, the subtle interaction between η-equality and functional base sorts which requires much attention in [JK93] is circumvented entirely, since extensionality is coded here by a reduction on systems, rather than on individual terms.

The unification procedure described above is not fundamentally new, being developed by Dougherty ([Dou93]) for a typed combinatory logic, but our algorithm differs from his in two important ways. First, our transformations must account for the fact that a given term may have multiple, perhaps related, sorts, and must therefore subsume an order-sorted syntactic unification algorithm rather than that of Martelli and Montanari ([MM82]). Secondly, we subscribe to the Church view of the lambda calculus (and therefore of combinatory logic) and require the sorts of our terms to be completely specified. Dougherty allows incompletely specified types, and as a result his algorithm is finitely branching while ours is not, although we expect to be able to recover this feature by introducing sort variables. Nevertheless, when the sort structure collapses to a simple type structure, we obtain precisely the fragment of Dougherty's typed terms whose types are completely specified and, for those terms, our algorithm coincides with his.

The benefits for automated deduction afforded by incorporating sort information are more pronounced in calculi supporting a higher-order sort concept than in calculi all of whose base sorts are required to be non-functional. This is because the sort hierarchy propagates into the higher-order structure of the logic by means of the induced partial orders on sorts. Methods other than ours for inducing orderings on the set of all sorts from a given ordering on base sorts exist in the literature ([Car88], [Qia90], [Mit91], [NQ92]), but none of these permit functional base sorts. Our more expressive sort system, together with the requirements that systems to be unified are normalized and that narrowing is restricted to the heads of terms, results in a unification search space considerably more constrained than that of a naive order-sorted higher-order unification algorithm — developed, for example, along the lines of that in [Sch89]. This pruning of course contributes to the efficiency of any (reasonable) deduction calculus based on our algorithm.

Huet observed that although higher-order unification is undecidable ([Hue73], [Gol81]) — so that higher-order unification algorithms cannot be guaranteed to terminate in general — for deduction purposes a higher-order unification computation need only detect the possibility of unification ([Hue75]). While our algebraic rendering of the order-sorted higher-order unification problem naturally inherits the basic characteristics of the problem in its more traditional guise, we expect our solution to be suitable for *pre-unification*, as well as for complete order-sorted higher-order unification (see Section 4 for a brief discussion).

The importance of sorted calculi for automated deduction was initially recognized by Hayes ([Hay71]); Walther ([Wal87]) developed the first calculus combining resolution and sorted logic via an order-sorted unification procedure. Sorted higher-order logic has also been the focus of considerable interest from the point of view of higher-order program specification, the theory of functional programming languages, and object-oriented programming (*e.g.*, [Car88], [BL90], [Qia90], [CG91], [FP91], [Mit91], [Pie91], [Qia91]). Dougherty's typed algorithm is currently being implemented by Silverman ([Sil94]); implementations of the algorithms developed here and in [JK93], which would provide further data for comparing the efficiencies of lambda calculus and combinatory logic-based unification, have not yet been attempted.

1.1 Typed Calculi

We begin with a discussion of the simply typed lambda calculus, simply typed combinatory logic, and the relationship between them, since the sorted calculi which we define will generalize these standard higher-order calculi. We assume that the reader is familiar with the basic results concerning the simply typed lambda calculus and simply typed combinatory logic; [HS86] is an excellent source covering both.

Definition 1.1 The set $\mathcal{T}$ of *types* is obtained by inductively closing a set of *base types* $\mathcal{T}_0$ under function construction, *i.e.*, under the operation $\alpha \rightarrow \beta$. The *length* of a type α, denoted $length(\alpha)$, is the number of top-level arrows appearing in it.

Types will be 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. All other subdivisions of the universe would then be coded into sort distinctions among individuals, as described in the next section.

For each type $\alpha \in \mathcal{T}$, fix a countably infinite set of variables of type α and a countably infinite set of constants of type α. Write $x_\alpha, y_\alpha, z_\alpha \ldots$ for variables of type α and $a_\alpha, b_\alpha, c_\alpha \ldots$ for constants of type α. When discussing combinatory logic, we assume that there are typed *redex atoms* $I^\alpha \equiv I_{\alpha\to\alpha}$, $K^{\alpha\beta} \equiv K_{\alpha\to\beta\to\alpha}$, and $S^{\alpha\beta\gamma} \equiv S_{(\alpha\to\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma}$ for all types α, β, and γ. The variables, constants, and various I, K, and S are collectively called *typed atoms*; we assume that no two distinct typed non-redex atoms have the same type erasure.

$\mathcal{LC}$ is the set of explicitly simply typed lambda terms over the typed atoms excluding the various I, K, and S; $\mathcal{CL}$ is the set of explicitly simply typed combinatory logic terms over all typed atoms. By a "typed term" we will mean either a $\mathcal{LC}$- or $\mathcal{CL}$-term. We will write T_α if T is a typed term with type α, and omit the type of T when this will not lead to confusion.

On $\mathcal{LC}$, *$\beta\eta$-equality* is generated by *$\beta\eta$-reduction*, determined by the rules $(\lambda x.X)Y \xrightarrow{\beta} X[x := Y]$ for β-reduction, and $\lambda x.Xx \xrightarrow{\eta} X$ for η-reduction. We assume that β-reduction occurs without free variable capture, and that x is not free in X for the η-reduction rule. On $\mathcal{CL}$, *weak equality* is generated by *weak reduction*, determined by the (type-preserving) rules $Ix \longrightarrow x$, $Kxy \longrightarrow x$, and $Sxyz \longrightarrow xz(yz)$. Each of $\beta\eta$-reduction and weak reduction is terminating and confluent (*i.e., convergent*) on typed terms, so we can speak of *the $\beta\eta$-normal form* and *the long β-normal form* of a $\mathcal{LC}$-term, as well as *the weak normal form* of a $\mathcal{CL}$-term. As usual, we denote β-reduction on $\mathcal{LC}$-terms by $\xrightarrow{\beta}$, η-reduction on $\mathcal{LC}$-terms by $\xrightarrow{\eta}$, $\beta\eta$-reduction on $\mathcal{LC}$-terms by $\xrightarrow{\beta\eta}$, and weak reduction on $\mathcal{CL}$-terms by $\xrightarrow{w}$.

The reflexive, transitive closure of a relation $\xrightarrow{\nu}$ is denoted $\overset{\nu}{\twoheadrightarrow}$, and we will write $=_\nu$ for the symmetric closure of $\overset{\nu}{\twoheadrightarrow}$, i.e., for the equivalence relation generated by $\xrightarrow{\nu}$. $T_1 \equiv T_2$ indicates that the terms T_1 and T_2 are identical up to renaming of bound variables. We consider terms identical up to renaming of bound variables to be the same.

There are effective translations between $\mathcal{LC}$ and $\mathcal{CL}$. As in [HS86], we define $\mathcal{L} : \mathcal{CL} \to \mathcal{LC}$ and $\mathcal{H} : \mathcal{LC} \to \mathcal{CL}$ by:

- $\mathcal{L}(a) \equiv a$, when a is a non-redex atom,
- $\mathcal{L}(I) \equiv \lambda x.x$,
- $\mathcal{L}(K) \equiv \lambda xy.x$,
- $\mathcal{L}(S) \equiv \lambda xyz.xz(yz)$, and
- $\mathcal{L}(MN) \equiv \mathcal{L}(M)\mathcal{L}(N)$;

and

- $\mathcal{H}(a) \equiv a$,
- $\mathcal{H}(PQ) \equiv \mathcal{H}(P)\mathcal{H}(Q)$, and
- $\mathcal{H}(\lambda x.L) \equiv [x]\mathcal{H}(L)$, where
  - $[x]M \equiv KM$ when x does not occur in M,
  - $[x]x \equiv I$,
  - $[x](Mx) \equiv M$ when x does not occur in M, and

– $[x](MN) \equiv S([x]M)([x]N)$ otherwise.

Here the types of the terms are such that these translations are type-preserving.

Although $\mathcal{H}$ and $\mathcal{L}$ translate terms between $\mathcal{LC}$ and $\mathcal{CL}$, they are not translations of the respective *theories* — weak equality on $\mathcal{CL}$ is not sufficiently fine to reflect $\beta\eta$-equality on $\mathcal{LC}$. As pointed out in [Dou93], this can be seen by considering the distinct weak normal forms SK and KI and observing that they have $\beta\eta$-equal translations under $\mathcal{L}$.

Define *extensional combinatory equality*, henceforth *C-equality*, on $\mathcal{CL}$-terms by

$$M =_c N \text{ iff } \mathcal{L}(M) =_{\beta\eta} \mathcal{L}(N).$$

The translations $\mathcal{L}$ and $\mathcal{H}$ have the properties that

$$\mathcal{L}(\mathcal{H}(X)) =_{\beta\eta} X \text{ and } \mathcal{H}(\mathcal{L}(M)) \equiv M,$$

and it follows that for any $\mathcal{LC}$-terms X and Y,

$$X =_{\beta\eta} Y \text{ iff } \mathcal{H}(X) =_c \mathcal{H}(Y).$$

A $\mathcal{CL}$-term M is said to be in *C-normal form* if it is $\mathcal{H}X$ for some X in long β-normal form.

2 The Sorted Calculi

In this section, we develop the order-sorted lambda and combinatory calculi with which we will be concerned. The combinatory logic defined here is shown to be a counterpart to the order-sorted lambda calculus originally developed in [JK93] in the sense that i) it refines the simply typed extensional combinatory logic just as the order-sorted lambda calculus in [JK93] refines the simply typed lambda calculus, and ii) the translations $\mathcal{L}$ and $\mathcal{H}$ of the last section can be generalized to translations of the respective terms, as well as of the *theories*, of the two order-sorted calculi. We show in Lemma 3.7 that we may therefore solve any higher-order order-sorted unification problem given in terms of the lambda calculus of [JK93] by solving the translation of said problem in the order-sorted combinatory logic setting developed here.

2.1 Order-sorted Structures

The order-sorted structures over which the order-sorted lambda calculus and our order-sorted combinatory logic are defined are precisely those of [JK93]. We repeat relevant definitions and comments here, but refer to reader to that document for omitted proofs.

The sort mechanism on which our calculi will be based supports both essentially first-order and higher-order classification of terms. In mathematics, subdividing the universe of individuals gives rise to new classes of functions, namely those whose domains and codomains are among the various subdivisions. But in addition to this essentially first-order way of partitioning function universes, the classes of functions defined by domains and codomains can be further divided into subclasses, since functions are explicit objects of higher-order logic. To reflect this richer structuring of higher-order objects, we introduce into our calculus base sorts of functional type, *i.e.,* base sorts that denote classes of functions, as well as non-functional base sorts. Syntactically, each sort comes with a type, a codomain sort, and — if it is of functional type — also with a domain sort.

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 function construction 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*; 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 $A \in \mathcal{S}$, 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 $\gamma\,|_{\mathcal{S}^{nf}}$ 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 will be denoted by upper case Roman letters from the beginning of the alphabet. If the context is clear, we will abbreviate by $\mathcal{S}$ the sort system $(\mathcal{S}_0, \mathcal{S}, \tau, \delta, \gamma)$, and we may suppress references to $\mathcal{S}$ entirely when no confusion will arise. Since we are ultimately interested in sorted terms and their typed counterparts, we will only consider sort systems for which τ is surjective. We will further assume that for each $\alpha \in \mathcal{T}$ there exist only finitely many $A \in \mathcal{S}_0$ such that $\tau(A) = \alpha$, *i.e.*, that sort systems have *finitely many base sorts per type*. The type $\tau(A)$ is called the *type* of the sort A.

It will be useful to have some notational conventions for domain and codomain sorts. For any $A \in \mathcal{S}$, recursively define the following notation: $\gamma^0(A) \equiv A$, $\delta^0(A) \equiv A$, $\delta_0(A) \equiv A$, and for $i \geq 1$, $\gamma^i(A) \equiv \gamma(\gamma^{i-1}(A))$, $\delta^i(A) \equiv \delta(\delta^{i-1}(A))$, and $\delta_i(A) \equiv \delta(\gamma^{i-1}(A))$. If $A \equiv (B \rightarrow C) \rightarrow D \rightarrow E$, for example, then $\gamma^2(A) = E$, $\delta^2(A) = B$, and $\delta_2(A) = D$.

Example 2.2 Functional base sorts are useful, for example, in the study of elementary analysis, where one can postulate a non-functional base sort R denoting the reals, and a functional base sort C such that $\delta(C) = R$ and $\gamma(C) = R$ denoting the class of continuous real-valued functions on the reals. It is worth noting that it is not possible to syntactically distinguish the continuous real-valued functions on reals solely in terms of their domains and codomains, so that functional base sorts indeed increase 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 represented by sorts. The next definition captures a consistency condition we require such orderings to satisfy.

Definition 2.3 Given a sort system $\mathcal{S}$, for each 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 $\mathcal{S}$) is defined 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 $A, B \in \mathcal{S}^f$.

Definition 2.4 Given a sort system $\mathcal{S}$, a *sort structure* (for $\mathcal{S}$) is any set Δ of subsort declarations such that the judgement $\vdash_{ss} \Delta$ is provable in the following calculus:

$$\frac{}{\vdash_{ss} \emptyset} \qquad (ss - start)$$

$$\frac{\vdash_{ss} \Delta}{\vdash_{ss} \Delta \cup \mathrm{Con}(A, B)} \qquad (ss - ext)$$

The judgement $\vdash_{ss} \Delta$ is precisely the declaration that Δ is a sort structure. The rule (ss-start) guarantees that the empty set is a sort structure, and (ss-ext) indicates that sort structures may be built inductively by adding to an existing sort structure a set of subsort declarations of the form $\mathrm{Con}(A, B)$. Note that $\tau(A) = \tau(B)$ is entailed in the assumption that $\mathrm{Con}(A, B)$ is defined; this implies that for any sort structure Δ, if $[A \leq B] \in \Delta$, then $\tau(A) = \tau(B)$. In addition, since sort structures have finite derivations and since $\mathrm{Con}(A, B)$ is always finite, sort structures are themselves always finite.

The following lemma is not difficult.

Lemma 2.5 *For a sort structure Δ, $[A \leq B] \in \Delta$ iff Con$(A, B) \subseteq \Delta$.*

Proof. Necessity follows from Definition 2.3. Sufficiency is proved by observing that if $[A \leq B] \in \Delta$, then $[A \leq B] \in \mathrm{Con}(D, E)$ for some $D, E \in \mathcal{S}$ such that $\tau(D) = \tau(E)$ and $[D \leq E] \in \Delta$, and then inducting on $length(\tau(D))$. □

Any sort structure Δ induces an *inclusion ordering* $\leq$ on $\mathcal{S}$, inductively defined by the rules of Definition 2.6. The rule ($\leq$-start) indicates that the inclusion ordering is indeed determined by Δ, ($\leq$-incl) reflects the natural inclusion of function spaces, ($\leq$-cov) insures covariance in the codomain sort, and ($\leq$-refl) and ($\leq$-trans) require that the inclusion ordering determined by Δ be a quasi-ordering. For this ordering, we will write $\leq_\Delta$, or just $\leq$ as above if Δ is clear from the context, and $\sim$ for the equivalence relation induced by $\leq$.

Definition 2.6 For any sort structure Δ, the *inclusion ordering determined by* Δ contains all judgements of the form $\Delta \vdash A \leq B$ which are provable by the following calculus:

$$\frac{[A \leq B] \in \Delta}{\Delta \vdash A \leq B} \qquad (\leq -start)$$

$$\frac{A \in \mathcal{S}^f}{\Delta \vdash A \leq \delta(A) \rightarrow \gamma(A)} \qquad (\leq -incl)$$

$$\frac{\Delta \vdash A \leq B}{\Delta \vdash C \rightarrow A \leq C \rightarrow B} \qquad (\leq -cov)$$

$$\frac{}{\Delta \vdash A \leq A} \qquad (\leq -refl)$$

$$\frac{\Delta \vdash A \leq B \qquad \Delta \vdash B \leq C}{\Delta \vdash A \leq C} \qquad (\leq -trans)$$

We do not insist in ($\leq$-cov) that $\Delta \vdash A \leq B$ holds for any sorts A and B with a common domain sort D whose codomain sorts satisfy $\Delta \vdash \gamma(A) \leq \gamma(B)$. Letting A and B denote the class of surjective functions from D to $\gamma(A)$ and the class of surjective functions from D to $\gamma(B)$, respectively, demonstrates the undesirability of such a constraint (assuming, for example, a standard semantics).

Using Lemma 2.5, it is not hard to see that if Δ is a sort structure and $A, B \in \mathcal{S}^f$, then $\Delta \vdash A \leq B$ implies $\Delta \vdash \delta(A) \sim \delta(B)$ and $\Delta \vdash \gamma(A) \leq \gamma(B)$. In addition, the fact that any sort structure Δ contains only finitely many subsort declarations $[A \leq B]$, together with the assumption that sort systems have only finitely many base sorts per type, implies the decidability of the inclusion ordering $\leq$ determined by Δ. Although only semi-decidability of $\leq$ will be necessary for establishing semi-decidability of sort assignment — and hence for determining applicability of our unification algorithm — in fact decidability is not hard to prove. The proof requires the next three results, which, like it, are taken from [JK93].

Lemma 2.7 *For a sort structure Δ, if $\Delta \vdash A \leq B$, then $\tau(A) = \tau(B)$.*

Corollary 2.8 *A 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 which are mutually incomparable. That is, 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 $\leq$. Moreover, since $\mathcal{S}$ has only finitely many base sorts per type, the subsets $\mathcal{S}_\alpha$ are finite,* i.e., *$\mathcal{S}$ has finitely many sorts per type.*

Theorem 2.9 *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.*

Corollary 2.10 *For any sort structure Δ, the inclusion ordering determined by Δ is decidable.*

To define the signatures over which our well-sorted terms will be built, we require a final preliminary notion. It will turn out to be important that signatures "respect function domains," in the sense that for any term T and any sorts A and B such that T has sort A and also sort B, $\delta(A) \sim \delta(B)$ holds. The proof that signatures indeed satisfy this property (see Lemma 2.22) will depend in part on the consistency conditions embodied by Definition 2.3 and in part on the fact that constant declarations meet the sort condition of the fifth clause of Definition 2.13 below, given in terms of the relation Rdom, which we now define.

Definition 2.11 Given a sort structure Δ, the binary relation Rdom_Δ is defined by

$$\frac{A, B \in \mathcal{S}^{nf} \qquad \tau(A) = \tau(B)}{\mathrm{Rdom}_\Delta(A, B)}$$

$$\frac{A, B \in \mathcal{S}^{f} \quad \Delta \vdash \delta(A) \sim \delta(B) \quad \mathrm{Rdom}_\Delta(\gamma(A), \gamma(B))}{\mathrm{Rdom}_\Delta(A, B)}$$

We will write Rdom for Rdom_Δ when Δ can be discerned from the context, and A Rdom B in place of $\mathrm{Rdom}(A, B)$.

We collect some easy but important facts about Rdom.

Lemma 2.12 *For any sort structure Δ, the following statements hold:*

1. *Rdom is an equivalence relation.*
2. *If A Rdom B, then $\tau(A) = \tau(B)$.*
3. *If $\Delta \vdash A \leq B$, then A Rdom B.*

We are at last in a position to describe the signatures which are suitable for our purposes. In testing equality and unifiability of terms it will be convenient to have at our disposal a set of constants not appearing in any term in the given context; this is the motivation for the set $Pars$ below.

Definition 2.13 A *(sorted) signature* Σ comprises

- a sort system $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}, \delta, \gamma, \tau)$,
- a sort structure Δ (for $\mathcal{S}$),
- a countably infinite set $Vars_A$ of *(sorted) variables* $x, y, z, \ldots$ for each $A \in \mathcal{S}$, and, when discussing combinatory logic, an infinite well-sorted set $Pars_A$ of *(sorted) parameters* $d, e, \ldots$ as well,
- a set $\mathcal{C}$ of typed (but unsorted) constants, including the various I, K, and S when discussing combinatory logic, and
- a set of *constant declarations* of the form $[c_\alpha :: A]$ for $c \in \mathcal{C}$ such that $\tau(A) = \alpha$. We require that constant declarations for I, K, and S be of the form $[I^\alpha :: A \rightarrow A], [K^{\alpha\beta} :: A \rightarrow B \rightarrow A]$, and $[S^{\alpha\beta\gamma} :: (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C]$, respectively. We further assume that there is at least one constant declaration per redex constant, and that if $[c :: A]$ and $[c :: B]$ are any constant declarations, then A Rdom B.

Let $Pars = \bigcup\{Pars_A \mid A \in \mathcal{S}\}$.

While not necessary for strictly theoretical concerns, in a computational setting it is both reasonable and convenient to insist that for each $\alpha \in \mathcal{T}$, any signature contains only finitely many constant declarations involving constants of type α, *i.e.*, only *finitely many constant declarations per type*. We will do so throughout this paper. As a consequence, signatures which respect function domains will contain precisely one constant declaration (up to Δ-equivalence of sorts) for each typed redex atom.

The sorted variables and parameters, and the typed constants appearing in the constant declarations of a signature Σ, will be called *sorted atoms* (for Σ). The various I, K, and S appearing in constant declarations are called *sorted redex atoms* (for Σ). Any sorted variable can be regarded as a typed variable in a natural way by "forgetting" its sort information and retaining only its type information; if we denote the forgetful functor by $\overline{}$, then we may regard the sorted variable $x \in Vars_A$ as the typed variable $x_{\tau(A)}$. By prudently naming the sorted variables, we may arrange that the forgetful functor provides a bijection between the typed and sorted variables. We may also arrange that the images of parameters under the forgetful functor are distinct non-redex constants from $\mathcal{C}$ not appearing in any constant declarations in Σ. These assumptions are intended to avoid (merely) technical complications that might otherwise arise.

The requirement that $\tau(A) = \alpha$ for a constant declaration $[c_\alpha :: A]$ insures that sort assignments respect the types of constants. According to Definition 2.13, signatures permit constant overloading of a restricted nature, consistent with this requirement.

2.2 Sorted Terms and Their Properties

We now define and explore properties of the calculi with which we will be concerned.

Definition 2.14 The set of $\mathcal{LC}(\Sigma)$-*preterms* is built inductively from the sorted non-redex atoms for the signature Σ by abstraction and application. The set of $\mathcal{CL}(\Sigma)$-*preterms* is built inductively from the set of all sorted atoms for Σ by application.

We will write $\mathcal{PLC}(\Sigma)$ and $\mathcal{PCL}(\Sigma)$ for the set of $\mathcal{LC}(\Sigma)$-preterms and the set of $\mathcal{CL}(\Sigma)$-preterms, respectively.

Definition 2.15 Sort assignment for $\mathcal{PLC}(\Sigma)$ is given inductively by the following inference rules:

$$\frac{x \in Vars_A}{\Sigma \vdash x : A} \qquad (var)$$

$$\frac{p \in Pars_A}{\Sigma \vdash p : A} \qquad (par)$$

$$\frac{[c :: A] \in \Sigma}{\Sigma \vdash c : A} \qquad (const)$$

$$\frac{\Sigma \vdash X : A \text{ and } \Sigma \vdash Y : B \text{ and } \Delta \vdash B \sim \delta(A)}{\Sigma \vdash XY : \gamma(A)} \qquad (app)$$

$$\frac{x \in Vars_B \qquad \Sigma \vdash X : A}{\Sigma \vdash \lambda x.X : B \rightarrow A} \qquad (abs)$$

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

$$\frac{\Sigma \vdash X : B \qquad \Delta \vdash B \leq A}{\Sigma \vdash X : A} \qquad (weaken)$$

Definition 2.16 Sort assignment for $\mathcal{PCL}(\Sigma)$ is given inductively by the following inference rules:

$$\frac{x \in Vars_A}{\Sigma \vdash x : A} \quad (var)$$

$$\frac{p \in Pars_A}{\Sigma \vdash p : A} \quad (par)$$

$$\frac{[c :: A] \in \Sigma}{\Sigma \vdash c : A} \quad (const)$$

$$\frac{\Sigma \vdash M : A \text{ and } \Sigma \vdash N : B \text{ and } \Delta \vdash B \sim \delta(A)}{\Sigma \vdash MN : \gamma(A)} \quad (app)$$

$$\frac{\Sigma \vdash M : B \qquad \Delta \vdash B \leq A}{\Sigma \vdash M : A} \quad (weaken)$$

Let $\mathcal{LC}_A(\Sigma) = \{X \in \mathcal{PLC}(\Sigma) \mid \Sigma \vdash X : A\}$ and $\mathcal{LC}(\Sigma) = \bigcup_{A \in \mathcal{S}} \mathcal{LC}_A(\Sigma)$. Similarly, let $\mathcal{CL}_A(\Sigma) = \{M \in \mathcal{PCL}(\Sigma) \mid \Sigma \vdash M : A\}$ and $\mathcal{CL}(\Sigma) = \bigcup_{A \in \mathcal{S}} \mathcal{CL}_A(\Sigma)$. Then $\mathcal{LC}(\Sigma)$ is the set of well-sorted $\mathcal{LC}$-preterms, and similarly for $\mathcal{CL}(\Sigma)$. We will henceforth refer to the elements of $\mathcal{LC}(\Sigma)$ and $\mathcal{CL}(\Sigma)$ as *$\mathcal{LC}(\Sigma)$-terms* or *$\mathcal{CL}(\Sigma)$-terms*, as appropriate. By the unqualified word "term" or "well-sorted term" we will mean a $\mathcal{LC}(\Sigma)$-term or $\mathcal{CL}(\Sigma)$-term unless otherwise specified.

For any term T, write $\mathcal{S}_\Sigma(T)$ for $\{A \in \mathcal{S} \mid T \in \mathcal{LC}_A(\Sigma) \cup \mathcal{CL}_A(\Sigma)\}$. If $A \in \mathcal{S}_\Sigma(T)$ we say that *T has sort A*. The first five clauses of Definition 2.15 and all but the last of Definition 2.16 give an inductive assignment of a sort to every well-sorted term over the signature Σ. As with types, we will not explicitly indicate the sorts of terms unless it is necessary. We consider terms which are identical up to renaming of (sorted) variables to be the same.

A *pure* term is one containing no parameters. A $\mathcal{CL}(\Sigma)$-term is *functional* if it is of one of the forms I, K, KM, S, SM, or SMN; it is *passive* if it is of the form $hM_1...M_k$, $k \geq 0$, where the *head* h of the term is a sorted non-redex atom. Note that functional terms must have functional sorts. A passive term is *flexible* if its head h is a variable , and *rigid* otherwise. These notions extend those already defined for $\mathcal{LC}$ by Huet ([Hue75]) and for $\mathcal{CL}$ by Dougherty ([Dou93]).

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 by the rules (weaken) in Definitions 2.15 and 2.16, $\mathcal{LC}_A(\Sigma) = \mathcal{LC}_B(\Sigma)$ (resp. $\mathcal{CL}_A(\Sigma) = \mathcal{CL}_B(\Sigma)$) whenever $\Delta \vdash A \sim B$. By 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 all sorts in $\mathcal{S}$ by canonical $\sim$-equivalence class representatives, we arrive at a signature whose equivalence relation is trivial and such that $\mathcal{LC}_A(\Sigma') = \mathcal{LC}_A(\Sigma)$ (resp., $\mathcal{CL}_A(\Sigma') = \mathcal{CL}_A(\Sigma)$) for all sorts A. We may, and will therefore, assume without loss of generality that $\leq$ is a partial ordering for all signatures in the remainder of this paper. We will also assume that we have ridded our sort structure of redundant subsort declarations of the form $[A \leq A]$, and that if $\Delta \vdash A \leq B$, then $length(A) \leq length(B)$ holds. The latter assumption, which is required for the unification algorithm for $\mathcal{LC}(\Sigma)$ given in [JK93], is without loss of generality under, for example, a standard semantics.

A signature is said to be *subterm closed* if each subterm of a well-sorted term is again well-sorted. It is natural in the context of mathematics to expect signatures to be subterm closed, since it does not make sense to allow ill-formed subexpressions in well-sorted expressions (a situation that may be different in, for example, field of natural language processing). The proof that signatures are indeed subterm closed is straightforward, although because the rules (η) and (abs) provide different ways of sorting certain abstraction terms, we must look at the *derivation* proving that a $\mathcal{LC}(\Sigma)$-term is well-sorted, rather than only at the structure of X itself, to prove this — and, in fact, any — result concerning the sort of a subterm of a X. Except for use of the (weaken) rule, sort assignment in $\mathcal{CL}(\Sigma)$, on the other hand, is entirely structural, so that consideration of the structure of terms themselves often suffices to prove analogous results in that setting. In any case, we may assume without loss of generality that we never follow one application of a rule (weaken) by another in constructing any sort derivation for any term, since the inclusion ordering $\leq$ determined by Δ is transitive.

In any signature, variables and parameters have unique least sorts:

Lemma 2.17 *If Σ is a signature with sort structure Δ and $t \in Vars_A \cup Pars_A$, then t has least sort A in Σ,* i.e., *for all $B \in \mathcal{S}_\Sigma(t)$, $\Delta \vdash A \leq B$.*

Proof. According to Definitions 2.15 and 2.16, if $\Sigma \vdash t : B$ for any $B \not\equiv A$, then this fact must be the conclusion of an application of (weaken). The result is thus immediate. □

On the other hand, due to the possibility of constant overloading, it is not necessarily true that every term will have a unique least sort, *i.e.,* not every signature is a *regular* signature. Yet the various sorts a given term may have are guaranteed to be related in at least a rudimentary way:

Lemma 2.18 *For any signature Σ and any term T, if $\Sigma \vdash T : A$ and $\Sigma \vdash T : B$, then $\tau(A) = \tau(B)$.*

Proof. For $\mathcal{LC}(\Sigma)$ this is proved in [JK93]; for $\mathcal{CL}(\Sigma)$, it is immediate from Definitions 2.13 and 2.16, and Lemmas 2.7 and 2.12. □

As a corollary, we observe that for every $T \in \mathcal{LC}(\Sigma)$ or $\mathcal{CL}(\Sigma)$, the set $\mathcal{S}_\Sigma(T)$ is finite, since Σ has only finitely many sorts per type.

As a further consequence of Lemma 2.18 and the fact that signatures are subterm closed, we see that if we consider the forgetful functor to be the identity on typed constants, then it can be extended to well-sorted terms by induction on the derivations proving the terms well-sorted. This extension gives injections from $\mathcal{LC}(\Sigma)$ into $\mathcal{LC}$ and from $\mathcal{CL}(\Sigma)$ into $\mathcal{CL}$. The forgetful functor is not, however, bijective on $\mathcal{LC}(\Sigma)$ or $\mathcal{CL}(\Sigma)$ in general.

If Σ is a signature with exactly one sort A such that $\tau(A) = \alpha$ for each $\alpha \in \mathcal{T}_0$, and such that Δ is the empty sort structure, then Lemma 2.7 implies that the sort system $\mathcal{S}$ of Σ is isomorphic to $\mathcal{T}$ via the type assignment τ. Moreover, the set of constant declarations contains at most one declaration $[c :: A]$ per constant $c \in \mathcal{C}$, since constant declarations must respect the typing of the constants, and so $\mathcal{LC}(\Sigma)$ (resp., $\mathcal{CL}(\Sigma)$) is isomorphic to a fragment of $\mathcal{LC}$ (resp., $\mathcal{CL}$) whose only constants are the finitely many per type appearing in the constant declarations of Σ together with the constants which are images of parameters under the forgetful functor. The inclusion ordering determined by Δ can thus be seen as refining the type structure of the simply typed lambda calculus (resp., simply typed combinatory logic).

In order to prove computability of sort assignment for $\mathcal{LC}(\Sigma)$ and $\mathcal{CL}(\Sigma)$, we extend the function $\mathcal{S}_\Sigma(\cdot)$ on $\mathcal{LC}(\Sigma)$ (resp., $\mathcal{CL}(\Sigma)$) to all of $\mathcal{LC}$ (resp., $\mathcal{CL}$) via the forgetful functor.

Definition 2.19 For $T \in \mathcal{LC}$ (resp., $\mathcal{CL}$) and a signature Σ, define

$$\mathcal{S}_\Sigma(T) = \{\mathcal{S}_\Sigma(U) \mid U \in \mathcal{LC}(\Sigma) \cup \mathcal{CL}(\Sigma) \text{ and } \overline{U} \equiv T\}$$

According to this definition, $T \in \mathcal{LC} \setminus \mathcal{LC}(\Sigma)$ (resp., $\mathcal{CL} \setminus \mathcal{CL}(\Sigma)$) iff $\mathcal{S}_\Sigma(T) = \emptyset$, *i.e.*, iff there exists no $U \in \mathcal{LC}(\Sigma)$ (resp., $\mathcal{CL}(\Sigma)$) such that $\overline{U} \equiv T$. If such a U exists it is unique; in this case we abuse terminology and say that $T \in \mathcal{LC}$ (resp., $\mathcal{CL}$) is *well-sorted* with respect to Σ.

Theorem 2.20 *For any signature Σ and term T, $\mathcal{S}_\Sigma(T)$ is effectively computable.*

Proof. For $\mathcal{LC}(\Sigma)$, the result is proved in [JK93]; for $\mathcal{CL}(\Sigma)$, we proceed by induction on the structure of T.

- If $T \equiv \overline{t_A}$ for $t \in \mathit{Vars} \cup \mathit{Pars}$, then $\mathcal{S}_\Sigma(T) = \{B \mid \Delta \vdash A \leq B\}$, which is computable by Corollary 2.10.
- If $T = c_\alpha$, then $\mathcal{S}_\Sigma(T) = \{B \mid \Delta \vdash A \leq B \text{ for some } A \in \mathcal{S} \text{ with } [c :: A] \in \Delta\}$. This set is also computable by Corollary 2.10 and the fact that signatures have finitely many constant declarations per type.
- If $T \equiv UV$, then

  $$\mathcal{S}_\Sigma(T) = \{B \mid \Delta \vdash \gamma(A) \leq B \text{ for some } A \in \mathcal{S} \text{ with } A \in \mathcal{S}_\Sigma(U) \text{ and } \delta(A) \in \mathcal{S}_\Sigma(V)\}.$$

  This set is computable by the induction hypothesis and Corollary 2.10.

□

Corollary 2.21 *For any signature Σ and typed term T, it is decidable whether or not T is well-sorted with respect to Σ.*

We now prove that signatures respect function domains, in the sense that for every term T of functional sort and any sorts $A, B \in \mathcal{S}_\Sigma(T)$, we must have $\delta(A) = \delta(B)$. This unique domain sort is called the *supporting sort* of T and is denoted $\mathit{supp}(T)$. 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 explicitly specified domains of functions, which unique supporting sorts are intended to syntactically capture. Indeed, in mathematics, functions are assumed to have a unique (explicitly specified) domain, and must therefore be distinguished from restrictions to subdomains. For example, the addition function on the reals must be distinguished from the addition function on the natural numbers, and in general functions f and g should only be considered the same if $fa = ga$ for all a in the common (explicitly specified) domain of f and g. Observing these distinctions is necessary for a correct treatment of extensional higher-order calculi, and they must be reflected in the syntax of any such calculus.

Lemma 2.22 *If $\Sigma \vdash T : A$ and $\Sigma \vdash T : B$, then A Rdom B. That is, any signature Σ respects function domains.*

Proof. For $\mathcal{LC}(\Sigma)$, the proof appears in [JK93]; for $\mathcal{CL}(\Sigma)$ the proof is by induction on the derivations of $\Sigma \vdash T : A$ and $\Sigma \vdash T : B$. If $A \leq B$ or $B \leq A$, then the lemma holds by the third part of Lemma 2.12. We may therefore assume without loss of generality that no (weaken) steps appear in the derivations $\Sigma \vdash T : A$ and $\Sigma \vdash T : B$.

- If $\Sigma \vdash T : A$ by (var), then $\Sigma \vdash T : B$ is also the result of (var), and so $A \equiv B$.
- If $\Sigma \vdash T : A$ by (par), then $\Sigma \vdash T : B$ is also the result of (par), and so $A \equiv B$.
- If $\Sigma \vdash T : A$ by (const), then $\Sigma \vdash T : B$ is also the result of (const). Thus A Rdom B by Definition 2.13.
- If $\Sigma \vdash T : A$ by (app), then $T \equiv UV$ for some U, V, and $\Sigma \vdash T : B$ is also the result of (app). The result follows immediately by applying the induction hypothesis to U.

$\square$

Because the forgetful functor provides injections $\mathcal{LC}(\Sigma) \hookrightarrow \mathcal{LC}$ and $\mathcal{CL}(\Sigma) \hookrightarrow \mathcal{CL}$, we can extend the translations $\mathcal{L}$ and $\mathcal{H}$ on typed terms to translations (which we also call $\mathcal{L}$ and $\mathcal{H}$) between $\mathcal{LC}(\Sigma)$ and $\mathcal{CL}(\Sigma)$, as follows:

Definition 2.23 Let Σ be any signature. Given $X \in \mathcal{LC}(\Sigma)$, let M_X be the unique $\mathcal{CL}(\Sigma)$-term such that $\overline{M_X} \equiv \mathcal{H}\overline{X}$. Likewise, given $M \in \mathcal{CL}(\Sigma)$, let X_M be the unique $\mathcal{LC}(\Sigma)$-term such that $\overline{X_M} \equiv \mathcal{L}\overline{M}$.

Define $\mathcal{H} : \mathcal{LC}(\Sigma) \rightarrow \mathcal{CL}(\Sigma)$ and $\mathcal{L} : \mathcal{CL}(\Sigma) \rightarrow \mathcal{LC}(\Sigma)$ by

$$\mathcal{H}(X) \equiv M_X \text{ and } \mathcal{L}(M) \equiv X_M.$$

Writing $supp^i(T)$ for $\delta_i(A)$ when $\Sigma \vdash T : A$, it is tedious but not difficult to see that these translations are sort preserving, and that they act on terms as do their typed counterparts, *i.e.*, that

- $\mathcal{L}(a) \equiv a$ when a is a non-redex atom,
- $\mathcal{L}(I) \equiv \lambda x.x$ for $x \in Vars_{supp(I)}$,
- $\mathcal{L}(K) \equiv \lambda xy.x$ for $x \in Vars_{supp(K)}$ and $y \in Vars_{supp^2(K)}$,
- $\mathcal{L}(S) \equiv \lambda xyz.xz(yz)$ for $x \in Vars_{supp(S)}$, $y \in Vars_{supp^2(S)}$, and $z \in Vars_{supp^3(S)}$, and
- $\mathcal{L}(MN) \equiv \mathcal{L}(M)\mathcal{L}(N)$;

and

- $\mathcal{H}(a) \equiv a$ if a is an atom,
- $\mathcal{H}(XY) \equiv \mathcal{H}(X)\mathcal{H}(Y)$, and
- $\mathcal{H}(\lambda x.X) \equiv [x]\mathcal{H}(X)$ where $x \in Vars_A$ and

- $[x]M \equiv K^{\beta\alpha}M$ when x does not occur in M, $\tau(A) = \alpha$, and $\tau(\overline{M}) = \beta$,
- $[x]x \equiv I^{\alpha}$ if $\tau(A) \equiv \alpha$,
- $[x](Mx) \equiv M$ when x does not occur in M, and
- $[x](MN) \equiv S^{\alpha\beta\gamma}([x]M)([x]N)$ if $\tau(A) = \alpha$, $\tau(\overline{M}) = \beta \rightarrow \gamma$, and $\tau(\overline{N}) = \beta$, otherwise.

2.3 Order-sorted Reduction and Equality

We now fix an arbitrary signature Σ for use throughout the remainder of this paper.

As per the discussion immediately preceding Lemma 2.22 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 yield the original function again if B represents the (explicitly specified) domain of the function denoted by X. This restriction is embodied in the order-sorted η-rule of the next definition.

Definition 2.24 *Order-sorted $\beta\eta$-reduction* is defined to be the least reduction relation on $\mathcal{LC}(\Sigma)$ generated by the following rules:

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

The first rule above, which we assume to happen without free variable capture, is called *order-sorted β-reduction* and the second is called *order-sorted η-reduction.* Of course there are restrictions on the sorts of the terms implicit in the rules for $\mathcal{LC}(\Sigma)$-term formation. Observe, for example, that we must have $B \leq \mathit{supp}(X)$ in the order-sorted η-rule in order to ensure that $\lambda x.Xx \in \mathcal{LC}(\Sigma)$. But in fact we require the stronger condition that B actually be identically $\mathit{supp}(X)$ for the sake of properly handling extensionality.

It is possible to define order-sorted β-reduction without reference to typed β-reduction by

$$X \xrightarrow{\beta} Y \text{ iff } \overline{X} \xrightarrow{\beta} \overline{Y},$$

an equivalence of which we will make much use in what follows. But in the interest of having a self-contained definition, we prefer instead to *define* β-reduction wholly in terms of the order-sorted calculus.

Since order-sorted $\beta\eta$-reduction generalizes ordinary typed $\beta\eta$-reduction, we will write $\xrightarrow{\beta\eta}$ for order-sorted $\beta\eta$-reduction as well as for the typed version. We will similarly abuse notation in denoting the transitive, as well as the reflexive, symmetric, and transitive, closure, of $\xrightarrow{\beta\eta}$, since the typed relations are subsumed by their order-sorted versions.

It is important to our program that the fundamental operations of our calculi do not allow the formation of ill-sorted terms from well-sorted ones. This will ensure that our unification algorithm never has to handle ill-sorted terms, even intermediately. It is shown in [JK93] that η-equality is sort-preserving, and that $X \overset{\beta}{\twoheadrightarrow} Y$ implies $\mathcal{S}_{\Sigma}(X) \subseteq \mathcal{S}_{\Sigma}(Y)$, although the reverse inclusion does not hold in general.

Order-sorted β-reduction satisfies the usual properties associated with typed β-reduction, particularly convergence. That order-sorted $\beta\eta$-reduction on $\mathcal{LC}(\Sigma)$ is terminating follows from the fact that $X \overset{\beta\eta}{\twoheadrightarrow} Y$ implies $\overline{X} \overset{\beta\eta}{\twoheadrightarrow} \overline{Y}$; confluence follows from weak confluence of typed $\beta\eta$-reduction and the fact that if $X \overset{\beta\eta}{\longrightarrow} Y$ then $supp(X) \equiv supp(Y)$. It therefore makes sense to speak of *the* order-sorted β-normal form of a $\mathcal{LC}(\Sigma)$-term X. It is also sensible to refer to *the* order-sorted long β-normal form of a $\mathcal{LC}(\Sigma)$-term X, denoted $osl\beta nf(X)$. By this we mean the term obtained by computing the order-sorted long β-normal form of X and then performing (if needed) some order-sorted η^{-1}-reductions, as in [Bre88].

We are of course interested in the equality induced on $\mathcal{CL}(\Sigma)$ by order-sorted $\beta\eta$-equality on $\mathcal{LC}(\Sigma)$ under the (extended) translation $\mathcal{L}$. If we define *order-sorted C-equality* to be precisely this equality, *i.e.,* if we define

$$M =_c N \text{ iff } \mathcal{L}M =_{\beta\eta} \mathcal{L}N,$$

then as for the typed calculi, we have

$$\mathcal{L}(\mathcal{H}(X)) =_{\beta\eta} X \text{ and } \mathcal{H}(\mathcal{L}(M)) \equiv M,$$

and it follows that for any $\mathcal{LC}(\Sigma)$-terms X and Y,

$$X =_{\beta\eta} Y \text{ iff } \mathcal{H}(X) =_c \mathcal{H}(Y).$$

By analogy with the typed calculi, a $\mathcal{CL}(\Sigma)$-term M is said to be in *order-sorted C-normal form* if $M \equiv \mathcal{H}X$ for some X in order-sorted long β-normal form.

Unfortunately, the natural extension of typed weak equality, defined immediately below, does not capture order-sorted C-equality (this is a consequence of the well-known fact that typed weak equality does not capture typed C-equality). But we see now that as in the typed case, closing the rules for order-sorted weak equality under an appropriate notion of extensionality will indeed be enough to reflect order-sorted C-equality. The fact that order-sorted weak equality and order-sorted C-equality stand in exactly the same relation to one another as do weak equality and C-equality on $\mathcal{CL}$ is crucial for extending Dougherty's techniques for unification in $\mathcal{CL}$ to the order-sorted combinatory logic setting.

Definition 2.25 *Order-sorted weak reduction* is defined to be the least reduction relation on $\mathcal{CL}(\Sigma)$ stable under instantiation by well-sorted substitutions (to be defined in Section 3.1) and generated by the following rules:

- $I^\alpha x \longrightarrow x$ if $x \in Vars_A$ and $[I^\alpha :: A \rightarrow A] \in \Sigma$
- $K^{\alpha\beta} xy \longrightarrow x$ if $x \in Vars_A, y \in Vars_B$, and $[K^{\alpha\beta} :: A \rightarrow B \rightarrow A] \in \Sigma$
- $S^{\alpha\beta\gamma} xyz \longrightarrow xz(yz)$ if $x \in Vars_{A\rightarrow B\rightarrow D}, y \in Vars_{A\rightarrow B}, z \in Vars_A$, and $[S^{\alpha\beta\gamma} :: (A \rightarrow B \rightarrow D) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow D] \in \Sigma$

We defer the definition of "well-sorted substitution" merely for convenience of presentation. As with order-sorted β-reduction, it is possible to define order-sorted weak reduction on $\mathcal{CL}(\Sigma)$ with reference to the corresponding typed reduction, as in, for example, $IM \longrightarrow M$ iff $\overline{IM} \longrightarrow \overline{M}$. But as above, we will not. The properties of order-sorted weak reduction investigated in this section follow from this alternative characterization of order-sorted weak reduction, however.

Since order-sorted weak reduction generalizes the corresponding typed reduction we write $\xrightarrow{w}$ for order-sorted weak reduction as well as for typed weak reduction. We again abuse notation and denote the transitive closure of $\xrightarrow{w}$ by $\xrightarrow{w}\!\!\!\!\rightarrow$, and its reflexive, symmetric, transitive closure by $=_w$. That order-sorted weak reduction is convergent follows easily from the analogous result for typed weak reduction. We will therefore speak of *the* order-sorted weak normal form of a $\mathcal{CL}(\Sigma)$-term M. We will see in Lemma 3.19 that every C-normal form is also in weak normal form.

Although it is not in general possible to deduce any relationship between $\mathcal{S}_\Sigma(M)$ and $\mathcal{S}_\Sigma(N)$ for $M =_c N$, it is immediate from Definition 2.25 that $\mathcal{S}_\Sigma(M) \subseteq \mathcal{S}_\Sigma(N)$ whenever $M \xrightarrow{w}\!\!\!\!\rightarrow N$.

Again taking care to properly handle extensionality in the presence of ordered sorts, we have

Definition 2.26 The *order-sorted extensionality rule* is: for terms M and N of the same supporting sort A, $M =_{ext} N$ iff $Mz =_{ext} Nz$ for any (and hence all) $z \in Vars_A$ such that $z \notin Vars(M) \cup Vars(N)$.

This order-sorted extensionality rule clearly extends the usual extensionality rule to the sorted setting. Write $=_{w+ext}$ for the equality generated by adding the order-sorted extensionality rule to the rules for order-sorted weak reduction.

The relationship between order-sorted $\beta\eta$-reduction and the combination of order-sorted weak reduction and extensionality exactly parallels that in the typed case:

Lemma 2.27 *1. If $M \xrightarrow{w} M'$, then $\mathcal{L}(M) \xrightarrow{\beta}\!\!\!\!\rightarrow \mathcal{L}(M')$ via a non-empty sequence of β-reductions.*

2. If $X \xrightarrow{\beta\eta} X'$, then $\mathcal{H}(X) =_{w+ext} \mathcal{H}(X')$.

Proof. The first statement follows directly from the analogous result for the typed calculus. For the second, first observe that it suffices to see that $X \xrightarrow{\beta} X'$ implies $\mathcal{H}X =_{w+ext} \mathcal{H}X'$. The proof is by induction on the structure of X.

- If $X \equiv \lambda x.Y \xrightarrow{\beta} \lambda x.Y'$ with $Y \xrightarrow{\beta} Y'$, then by the induction hypothesis, $([x](\mathcal{H}Y))x =_w \mathcal{H}(Y) =_{w+ext} \mathcal{H}(Y') =_w ([x](\mathcal{H}Y'))x$, so that the order-sorted extensionality rule, which does indeed apply, gives $\mathcal{H}(\lambda x.Y) \equiv [x](\mathcal{H}Y) =_{w+ext} [x](\mathcal{H}Y') \equiv \mathcal{H}(\lambda x.Y')$.

- If $X \equiv (\lambda x.Y)Z \xrightarrow{\beta} Y[x := Z]$, then $\mathcal{H}X \equiv ([x]\mathcal{H}Y)(\mathcal{H}Z) \xrightarrow{w}\!\!\!\!\rightarrow (\mathcal{H}Y)[x := \mathcal{H}Z] \equiv \mathcal{H}(Y[x := Z])$.

- If $X \equiv YZ \xrightarrow{\beta} Y'Z'$ where $Y \xrightarrow{\beta}\!\!\!\!\rightarrow Y'$ and $Z \xrightarrow{\beta}\!\!\!\!\rightarrow Z'$, then the result follows immediately from the induction hypothesis.

□

The order-sorted extensionality rule, together with weak reduction, is expressive enough to capture order-sorted C-equality:

Theorem 2.28 *$M =_{w+ext} N$ iff $M =_c N$*

Proof. For the first direction, it suffices to see that $M =_c N$ if either $M \xrightarrow{w} N$ or $M =_c N$ by an application of the order-sorted extensionality rule. We have $M \xrightarrow{w} N$ implies $\mathcal{L}M \overset{\beta}{\twoheadrightarrow} \mathcal{L}N$, which in turn implies $M \equiv \mathcal{HL}M =_c \mathcal{HL}N \equiv N$. We also have that if $M =_{w+ext} N$ by an application of order-sorted extensionality, then $Mz =_{ext} Nz$ for all $z \in Vars_A$, where A is the common supporting sort of M and N and $z \notin Vars(M) \cup Vars(N)$. Then $Mz =_c Nz$, so that $(\mathcal{L}M)z \equiv \mathcal{L}(Mz) =_{\beta\eta} \mathcal{L}(Nz) \equiv (\mathcal{L}N)z$, and thus $\mathcal{L}M =_\eta \lambda z.(\mathcal{L}M)z =_{\beta\eta} \lambda z.(\mathcal{L}N)z =_\eta \mathcal{L}N$. Therefore $M \equiv \mathcal{HL}M =_c \mathcal{HL}N \equiv N$.

Conversely, $M =_c N$ iff $\mathcal{L}M =_{\beta\eta} \mathcal{L}N$, so that Lemma 2.27 implies $M \equiv \mathcal{HL}M =_{w+ext} \mathcal{HL}N \equiv N$. □

3 Order-sorted C-unification

Narrowing is a method for generating unifiers of systems modulo equational theories which is complete for theories admitting presentation as convergent term rewriting systems. Because it imposes constraints on the deduction steps that can be performed, narrowing, when complete, results in a smaller unification search space than would be obtained using more generally applicable techniques. Order-sorted weak reduction is convergent; narrowing is therefore a promising approach to unification modulo the equational theory generated by order-sorted weak equality.

As we observed in the last section, however, order-sorted weak reduction does not generate order-sorted C-equality. But adaptations of Curry's four equations (see [HS86]) which generate typed C-equality when added to the defining equations for the typed I, K, and S do combine with the rules for order-sorted weak reduction to exhibit an (infinite) equational presentation of order-sorted C-equality. It is thus not hard to see that order-sorted C-equality could submit to an order-sorted equational unification algorithm developed along the lines of that in [Sch89]. On the other hand, since order-sorted C-equality is decidable (by passing to $\mathcal{LC}(\Sigma)$ and performing order-sorted $\beta\eta$-reductions), we might hope for a convergent term rewriting system capturing it and therefore providing the foundation for a narrowing algorithm for order-sorted C-unification. Unfortunately, no confluent rewriting system is known, even for typed C-equality.

We nevertheless see in this section how the fact that order-sorted weak equality and order-sorted extensionality together generate order-sorted C-equality can be used to extend rewriting by order-sorted weak reduction to a relation on *systems* which is expressive enough to capture order-sorted C-equality. This technique, adapted to our sorted setting from [Dou93], is justified by Theorem 2.28. Furthermore, we will show that this relation on systems indeed supports a narrowing-like algorithm for unification modulo order-sorted C-equality in the presence of constant declarations, as does its unsorted counterpart for C-equality on $\mathcal{CL}$.

3.1 Substitutions and Systems

In the following, we represent unification problems by equational systems comprising the pairs of $\mathcal{CL}(\Sigma)$-terms to be simultaneously unified, and use transformations of such systems as our main tool for solving the unification problems they represent. Systems are also a convenient way to phrase questions of C-equality between terms. In this section notions of "system" and "substitution" appropriate to our order-sorted setting are defined and the connections between them are investigated.

Definition 3.1 A *pair* is a two-element multiset of $\mathcal{CL}(\Sigma)$-terms. A *system* is a finite set of pairs. A pair is *trivial* if its elements are identical, and *order-sorted C-valid* if its elements are equal modulo order-sorted C-equality. A system is *trivial* if each of its pairs is trivial, and *order-sorted C-valid* if each of its pairs is order-sorted C-valid.

If the symmetric difference of the systems Γ and Γ' is trivial, we write $\Gamma \simeq \Gamma'$. $Vars(\Gamma)$ denotes the set containing all variables appearing in terms in the system Γ. As usual, we write $\Gamma, \langle M, N\rangle$ instead of $\Gamma \cup \{\langle M, N\rangle\}$, but since Γ may or may not also contain $\langle M, N\rangle$, such a decomposition is ambiguous. We will use the notation $\Gamma; \langle M, N\rangle$ to abbreviate $\Gamma \cup \{\langle M, N\rangle\}$ when $\langle M, N\rangle$ is *not* a pair in Γ.

Definition 3.2 A pair $\langle M, N\rangle$ is *solved in* Γ if it is either trivial or of the form $\langle x, M\rangle$ where $x \in Vars_A$ appears exactly once in Γ and $A \in \mathcal{S}_\Sigma(M)$. In the latter case, x is said to be *solved* in Γ. If each pair in Γ is solved in Γ, then Γ is a *solved system.*

Definition 3.3 A *substitution* is an ordinary (type-preserving) finitely supported map from variables to $\mathcal{LC}(\Sigma)$- or $\mathcal{CL}(\Sigma)$-terms, as appropriate. A substitution θ induces a mapping on terms, which we will also denote by θ.

We will write substitution application as juxtaposition, so that θT is the application of the substitution θ to the term T. By $D(\theta)$ and $I(\theta)$ we will denote the set of variables in the domain of θ and the set of variables introduced by θ, respectively.

Definition 3.4 A substitution θ is *well-sorted* (or a *well-sorted substitution*) if for every $x \in Vars_A$, $A \in \mathcal{S}_\Sigma(\theta x)$; θ is *pure* if for every $x \in Vars$, θx is a pure term.

It follows that if $T \in \mathcal{LC}(\Sigma)$ (resp., $\mathcal{CL}(\Sigma)$) and θ is well-sorted, then $\theta T \in \mathcal{LC}(\Sigma)$ (resp., $\mathcal{CL}(\Sigma)$) as well. That the set of well-sorted substitutions is closed under composition is not hard to prove. We assume the standard results about ordinary (not necessarily well-sorted) substitutions.

We can extend equalities on terms to (well-sorted) substitutions in the usual manner:

Definition 3.5 Let $=_*$ be an equational theory on $\mathcal{LC}(\Sigma)$ or $\mathcal{CL}(\Sigma)$, W be a set of variables, and θ and θ' be substitutions. Then $\theta =_* \theta'[W]$ means that for every variable in $x \in W$, $\theta x =_* \theta' x$. Define the subsumption relation $\theta' \leq_* \theta[W]$ to hold provided there exists a substitution ρ such that $\theta =_* \rho\theta'[W]$.

If W is the set of all variables, we drop the notation "$[W]$." We will be primarily concerned with the cases when $=_*$ is $=_{\beta\eta}$, $=_c$, or the empty equational theory. In the latter case, we write simply "$\equiv$" and "$\leq$" for the induced equality and subsumption ordering on substitutions.

Substitutions on terms can be extended to mappings on systems $\Gamma \equiv \{\langle M_i, N_i\rangle \mid i \leq n\}$ by defining $\theta\Gamma$ to be the system $\{\langle \theta M_i, \theta N_i\rangle \mid i \leq n\}$.

Definition 3.6 A well-sorted substitution θ is an *order-sorted C-unifier* (resp., *order-sorted syntactic unifier*) of a system Γ if $\theta\Gamma$ is C-valid (resp., trivial). If σ is an idempotent C-unifier (resp., syntactic unifier) of Γ with the properties that $D(\sigma) \subseteq Vars(\Gamma)$ and that for any order-sorted C-unifier (resp., order-sorted syntactic unifier) θ of Γ, $\sigma \leq_C \theta$ (resp., $\sigma \leq \theta$) holds, then σ is said to be a *most general order-sorted C-unifier* (resp., *most general order-sorted syntactic unifier*) of Γ. A system Γ is *order-sorted C-unifiable* (resp., *order-sorted syntactically unifiable*) if there exists some order-sorted C-unifier (resp., order-sorted syntactic unifier) of Γ.

The following lemma shows how a method for unification with respect to order-sorted C-equality yields an alternate method for order-sorted higher-order unification in $\mathcal{LC}(\Sigma)$ to that in [JK93], and justifies our translation of unification problems from $\mathcal{LC}(\Sigma)$ to $\mathcal{CL}(\Sigma)$. If σ is a substitution, let the substitutions $(\mathcal{H} \circ \sigma)$ and $(\mathcal{L} \circ \sigma)$ be defined by $(\mathcal{L} \circ \sigma)T \equiv \mathcal{L}(\sigma T)$ and $(\mathcal{H} \circ \sigma)T \equiv \mathcal{H}(\sigma T)$.

Lemma 3.7 *Let X and Y be well-sorted terms. The well-sorted substitutions σ such that $\sigma X =_{\beta\eta} \sigma Y$ are (up to pointwise order-sorted $\beta\eta$-conversion) those of the form $(\mathcal{L} \circ \theta)$ where θ ranges over the order-sorted C-unifiers of $\langle \mathcal{H}(X), \mathcal{H}(Y)\rangle$.*

Proof. As in [Dou93], with the added observation that the translations preserve sorts. □

The remainder of this subsection explores the relationship between systems and substitutions.

If Γ is a solved system whose non-trivial pairs are $\langle x_1, M_1\rangle, ..., \langle x_n, M_n\rangle$ then these pairs determine an idempotent well-sorted substitution $\sigma_\Gamma = \{x_1 \mapsto M_1, ..., x_n \mapsto M_n\}$. Note that such a pair $\langle x, y\rangle$ requires a choice as to which of x and y is to be in the domain of the substitution; we will assume that a uniform way exists for making such a choice, and so will refer to *the* well-sorted substitution determined by a solved system. On the other hand, 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.

Note that any system Γ can be written as $\Gamma'; [\sigma]$ where σ is the set of solved pairs in Γ. 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 order-sorted unifiers is the following fact, which shows that solved systems indeed describe their own solutions:

Lemma 3.8 *If $\Gamma \equiv \langle M_1, N_1\rangle, ..., \langle M_n, N_n\rangle$ is a solved system, then σ_Γ is a most general order-sorted C-unifier (resp., most general order-sorted syntactic unifier) for Γ. In fact, for any order-sorted C-unifier θ of Γ, $\theta =_c \theta\sigma_\Gamma$ (resp., $\theta \equiv \theta\sigma_\Gamma$).*

Proof. Clearly σ_Γ is an order-sorted C-unifier (resp., order-sorted syntactic unifier) of Γ. Suppose that the non-trivial pairs of Γ are $\langle x_{i_j}, M_{i_j}\rangle$. If θ is any order-sorted C-unifier (resp., order-sorted syntactic unifier) of Γ, then $\theta\sigma_\Gamma x_{i_j} \equiv \theta M_{i_j} =_c \theta x_{i_j}$ (resp., $\theta\sigma_\Gamma x_{i_j} \equiv \theta M_{i_j} \equiv \theta x_{i_j}$) for $j = 1, ..., k$, and $\theta x \equiv \theta\sigma_\Gamma x$ for $x \notin D(\sigma_\Gamma)$, so that indeed $\theta =_c \theta\sigma_\Gamma$ (resp., $\theta \equiv \theta\sigma_\Gamma$). □

In general, however, an order-sorted syntactically unifiable system Γ will not have a single most general order-sorted syntactic unifier, although such a system will indeed have a finite complete set of order-sorted syntactic unifiers since signatures contain only finitely many constant declarations per type. An order-sorted C-unifiable system may not have a finite *complete set of order-sorted C-unifiers*, *i.e.*, a finite set U of order-sorted C-unifiers such that for every order-sorted C-unifier θ of Γ there exists a substitution $\sigma \in U$ such that $\sigma \leq_c \theta[Vars(\Gamma)]$. This quarrelsome behavior has nothing to do with sorts or our combinatory logic framework, however — it is inherited from the simply typed lambda calculus ([Gou66]).

3.2 Order-sorted C-validity

In extensional calculi, deciding order-sorted C-equality for arbitrary $\mathcal{CL}(\Sigma)$-terms reduces to the problem of deciding it for order-sorted weak normal forms of non-functional sort. The observation that two such terms $hM_1...M_k$ and $h'M'_1...M'_{k'}$ are equal modulo order-sorted C-equality precisely when $h \equiv h'$, $k = k'$, and for each i, $M_i =_c M'_i$ provides a straightforward treatment of order-sorted C-validity. By a *fresh* parameter we will mean one not occurring in any term in the current context.

Definition 3.9 The set $OSVT$ consists of the following reductions:

1. REDUCE
$$\Gamma; \langle M, N\rangle \longrightarrow \Gamma, \langle M', N\rangle$$
when M reduces to M' by order-sorted weak reduction.

2. ADD ARGUMENT
$$\Gamma; \langle M, N\rangle \longrightarrow \Gamma, \langle Md, Nd\rangle$$
when at least one of M and N is functional, M and N have the same supporting sort A, and d is the first fresh parameter in $Pars_A$.

3. PASSIVE DECOMPOSE
$$\Gamma; \langle hM_1...M_k, hN_1...N_k\rangle \longrightarrow \Gamma, \langle h, h\rangle, \langle M_1, N_1\rangle, ..., \langle M_k, N_k\rangle$$
when h is a non-redex atom.

Note that ";" is used on the left-hand sides of transformations, so that the effect of transformations is unambiguous, whereas "," is used on their right-hand sides to preclude repetition of identical pairs. We adopt the convention that no $OSVT$ reduction is done out of a trivial pair.

The notation for the $OSVT$ reductions exploits our convention that pairs are unordered. In ADD ARGUMENT, fresh parameters rather than variables are used as a reminder that these new arguments are not part of the original terms and should therefore not be instantiated. The restriction on d in that transformation is necessary to avoid generation of an order-sorted C-valid pair from one which is not order-sorted C-valid.

Although we may think of the $OSVT$ transformations as a rewriting system on systems of $\mathcal{CL}(\Sigma)$-terms, this analogy is imperfect in that i) ADD ARGUMENT can be applied only to the heads of terms since it changes their sorts, and ii) PASSIVE DECOMPOSE is not stable under substitution for a head variable. Fortunately, the facts that rewriting systems are closed under term-formation and stable under substitution do not have any effect on their ability to support unification procedures, as observed in [Dou93].

The analogous observations in [Dou93] directly give the following facts about $OSVT$:

Lemma 3.10 *1. Suppose $\Gamma \longrightarrow \Gamma'$. Then Γ' is order-sorted C-valid iff Γ is.*

2. Suppose that Γ is $OSVT$-irreducible. Then Γ is order-sorted C-valid iff it is trivial.

3. Every sequence of $OSVT$ reductions terminates.

Proof. The proofs of 1) and 2) are exactly as in the typed case. To establish 3), observe that any sequence of $OSVT$ reductions induces a non-collapsing sequence of VT reductions as defined in [Dou93], which are shown there to terminate (the $OSVT$ reductions clearly extend VT to our sorted setting.) □

We may thus decide order-sorted C-equality between terms M and N simply, by repeatedly applying $OSVT$ reductions in any order to the system originally comprising $\langle M, N\rangle$. Lemma 3.10 guarantees that any such reduction sequence terminates, and that $M =_c N$ iff the $OSVT$-irreducible system thus obtained is trivial. As observed in [Dou93], we may halt and report that M and N are not order-sorted C-equal if ever we generate a pair of passive terms whose heads are not identical.

Since the proof of the second part of Lemma 3.10 depends only on the assumption that terms M and N admit no weak *head* reductions, restricting REDUCE to applications at the heads of terms still leads to a complete decision procedure for order-sorted C-validity. This observation will allow us to correspondingly restrict the search space in our unification procedure, as discussed in Section 3.5.

Finally, it is worth observing that by contrast with validity in $\mathcal{CL}$, a pair of $\mathcal{CL}(\Sigma)$-terms need not comprise terms having precisely the same sorts to be order-sorted C-valid. Consequently, terms having different sorts can be order-sorted C-unifiable, although the images under the forgetful functor of order-sorted C-unifiable terms, like those of order-sorted C-valid terms, must have the same types.

3.3 Transformations for Order-sorted C-unification

In the remainder of this section we show how the reduction relation on systems developed in the last subsection can be lifted to give a narrowing-like order-sorted C-unification algorithm in which terms are "normalized" with respect to that relation. The following set of transformations, a refined version of which is proved in Section 3.6 to induce an order-sorted higher-order unification algorithm complete for all higher-order unification problems in our calculi, is adapted from the set UT in [Dou93], with which it coincides when $\mathcal{S}$ is isomorphic to $\mathcal{T}$. That these transformations are also complete for order-sorted syntactic unification in $\mathcal{CL}(\Sigma)$, as shown in Corollary 3.16, will be useful in proving their order-sorted C-unification completeness.

Definition 3.11 The set $OSUT$ comprises the following transformations:

1. *Decompose*

$$\Gamma; \langle hM_1...M_k, hM'_1...M'_k\rangle \Longrightarrow \Gamma, \langle M_1, M'_1\rangle, ..., \langle M_k, M'_k\rangle$$

   if h is any atom.

2. *Eliminate*

$$\Gamma; \langle x, M\rangle \Longrightarrow \sigma\Gamma, \langle x, M\rangle$$

   where $x \in Vars_A$, $x \notin Vars(M)$, $x \in Vars(\Gamma)$, and $\sigma = \{x \mapsto M\}$ is well-sorted.

3. *Narrow*

$$\Gamma; \langle M, N\rangle \Longrightarrow \sigma\Gamma, [\sigma], \langle \sigma M^*, \sigma N\rangle$$

   where there exists a non-variable subterm occurrence U of M and an order-sorted weak reduction rule $L \longrightarrow R$ with fresh variables such that L and U have most general order-sorted syntactic unifier σ, and M^* is obtained from M by substituting R for U.

4. *Add Argument*

$$\Gamma; \langle M, N\rangle \Longrightarrow \Gamma, \langle Md, Nd\rangle$$

   where d is the first fresh parameter of sort $supp(M) \equiv supp(N)$.

5. *Split*

$$\Gamma; \langle xM_1...M_n, hP_1...P_kN_1...N_n\rangle \Longrightarrow$$
$$\sigma\Gamma, [\sigma], \langle z_1, \sigma P_1\rangle, ..., \langle z_k, \sigma P_k\rangle, \sigma\langle M_1, N_1\rangle, ..., \sigma\langle M_n, N_n\rangle$$

   where $k, n \geq 0$, $x \in Vars_A$, h is a pure atom, and either

   (a) $h \in Vars_B$ or $[h :: B] \in \Sigma$ for some sort B such that $\Delta \vdash \gamma^k(B) \leq A$, $z_i \in Vars_{supp^i(h)}$ is fresh for $i = 1, ..., k$, and $\sigma = \{x \mapsto hz_1...z_k\}$,

   (b) $h \in Vars_B$ and $y \in Vars_D$ is fresh for some sorts B and D such that $m = length(\tau(D)) - length(\tau(B)) \geq 0$, $\Delta \vdash \gamma^{k+m}(D) \leq A$, and $\Delta \vdash \gamma^{k+m}(D) \leq \gamma^k(B)$, $w_i \in Vars_{\delta_i(D)}$ is fresh for $i = 1, ..., m$, $z_i \in Vars_{\delta_{m+i}(D)}$ is fresh for $i = 1, ..., k$, and $\sigma = \{x \mapsto yw_1...w_mz_1...z_k, h \mapsto yw_1...w_m\}$, or

   (c) $h \in Vars_B$ and there exist constant declarations $[a :: D]$ and $[a :: E]$ in Σ for some sorts B, D, and E, such that $m = length(\tau(D)) - length(\tau(B)) \geq 0$, $\Delta \vdash \gamma^{k+m}(D) \leq A$, $\Delta \vdash \gamma^{k+m}(E) \leq \gamma^k(B)$, $w_i \in Vars_{\delta_i(D)}$ is fresh for $i = 1, ..., m$, $z_i \in Vars_{\delta_{m+i}(D)}$ is fresh for $i = 1, ..., k$, and $\sigma = \{x \mapsto aw_1...w_mz_1...z_k, h \mapsto aw_1...w_m\}$.

In light of our observation that order-sorted syntactically unifiable terms need not have a most general order-sorted syntactic unifier, the assertion in *Narrow*, that L and U have a most general syntactic unifier requires comment. But it is easy to see that this follows from the fact that the left-hand-sides of order-sorted weak reduction rules do not have variables at the heads.

We adopt the convention that no $OSUT$ transformation is done out of a solved pair. This accords with our intuition that the solved pairs in a system are merely recording an answer substitution as it is incrementally built up. Notice that if $\Gamma \Longrightarrow \Gamma'$, then $\{x \mid x \text{ is solved in } \Gamma\} \subseteq \{x \mid x \text{ is solved in } \Gamma'\}$, so that solved variables remain solved after application of a $OSUT$ transformation.

We emphasize that there is no deletion of trivial pairs involving variables in this presentation, which simplifies certain arguments. For example, when a fresh variable is chosen during a computation, that variable is guaranteed to be new to the entire computation. This ensures that if $\Gamma \Longrightarrow \Gamma'$ under *OSVT* or *OSUT*, then $Vars(\Gamma) \subseteq Vars(\Gamma')$, and in Lemma 3.15 and Theorem 3.27 below eliminates the manipulation of "protected sets of variables" typically found in completeness proofs in the literature. It also eliminates complications in proving the soundness of resolution procedures based on our unification algorithm, and respects the fundamental idea behind the use of transformations to describe algorithms, namely, that the logic of the problem being considered can be abstracted from issues of data structures and control.

To reinforce the intuition that parameters are not part of out unification problems but are introduced only as dummy arguments, we will focus on pure answer substitutions in proving the order-sorted C-unification completeness of *OSUT*. While this ostensibly requires that we restrict our attention to pure unification problems, by suitably defining the set *Pars*, as pointed out in [Dou93], any problem may be considered a pure one.

Since *Split* is the only transformation in *OSUT* differing significantly from its *UT*-counterpart, we briefly illustrate its use.

Example 3.12 Let $[h_0 :: E]$, $[f_0 :: D \rightarrow E \rightarrow A]$, and $[f_0 :: D \rightarrow B]$ be non-redex constant declarations in a signature Σ with base sorts A, B, D, and E such that $\tau(A) = \iota$, $\tau(B) = \iota \rightarrow \iota$, $\tau(D) = \iota$, $\tau(E) = \iota$, and $\delta(B) = E$. Assume that $\gamma(B)$ is distinct from A, and that Σ contains any constant declarations according with Definition 2.13 for the redex constants and no subsort declarations. Let $f \in Vars_A$ and $g \in Vars_B$. Consider the order-sorted C-unifiable system

$$\Gamma \equiv \langle f, gh_0 \rangle.$$

This pair is not solved since $\Sigma \not\vdash gh_0 : A$, and *Eliminate*, *Decompose*, *Narrow*, and *Add Argument* do not apply to Γ. But clause (c) of *Split* does apply, since $[f_0 :: D \rightarrow E \rightarrow A]$ and $[f_0 :: D \rightarrow B]$ are in Σ, $m = 1$, and $\Delta \vdash \gamma^2(D \rightarrow E \rightarrow A) = A$ and $\Delta \vdash \gamma^2(D \rightarrow B) = \gamma(B)$. Letting w be fresh in $Vars_D$ and z be fresh in $Vars_E$, we have the *Split* step

$$\Gamma \equiv \langle f, gh_0 \rangle \Longrightarrow \langle f, f_0wz \rangle, \langle g, f_0w \rangle, \langle z, h_0 \rangle.$$

An application of *Eliminate* yields the solved system

$$\langle f, f_0wh_0 \rangle, \langle g, f_0w \rangle, \langle z, h_0 \rangle,$$

from which, anticipating Lemmas 3.13 and 3.17, we conclude that the substitution $\sigma = \{f \mapsto f_0wh_0, g \mapsto f_0w\}$ is both an order-sorted syntactic unifier, and an order-sorted C-unifier, of Γ.

Applications of the other clauses of *Split* to order-sorted C-unification problems proceed in a similar manner.

Let *FOSUT* be the subset of *OSUT* comprising *Decompose*, *Eliminate*, and *Split*. Then *FOSUT* is a sound and complete set of transformations for order-sorted syntactic unification in $\mathcal{CL}(\Sigma)$:

Lemma 3.13 *FOSUT is sound for order-sorted syntactic unification in* $\mathcal{CL}(\Sigma)$, i.e., *if* $\Gamma \Longrightarrow \Gamma'$ *by a FOSUT transformation and* θ *is an order-sorted syntactic unifier of* Γ', *then* θ *is also an order-sorted syntactic unifier of* Γ.

Proof. Since θ is assumed to be well-sorted, it suffices to see that it is a syntactic unifier of Γ.

When Γ' is obtained from Γ by *Decompose* or *Eliminate* this is proved in the usual way. In case Γ' is obtained from Γ by *Split*, the hypothesis entails that $\theta[\sigma]$ is trivial, so that $\theta\sigma \equiv \theta$. Thus $\theta\sigma\Gamma \equiv \theta\Gamma$, and so we need only see that $\theta(xM_1...M_n) \equiv \theta(hP_1...P_kN_1...N_n)$. But by hypothesis, $\theta z_i \equiv \theta\sigma P_i \equiv \theta P_i$ for $i = 1, ..., k$, and $\theta M_i \equiv \theta\sigma M_i \equiv \theta\sigma N_i \equiv \theta N_i$ for $i = 1, ..., n$, so that in fact we need only argue that $\theta x \equiv \theta(hP_1...P_k)$. If clause (b) of *Split* was used to obtain Γ', we compute

$$\theta x \equiv \theta\sigma x \equiv \theta(yw_1...w_kz_1...z_n) \equiv \theta\sigma(hP_1...P_k) \equiv \theta(hP_1...P_k).$$

The result follows in a similar fashion in case clause (a) or (c) was used to obtain Γ'. $\Box$

The proof of completeness uses the straightforward fact that if $\Sigma \vdash hM_1...M_n : A$ then there exists a sort B such that either $h \in Vars_B$ or $[h :: B] \in \Sigma$, and $\Delta \vdash \gamma^n(B) \leq A$, *i.e.*, that sort assignment for $\mathcal{CL}(\Sigma)$-terms is determined by the sorts of their heads. The proof is ultimately by induction on the measure μ defined on a substitution θ and a system Γ by letting $\mu(\theta, \Gamma)$ be the multiset of depths of terms in $\theta(\Gamma_U)$, where Γ_U is the unsolved part of Γ.

Lemma 3.14 *Let θ be a pure order-sorted syntactic unifier of an unsolved system Γ. Then there exist a system Γ' obtained from Γ by a FOSUT transformation, and a pure order-sorted syntactic unifier θ' of Γ', such that $\theta' \equiv \theta[Vars(\Gamma)]$ and $\mu(\theta', \Gamma') < \mu(\theta, \Gamma)$.*

Proof. Choose $\langle M, N\rangle$ unsolved in Γ. If the heads of M and N are not in $D(\theta)$, then they must be identical, so that we can apply *Decompose* to obtain Γ' and take $\theta' \equiv \theta$. Otherwise, $\langle M, N\rangle$ is of the form

$$\langle xM_1...M_n, hP_1...P_kN_1...N_m\rangle$$

with $x \in Vars_A \cap D(\theta)$, $k, n \geq 0$, and h pure.

If $n = 0$ and $\Sigma \vdash hP_1...P_k : A$, then *Eliminate* must apply to Γ since $\langle M, N\rangle$ is not solved. We may thus obtain Γ' and take $\theta' \equiv \theta$. Suppose, then, that neither *Decompose* nor *Eliminate* apply to Γ. Since $\langle M, N\rangle$ is order-sorted syntactically unifiable, we must have $\theta x \equiv \theta(hP_1...P_k)$ and $\theta M_i \equiv \theta N_i$ for $i = 1, ..., n$. Write $h'V_1...V_m$ for θh and W_i for θP_i, $i = 1, ..., k$. Then $\Sigma \vdash \theta x \equiv h'V_1...V_mW_1...W_k : A$, and $\Sigma \vdash \theta x \equiv h'V_1...V_mW_1...W_k : \gamma^k(B)$, since $\Sigma \vdash x : A$ and $\Sigma \vdash h : B$, respectively. There must therefore exist a sort D such that either $h' \in Vars_D$ or $[h' :: D] \in \Sigma$ and $\Delta \vdash \gamma^{m+k}(D) \leq A$, and there must exist a sort E such that either $h' \in Vars_E$ or $[h' :: E] \in \Sigma$ and $\Delta \vdash \gamma^{m+k}(E) \leq \gamma^k(B)$.

- If $h' \in Vars$, then $D \equiv E$ and indeed $m = length(\tau(D)) - length(\tau(B)) \geq 0$, so that we can apply clause (b) of *Split* with $y \equiv h'$ to $\langle M, N\rangle$ to get Γ' and take θ' to be $\theta \cup \{z_1 \mapsto W_1, ...z_n \mapsto W_k, w_1 \mapsto V_1, ..., w_m \mapsto V_m\}$. That θ' is a well-sorted, pure, order-sorted syntactic unifier of Γ' agreeing with θ on $Vars(\Gamma)$ is clear. To see that $\mu(\theta', \Gamma') < \mu(\theta, \Gamma)$, observe that $[\sigma]$ is solved in Γ' and that $\theta'\sigma\Gamma \equiv \theta'\Gamma \equiv \theta\Gamma$, so that it suffices to see that for each pair $\langle z_i, \theta P_i\rangle$ we have $\mu(\theta', \langle z_i, \sigma P_i\rangle) < \mu(\theta, \langle M, N\rangle)$, and that for each pair $\langle M_i, N_i\rangle$ we have $\mu(\theta', \langle\sigma M_i, \sigma N_i\rangle) < \mu(\theta, \langle M, N\rangle)$. But this is indeed true since for $i = 1, ..., k$, $\theta' z_i \equiv W_i$ is a proper subterm of θx and $\theta'\sigma P_i \equiv \theta' P_i \equiv \theta P_i$ is a proper subterm of $\theta(hP_1...P_k)$, and $\theta'\sigma M_i \equiv \theta M_i$ and $\theta'\sigma N_i \equiv \theta N_i$ for $i = 1, ..., n$.

- If $h' \in \mathcal{C}$, then $[h' :: D]$ and $[h' :: E]$ are in Σ and indeed $m = \mathit{length}(\tau(D)) - \mathit{length}(\tau(B)) \geq 0$. If $h \equiv h'$, then $m = 0$ and $[h :: D] \in \Sigma$ with $\gamma^k(D) \leq A$, so that we can apply clause (a) of *Split* to $\langle M, N \rangle$ to get Γ' and take θ' to be $\theta \cup \{z_1 \mapsto W_1, ..., z_k \mapsto W_k\}$. If $h \not\equiv h'$, then we can apply clause (c) of *Split* with $a \equiv h'$ to get Γ' and take θ' to be $\theta \cup \{z_1 \mapsto W_1, ...z_n \mapsto W_k, w_1 \mapsto V_1, ..., w_m \mapsto V_m\}$. In either case, the analysis then proceeds exactly as above.

□

Lemma 3.15 *FOSUT is a complete set of transformations for order-sorted syntactic unification in* $\mathcal{CL}(\Sigma)$, i.e., *if* θ *is a pure order-sorted syntactic unifier of a system* Γ*, then there exists a sequence of FOSUT transformations yielding a solved system* $[\sigma]$ *where* Σ *is a pure order-sorted syntactic unifier of* Γ *such that* $\sigma \leq \theta[Vars(\Gamma)]$.

Proof. Let θ be a pure order-sorted syntactic unifier of Γ. The proof is by induction on the well-founded measure $\mu(\theta, \Gamma)$.

- If $\mu(\theta, \Gamma) = 0$, then every pair in $\Gamma \simeq [\sigma_\Gamma]$ is solved. By Lemma 3.8, $\sigma_\Gamma \leq \theta$.
- If $\mu(\theta, \Gamma) > 0$, then Γ is unsolved, so that by the previous lemma, there exist a system Γ' such that Γ' is obtained from Γ by a *FOSUT* transformation and a pure order-sorted syntactic unifier θ' of Γ' such that $\theta' \equiv \theta[Vars(\Gamma)]$ and $\mu(\theta', \Gamma') < \mu(\theta, \Gamma)$. By the induction hypothesis, there is a sequence of *FOSUT* transformations out of Γ' yielding a solved system $[\sigma]$ whose associated substitution σ is a pure order-sorted syntactic unifier of Γ' satisfying $\sigma \leq \theta'[Vars(\Gamma')]$. The σ is an order-sorted syntactic unifier of Γ, and $[\sigma]$ can be obtained from Γ by a sequence of *FOSUT* transformations, and since $Vars(\Gamma) \subseteq Vars(\Gamma')$, $\sigma \leq \theta'[Vars(\Gamma)]$ holds. Finally, since $\theta' \equiv \theta[Vars(\Gamma)]$, we have $\sigma \leq \theta[Vars(\Gamma)]$, as desired.

□

Corollary 3.16 *OSUT is a complete set of transformations for order-sorted syntactic unification in* $\mathcal{CL}(\Sigma)$.

OSUT is also sound for order-sorted C-unification.

Lemma 3.17 *OSUT is sound for order-sorted* C*-unification on* $\mathcal{CL}(\Sigma)$, i.e., *if* $\Gamma \Longrightarrow \Gamma'$ *by an OSUT transformation and* θ *is an order-sorted* C*-unifier of* Γ'*, then* θ *is also an order-sorted* C*-unifier of* Γ.

Proof. We need only check that θ is a C-unifier of Γ. When Γ' is obtained from Γ by *Decompose* or *Eliminate* this is easily proved in the usual manner.

When Γ' is obtained from Γ by *Add Argument*, we must deduce $\theta M =_c \theta N$ from $\theta(Md) =_c \theta(Nd)$, where d is the first fresh parameter of sort A where A is the common supporting sort of M and N. But $(\theta M)d \equiv \theta(Md) \equiv \theta(Nd) \equiv (\theta N)d$, and A is also the common supporting sort of θM and θN, so that by order-sorted extensionality we may conclude $\theta M =_c \theta N$.

When Γ' is obtained from Γ by *Narrow* or *Split*, the hypothesis entails that $\theta[\sigma]$ is order-sorted C-valid, so that $\theta\sigma =_c \theta$. Thus $\theta\sigma\Gamma =_c \theta\Gamma$, and so we need only show that θ order-sorted C-unifies the "redex pair" of each transformation.

When the transformation applied is *Narrow*, we have $\sigma M =_c \sigma M^*$, so that $\theta M =_c \theta\sigma M =_c \theta\sigma M^* =_c \theta\sigma N =_c \theta N$, as desired.

When the transformation applied is *Split*, the hypothesis guarantees that $\theta z_i =_c \theta\sigma P_i =_c \theta P_i$ for $i = 1, ..., k$, and $\theta M_i =_c \theta\sigma M_i =_c \theta\sigma N_i =_c \theta N_i$ for $i = 1, ..., n$. We need only argue that $\theta x =_c \theta(hP_1...P_k)$. But the proof that this indeed holds is as exactly in Lemma 3.13 except that syntactic identity must be replaced throughout by order-sorted C-equality. □

We now address the completeness of $OSUT$ for order-sorted C-unification.

3.4 The Key Lemma

In proving the order-sorted C-unification completeness of $OSUT$ it is convenient to isolate a notion of C-unifier satisfying certain technical conditions. First, as mentioned above, we focus on pure answer substitutions. Second, following Dougherty, we relax the standard requirement that normalized substitutions map all variables to normal forms and allow solved variables to be mapped arbitrarily.

Definition 3.18 A pure idempotent well-sorted substitution θ is an *normalized order-sorted C-unifier of* Γ if

- $D(\theta) \subseteq Vars(\Gamma)$,
- $\theta\Gamma$ is C-valid, and
- for every variable x not solved in Γ, θx is in order-sorted C-normal form.

Write $NSCU(\Gamma)$ for the set of order-sorted normalized C-unifiers of Γ.

It is clear that every well-sorted substitution θ is order-sorted C-equal to a well-sorted substitution θ' with $D(\theta) = D(\theta')$ and $\theta' x$ in order-sorted C-normal form for all $x \in Vars$. Such a substitution is said to be *in order sorted C-normal form.* Thus for any order-sorted C-unifier θ of a system Γ, there exists a $\theta' \in NSCU(\Gamma)$ such that $\theta' =_c \theta[Vars(\Gamma)]$. In particular, every order-sorted C-unifiable system has a normalized order-sorted C-unifier.

The Lifting Lemma below is the key step in showing the order-sorted C-unification completeness of $OSUT$. The following properties of order-sorted C-normal forms, which rely on the corresponding results for typed combinatory logic (see [CF58]), guarantee behavior of order-sorted C-normal forms crucial to its proof.

Lemma 3.19 *The class of order-sorted C-normal forms has the following properties:*

1. *Order-sorted C-normal forms are unique in their order-sorted C-equivalence classes.*
2. *Every order-sorted C-normal form is in order-sorted weak normal form.*
3. *Every subterm of an order-sorted C-normal form is also in order-sorted C-normal form.*

Proof.

1. If $M =_c N$ and M and N are in order-sorted C-normal form, then there exist order-sorted long $\beta\eta$-normal forms X and Y such that $\mathcal{H}X \equiv M$ and $\mathcal{H}Y \equiv N$, and $X =_{\beta\eta} Y$. But order-sorted long β-normal forms are unique in their order-sorted $\beta\eta$-equivalence classes, so that $X \equiv Y$ follows. Therefore $M \equiv \mathcal{H}X \equiv \mathcal{H}Y \equiv N$.

2. We first see that if M is in C-normal form, then $\overline{M}$ is as well: since $M \equiv \mathcal{H}X$ for some order-sorted long β-normal form X, $\overline{M} \equiv \mathcal{H}\overline{X}$, and $\overline{X}$ is in β-normal form since X is (although not necessarily in long β-normal form). Then $\overline{M} \equiv \mathcal{H}\overline{Y}$ where Y is the long β-normal form of $\overline{X}$, and therefore $\overline{M}$ is indeed in C-normal form. With this fact in hand, we infer that since $\overline{M}$ is weakly irreducible, M is as well.

3. This follows from the observation that if M is in order-sorted C-normal form, then $\overline{M}$ is in C-normal form, together with the converse, a corollary of the next lemma. Momentarily assuming this, let M be in order-sorted C-normal form and let N be a subterm of M. Then $\overline{N}$ is a subterm of the C-normal form $\overline{M}$, and by the corresponding result for the typed case, $\overline{N}$ is in C-normal form. Therefore, so is N.

□

Lemma 3.20 *If X is the unique long $\beta\eta$-normal form such that $\mathcal{H}X \equiv \overline{M}$, then there exists an order-sorted long $\beta\eta$-normal form Y such that $\mathcal{H}Y \equiv M$ and $\overline{Y} \equiv X$.*

Proof. The proof is by structural induction on M.

- If M is a variable, parameter, or constant, take $Y \equiv osl\beta nf(\mathcal{L}M)$.
- If $M \equiv UV$ and $\overline{M} \equiv \mathcal{H}X \equiv \overline{UV}$, then $X \equiv YZ$ with $\mathcal{H}Y \equiv \overline{U}$ and $\mathcal{H}Z \equiv \overline{V}$. Then Z is in long $\beta\eta$-normal form since X is, and so is $\lambda w.Yw$. In addition, $\mathcal{H}(\lambda w.Yw) \equiv \mathcal{H}Y \equiv \overline{U}$. By the induction hypothesis, there are order-sorted long $\beta\eta$-normal forms Y' and Z' such that $\mathcal{H}Y' \equiv U$, $\mathcal{H}Z' \equiv V$, $\overline{Y'} \equiv \lambda w.Yw$, and $\overline{Z'} \equiv Z$. Thus $Y' \equiv \lambda u.Wu$ where $\overline{W} \equiv Y$, and $\mathcal{H}(WZ') \equiv M$ and $\overline{WZ'} \equiv YZ \equiv X$. Finally, WZ' is in long $\beta\eta$-normal form since X is.

□

Corollary 3.21 *If $\overline{M}$ is in C-normal form, then M is in order-sorted C-normal form.*

We are now in a position to prove the main lemma necessary for our completeness argument.

Lemma 3.22 *(Lifting Lemma) Let $\theta \in NSCU(\Gamma)$ and let $\langle M, N\rangle$ be an unsolved pair in Γ. If*

$$\theta\Gamma \longrightarrow \Lambda,$$

is an OSVT reduction out of $\langle \theta M, \theta N\rangle$, then there exists a Γ' and a θ' with

$$\Gamma \Longrightarrow \Gamma'$$

via OSUT such that

1. $\theta' \equiv \theta[Vars(\Gamma)]$,

2. $\theta'\Gamma' \simeq \Lambda$, *and*

3. $\theta' \in NSCU(\Gamma')$.

Proof. Let $\Gamma \equiv \Pi; \langle M, N\rangle$, where $\langle M, N\rangle$ is not solved and θ is order-sorted C-normal on the variables of M and N. The proof proceeds analogously to that in [Dou93].

If Λ is obtained by REDUCE, then

$$\theta\Gamma \equiv \theta\Pi, \langle \theta M, \theta N\rangle \longrightarrow \theta\Pi, \langle (\theta M)', \theta N\rangle \equiv \Lambda,$$

where $(\theta M)'$ is obtained from θM by an order-sorted weak reduction rule $L \longrightarrow R$ with fresh variables replacing, in θM, subterm $V \equiv \delta L$ by δR. Note that δ is well-sorted here. Then V is of the form θU for some subterm U of M. Since θ is pointwise order-sorted C-normal on the variables of M, U is not a variable. Construct M^* by substituting R for U in M and let σ be a most general order-sorted syntactic unifier of L and U, so that

$$\Gamma \equiv \Pi; \langle M, N\rangle \Longrightarrow [\sigma], \sigma\Gamma, \langle \sigma M^*, \sigma N\rangle \equiv \Gamma'$$

is a *Narrow* step. If θ' is $\theta \cup \delta$, then θ' is well-sorted since both θ and δ are, and the proof the conditions of the lemma hold proceeds exactly as in [Dou93].

If Λ is obtained by ADD ARGUMENT, then

$$\theta\Gamma \equiv \theta\Pi, \langle \theta M, \theta N\rangle \longrightarrow \theta\Pi, \langle (\theta M)d, (\theta N)d\rangle \equiv \Lambda.$$

Since d is the first fresh parameter of sort A, where A is the common supporting sort of θM and θN (and therefore of M and N), an application of *Add Argument* to Γ yields

$$\Gamma \equiv \Pi; \langle M, N\rangle \Longrightarrow \Pi, \langle Md, Nd\rangle \equiv \Gamma'.$$

If we take θ' to be θ, then the proof that the conditions of the lemma hold is again as in [Dou93].

If Λ is obtained by PASSIVE DECOMPOSE, there are two cases. If neither M nor N has a variable in $D(\theta)$ at the head, then we can obtain Γ' by applying *Decompose* to $\langle M, N\rangle$ and take θ' to be θ. Otherwise we can describe $\langle M, N\rangle$ as

$$\langle M, N\rangle \equiv \langle xM_1...M_n, hP_1...P_kN_1...N_n\rangle,$$

where $x \in Vars_A \cap D(\theta)$, $k, n \geq 0$, and h is a pure atom. We can describe $\langle \theta M, \theta N\rangle$ as

$$\langle \theta M, \theta N\rangle \equiv \langle aV_1...V_mU_1...U_kL_1...L_n, aV_1...V_mW_1...W_kQ_1...Q_n\rangle,$$

for some $m \geq 0$, with

$$\theta x \equiv aV_1...V_mU_1...U_k,$$
$$\theta h \equiv aV_1...V_m,$$
$$\theta P_i \equiv W_i, 1 \leq i \leq k,$$
$$\theta M_i \equiv L_i, 1 \leq i \leq n, \text{ and}$$
$$\theta N_i \equiv Q_i, 1 \leq i \leq n.$$

The repetition of the V_i is justified by the facts that θ is order-sorted C-normal on the variables of M and N and that order-sorted C-normal terms are unique in their order-sorted C-equivalence classes. Note that h cannot be a parameter since θ is pure.

If $h \notin D(\theta)$, then $\theta h \equiv h$ and $\Sigma \vdash \theta x \equiv hU_1...U_k : A$. Then there must exist a sort B such that either $h \in Vars_B$ or $[h :: B] \in \Sigma$ and $\Delta \vdash \gamma^k(B) \leq A$. We can apply clause (a) of *Split* to get Γ' and let θ' be $\theta \cup \{z_1 \mapsto U_1, ..., z_k \mapsto U_k\}$, so that indeed $\theta' \equiv \theta[Vars(\Gamma)]$. To see that $\theta'\Gamma' \simeq \Lambda$, we first note that $\theta' x \equiv \theta x \equiv hU_1...U_k \equiv \theta'(hz_1...z_k) \equiv \theta'\sigma x$, so that $\theta'\sigma \equiv \theta'$. Thus the pairs of $\theta'\Gamma'$ are identical to the pairs of Λ except that the trivial subsystem $\theta[\sigma]$ of $\theta'\Gamma'$ does not appear in Λ and the trivial pair $\langle h, h\rangle$ does not appear in $\theta'\Gamma'$. That $\theta' \in NSCU(\Gamma')$ follows from the facts that $\theta \in NSCU(\Gamma)$, $\Sigma \vdash U_i : supp^i(h)$ for $i = 1, ..., k$, $\theta'\Gamma' \simeq \Lambda$, and $OSVT$ reductions preserve order-sorted C-validity.

If $h \in Vars_B \cap D(\theta)$, then $\Sigma \vdash \theta x : A$ and $\Sigma \vdash \theta x \equiv \theta h : B$. Thus if $a \in Vars_D$, then $m = length(\tau(D)) - length(\tau(B))$, $\Delta \vdash \gamma^{m+k}(D) \leq A$, and $\Delta \vdash \gamma^{k+m}(D) \leq \gamma^k(B)$. We can apply clause (b) of *Split* with $y \equiv a$ to get Γ' and let θ' be $\theta \cup \{w_1 \mapsto V_1, ..., w_m \mapsto V_m, z_1 \mapsto U_1, ..., z_k \mapsto U_k\}$, so that $\theta' \equiv \theta[Vars(\Gamma)]$ is immediate. To see that $\theta'\Gamma' \simeq \Lambda$, we observe that $\theta' x \equiv \theta'(aw_1...w_m z_1...z_k)$ and $\theta' h \equiv \theta'(aw_1...w_m)$ so that $\theta'\sigma \equiv \theta'$. The rest of the analysis proceeds as in the case $h \notin D(\theta)$.

If $h \in Vars_B \cap D(\theta)$ and $A \in \mathcal{C}$, then $\Sigma \vdash \theta h \equiv aV_1...V_m : B$ implies that there exists a sort E such that $[a :: E] \in \Sigma$, $m = length(\tau(E)) - length(\tau(B))$, and $\Delta \vdash \gamma^{k+m}(E) \leq \gamma^k(B)$. On the other hand, $\Sigma \vdash \theta x \equiv aV_1...V_m U_1...U_k : A$ implies that there exists a sort D such that $[a :: D] \in \Sigma$ and $\Delta \vdash \gamma^{m+k}(D) \leq A$. We can apply clause (c) of *Split* to get Γ' and let θ' be $\theta \cup \{w_1 \mapsto V_1, ..., w_m \mapsto V_m, z_1 \mapsto U_1, ..., z_k \mapsto U_k\}$, and then proceed as in the previous cases to verify the conclusion of the lemma. □

3.5 Refinements

Before presenting our order-sorted C-unification algorithm based on $OSUT$ and proving its completeness, we describe the sense in which it is a "normalized narrowing" algorithm, and make a few additional observations which help constrain the non-determinism of the algorithm. First we note that rigid/rigid reductions preserve the order-sorted C-unifiers of systems, where a *rigid/rigid* reduction is an application of PASSIVE DECOMPOSE out of a pair of rigid terms. If we say that a system Γ is *simple* if each term in Γ is passive and irreducible with respect to order-sorted weak reduction and there is no pair of rigid terms in Γ with identical heads, then the next lemma is an immediate consequence of the fact that $OSVT$ is terminating.

Lemma 3.23 *Any sequence of* REDUCE, ADD ARGUMENT, *and rigid/rigid* PASSIVE DECOMPOSE *reductions applied to a system will terminate in a simple system with the same order-sorted C-unifiers.*

Simple systems are thus those which are "normalized" with respect to $OSVT$ reductions.

In light of our convention that $OSVT$ reductions not be applied out of trivial pairs, it will be most efficient to perform rigid/rigid PASSIVE DECOMPOSE reductions eagerly. In Section 3.2 we observed that applications of $OSVT$ reductions can be confined to the heads of terms in order-sorted C-validity proofs, suggesting that the transformations of the next definition, together with *Decompose*, *Eliminate*, *Add Argument*, and *Split*, are a suitable basis for an order-sorted C-unification algorithm.

Definition 3.24 The set $HOSUT$ consists of *Decompose*, *Eliminate*, *Add Argument*, and *Split*, together with the following *order sorted Head-Narrow* transformations:

- *I-Narrow*

  $$\Gamma; \langle xUZ_1...Z_n, Y\rangle \Longrightarrow \sigma\Gamma, [\sigma], \sigma\langle UZ_1...Z_n, Y\rangle$$

  if $x \in Vars_A$, $\tau(A) = \alpha \to \alpha$, and $\sigma = \{x \mapsto I^\alpha\}$ is well-sorted.

- *K-Narrow*

  $$\Gamma; \langle xUVZ_1...Z_n, Y\rangle \Longrightarrow \sigma\Gamma, [\sigma], \sigma\langle UZ_1...Z_n, Y\rangle$$

  if $x \in Vars_A$, $\tau(A) = \alpha \to \beta \to \alpha$, and $\sigma = \{x \mapsto K^{\alpha\beta}\}$ is well-sorted.

- *Kz-Narrow*

  $$\Gamma; \langle xVZ_1...Z_n, Y\rangle \Longrightarrow \sigma\Gamma, [\sigma], \sigma\langle zZ_1...Z_n, Y\rangle$$

  if $x \in Vars_A$, $z \in Vars_{\gamma(A)}$ is fresh, $\tau(A) = \beta \to \alpha$, and $\sigma = \{x \mapsto K^{\alpha\beta}z\}$ is well-sorted.

- *S-Narrow*

  $$\Gamma; \langle xUVWZ_1...Z_n, Y\rangle \Longrightarrow \sigma\Gamma, [\sigma], \sigma\langle UW(VW)Z_1...Z_n, Y\rangle$$

  where $x \in Vars_A$, $\tau(A) = (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$, and $\sigma = \{x \mapsto S^{\alpha\beta\gamma}\}$ is well-sorted.

- *Sz-Narrow*

  $$\Gamma; \langle xVWZ_1...Z_n, Y\rangle \Longrightarrow \sigma\Gamma, [\sigma], \sigma\langle zW(VW)Z_1...Z_n, Y\rangle$$

  where $x \in Vars_A$, $z \in Vars_{\delta^2(A)\to\gamma(\delta(A))\to\gamma^2(A)}$ is fresh, $\tau(A) = (\alpha \to \beta) \to \alpha \to \gamma$, and $\sigma = \{x \mapsto S^{\alpha\beta\gamma}z\}$ is well-sorted.

- *Syz-Narrow*

  $$\Gamma; \langle xWZ_1...Z_n, Y\rangle \Longrightarrow \sigma\Gamma, [\sigma], \sigma\langle yW(zW)Z_1...Z_n, Y\rangle$$

  where $x \in Vars_A$, β is any type and B is any sort such that $\tau(B) = \beta$, $y \in Vars_{\delta(A)\to B\to\gamma(A)}$ and $z \in Vars_{\delta(A)\to B}$ are fresh, $\tau(A) = \alpha \to \gamma$, and $\sigma = \{x \mapsto S^{\alpha\beta\gamma}yz\}$ is well-sorted.

The *Head Narrow* transformations correspond to order-sorted weak reductions out of the heads of terms.

Application of the transformations is more constrained than the notation here suggests, since all terms in systems resulting from their application must be well-sorted and have appropriate associated types. But the fact that B cannot be completely specified in *Syz-Narrow* means that while each of the other transformations in *HOSUT* is finitely branching, it is not; this undesirable behavior comes from the corresponding behavior in $\mathcal{CL}$, although it may be remediable via the introduction of sort variables in a manner similar to that of [Dou93]. Our order-sorted C-unification algorithm based on *HOSUT* is therefore not necessarily finitely branching.

3.6 The Algorithm and its Completeness

Bearing in mind the above discussion, we arrive at the following algorithm.

Definition 3.25 The non-deterministic algorithm $\mathcal{OSU}$ is the following process. Repeatedly

1. Reduce the system to a simple system and then apply some $OSUT$ transformation out of an unsolved pair.

2. If at any point we arrive at a system Γ which is order-sorted syntactically unifiable, then *optionally* use $FOSUT$ to compute an order-sorted syntactic unifier γ of the unsolved part of Γ, and return $\gamma\sigma$, where $[\sigma]$ is the solved part of Γ.

Of course if at any point in an $\mathcal{OSU}$ computation we arrive at a pair of passive terms whose heads are distinct constants, or at a pair of terms with different types, then we may halt and report non-unifiability. Correctness of Algorithm $\mathcal{OSU}$ follows from Lemmas 3.23 and 3.17.

We saw in Lemma 3.8 that solved systems represent their own most general order-sorted syntactic and C-unifiers. But it is clear that semantic unification procedures cannot simply transform systems to be unified into solved systems, since some semantic unifiers of a system may be more general than each of its most general syntactic unifiers. The system $\langle Kax, Kay\rangle$ given in [Dou93], for example, has the identity as an order-sorted C-unifier, but each of its most general order-sorted syntactic unifiers has non-empty domain. This explains why computing order-sorted syntactic unifiers in the second step of Algorithm $\mathcal{OSU}$ must be optional. The restriction in Definition 3.25 to unsolved pairs is justified by the following lemma, which shows that we need not be concerned with solved pairs of a system when computing its order-sorted C-unifiers. The lemma is therefore consistent with the intuition that the solved part of a system is merely a record of an answer substitution in the process of being constructed.

Lemma 3.26 *Suppose Γ is an order-sorted syntactically unifiable system with solved part $[\sigma]$ and unsolved part Γ_U. If θ is an order-sorted C-unifier of Γ and an order-sorted syntactic unifier of Γ_U, then for every order-sorted syntactic unifier γ of Γ_U such that $D(\gamma) \subseteq Vars(\Gamma_U)$ and $\gamma \leq \theta[Vars(\Gamma_U)]$, $\gamma\sigma$ is an order-sorted syntactic unifier of Γ and $\gamma\sigma \leq_c \theta[Vars(\Gamma)]$.*

Proof. Let ν be an order-sorted syntactic unifier of Γ_U such that $D(\nu) \subseteq Vars(\Gamma_U)$ and $\nu \leq \theta[Vars(\Gamma_U)]$. Then $\nu\sigma \leq \theta\sigma[Vars(\Gamma)]$ and $\nu\sigma$ unifies Γ since $\nu\sigma[\sigma]$ is trivial and $\nu\sigma\Gamma_U \equiv \nu\Gamma_U$ is trivial. Since θ C-unifies $[\sigma]$ and σ is idempotent, we have $\theta\sigma =_c \theta$. That $\nu\sigma \leq \theta\sigma =_c \theta[Vars(\Gamma)]$ follows. □

Theorem 3.27 *(Completeness) Let θ be a pure order-sorted C-unifier of Γ. Then there is a computation of Algorithm $\mathcal{OSU}$ on Γ producing a pure order-sorted C-unifier ν of Γ with $\nu \leq_c \theta[Vars(\Gamma)]$.*

Proof. Since every pure C-unifier of Γ is pointwise C-equal to an order-sorted normalized C-unifier, we may prove the theorem under the additional hypothesis that $\theta \in NSCU(\Gamma)$.

Let the *degree* of a system be the maximum length of an $OSVT$ sequence out of it. The proof is by induction on the degree of $\theta\Gamma$.

Let $[\sigma]$ be the solved part of Γ and Γ_U be its unsolved part. If θ is an order-sorted syntactic unifier of Γ_U, then since the *FOSUT* transformations are complete for order-sorted syntactic unification, we can apply a sequence of them to Γ_U to arrive at an order-sorted syntactic unifier σ' of Γ_U such that $\sigma' \leq \theta[Vars(\Gamma_U)]$. Let γ be the restriction of σ' to $Vars(\Gamma_U)$. Then $\gamma \leq \theta[Vars(\Gamma_U)]$ and so by Lemma 3.26, $\gamma\sigma$ is an order-sorted syntactic unifier of Γ with $\gamma\sigma \leq_c \theta[Vars(\Gamma)]$. We may return $\nu \equiv \gamma\sigma$ as an answer substitution. This situation obtains when the degree of $\theta\Gamma$ is 0.

Otherwise we define a system Γ' and a substitution θ' as follows:

1. If Γ is not simple, apply an *OSVT* reduction to obtain Γ' and let θ' be θ.
2. Otherwise there exists an unsolved pair $\langle M, N\rangle$ from Γ such that $\theta M \not\equiv \theta N$ and an *OSVT* reduction out of $\langle \theta M, \theta N\rangle$ (at the head if it is to be a REDUCE reduction) yielding Δ. The lifting lemma applies, yielding Γ' and θ'.

In each case the action performed is an $\mathcal{OSU}$ step, $\theta' \in NSCU(\Gamma')$, and the degree of $\theta'\Gamma'$ is less than the degree of $\theta\Gamma$ (since $\theta\Gamma'$ and Δ are identical up to trivial pairs, and no *OSVT* reductions are done out of trivial pairs).

By induction, there is a computation of Algorithm $\mathcal{OSU}$ on Γ' producing a C-unifier ν of Γ' with $\nu \leq_c \theta'[Vars(\Gamma')]$. ν is a C-unifier of Γ, and since $Vars(\Gamma) \subseteq Vars(\Gamma')$, we have $\nu \leq_c \theta'[Vars(\Gamma)]$. But since $\theta' \equiv \theta[Vars(\Gamma)]$, $\nu \leq_c \theta[Vars(\Gamma)]$ as desired. □

Since the choice of *OSUT* transformation to be applied in Algorithm $\mathcal{OSU}$ is nondeterministic (as is the choice of pair to which it is to be applied), we may infer that the C-unification strategy of eagerly applying *Eliminate* to systems is complete. It is not, however, known to be true that eager variable elimination is complete for an arbitrary calculus and equational theory, even if both are first-order.

We conclude with an example illustrating the way in which Algorithm $\mathcal{OSU}$ can also be used to discover non-unifiability of systems constrained by sort requirements.

Example 3.28 Let Σ be a signature with base sorts D, I, and R, where the non-functional sort R (with $\tau(R) = \iota$) is intended to denote 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 $\gamma(D) = \gamma(I) = R$. Let $[n :: D \longrightarrow I]$ and $[4 :: R]$ comprise the set of non-redex constant declarations of Σ, where n is intended to denote the "negation functor" mapping each function F to $-F$, and 4 denotes the real number four. Finally, assume any constant declarations in accordance with Definition 2.13 for the redex constants.

Let $x \in Vars_R$, $f \in Vars_I$, and $g \in Vars_D$, and consider the unification problem given by the pairs

$$\Gamma \equiv \langle f4, ngx\rangle, \langle gx, 4\rangle.$$

We may apply *Split* to get

$$\langle f, nz\rangle, \langle z, g\rangle, \langle x, 4\rangle, \langle gx, 4\rangle,$$

where z is fresh in $Vars_D$. Two applications of *Eliminate* (the only transformations which apply) yield

$$\langle f, ng\rangle, \langle z, g\rangle, \langle x, 4\rangle, \langle g4, 4\rangle,$$

which is unsolvable, because the last pair is. The only other possible transformation applying to the original pair is *Decompose*, which gives

$$\langle f, ng\rangle, \langle x, 4\rangle, \langle gx, 4\rangle.$$

Again, only *Eliminate* applies, yielding a system essentially the same as that obtained via the first sequence, and which contains the unsolvable pair $\langle g4, 4\rangle$. We may therefore conclude that the original system is unsolvable, in accordance with the fact that neither the identity function nor the function which is constantly four is strictly decreasing.

Of course, if we were to interpret D as denoting the (not strictly) decreasing real-valued functions on the reals, then we would want to be able to compute the function whose value is constantly four as a binding for g in Example 3.28. So while considering the original system Γ there as a typed system permits too many bindings for certain applications, a system supporting only constant declarations may permit too few. A calculus allowing arbitrary term declarations finds a middle road: in a signature with a term declaration assigning $K4$ to have sort D (for an appropriate K) Γ would have precisely the desired solutions.

4 Discussion

4.1 Implementation Issues

For implementation purposes, several refinements of $\mathcal{OSU}$ are possible. As noted in [Dou93], one may, for example, incorporate *Add Argument* into more generous versions of the *Head Narrow* transformations which supply arguments as needed, rather than treating it as a separate transformation. In addition, solved pairs $\langle x, M\rangle$ where x does not appear in the original system Γ to be solved can be deleted from $\mathcal{OSU}$ computations originating with Γ, and if a "relevant" set of variables is explicitly indicated for each system ([GM81], [GM85]), then the deletion of trivial pairs, which one would not want to carry along in an implementation, will pose no computational difficulties (such as might otherwise occur, for example, in resolution computations).

In an implementation of $OSUT$, we would require the sort D in clause (b) of *Split* to be maximal meeting the other requirements for application of this transformation. This eliminates some redundant branching in the solution search space by allowing only "most general" inference steps. In regular signatures, we can also require in clause (a) of *Split* that the least sort B of h be such that $\Delta \vdash \gamma^k(B) \leq A$, and in clause (c) that the least sort of a satisfy the conditions on D and E stated there. The restriction to regular signatures is not unreasonable; for example, Smolka, *et. al.* ([SNGM89]) maintain that the only non-regular signatures appearing in theorem proving practice are pathological.

4.2 Topics for Investigation

Investigations of combinator-based order-sorted higher-order unification algorithms can benefit from appropriate modifications of any advances for unification in $\mathcal{CL}$, such as the integration of additional combinators (*e.g.*, B and C) and the development of transformations, along the lines of those presented in [DJ92], accommodating first-order order-sorted equational theories into the higher-order paradigm. For automated deduction purposes, the development of techniques permitting the added expressiveness of Schmidt-Schauß-style sort declarations for arbitrary terms ([Sch89]), rather than just for constants, represents an important direction for future research. The development of a resolution calculus and a pre-unification algorithm for $\mathcal{CL}(\Sigma)$ and its future extensions will also prove crucial to any automated deduction applications.

Since simply typed higher-order unification is undecidable, so is order-sorted higher-order unification. Algorithm $\mathcal{OSU}$, which is complete, can therefore not be terminating in general. Huet ([Hue73], [Hue75]) observed, however, that for deduction purposes a higher-order computation need not always discover actual higher-order unifiers, and that *pre-unification*, or the detection of the possibility of higher-order unifiers, suffices. This observation has resulted in an irredundant (and therefore incomplete) pre-unification procedure which has proven instrumental in making higher-order resolution-like computations practicable.

We expect that our algorithm is suitable for order-sorted pre-unification as well as order-sorted C-unification. Although there exists at present a pre-unification algorithm for neither Dougherty's typed combinatory logic nor the order-sorted lambda calculus of [JK93], once detailed, any correspondence between pre-unification processes in the typed lambda and combinatory calculi, and/or pre-unification refinements of the algorithm in [JK93], would serve as a point of departure for investigating pre-unification in the order-sorted combinatory logic presented here. Such studies might determine, for example, whether or not there exist restrictions on Algorithm $\mathcal{OSU}$ or its typed counterpart making the induced pre-unification procedures, like Huet's, irredundant. Since any reasonable notion of pre-unification seems to require restricting attention to regular signatures (see [JK93] for a brief discussion), the refinements discussed in the previous subsection can be incorporated into a pre-unifying version of Algorithm $\mathcal{OSU}$ as well.

Finally, Huet's pre-unification procedure is finitely branching. Dougherty's combinator based unification algorithm is also finitely branching, by virtue of a typing scheme flexible enough to accommodate incompletely specified types. The calculus given here supports only completely specified sorts, and is therefore infinitely branching (because of sort non-determinism in the *Syz-Narrow* transformation), but it would be useful to be able to recover, if possible, finite branching by enriching our sort structure to allow incompletely specified sorts.

References

[ALMP84] P. B. Andrews, E. Longini-Cohen, D. Miller, and F. Pfenning. Automating Higher-order Logics. *Contemporary Mathematics* 29, pp. 169 – 192, 1984.

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

[Bre88] V. Breazu-Tannen. Combining Algebra and Higher-order Types. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, IEEE, pp. 82 – 90, 1988.

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

[CF58] H. B. Curry and R. Feys. *Combinatory Logic, Volume I*, North-Holland, Amsterdam, 1958.

[CG91] P.-L. Curien and G. Ghelli. Subtyping + Extensionality: Confluence of $\beta\eta$top reduction in $F_{\leq}$. In 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.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In *Handbook of Theoretical Computer Science B: Formal Methods and Semantics*, J. van Leeuwen, ed., North-Holland, Amsterdam, 1990.

[DJ92] D. J. Dougherty and P. Johann. A Combinatory Logic Approach to Higher-order E-unification. In *Proceedings of the Eleventh International Conference on Automated Deduction*, Springer-Verlag LNAI 607, pp. 79 – 93, 1992. Revised and expanded version submitted, *Theoretical Computer Science*.

[Dou93] D. J. Dougherty. Higher-order Unification via Combinators. Presented at the Fourth Workshop on Unification, University of Leeds, UK, 1990. *Theoretical Computer Science B*, 114, pp. 273 – 298, 1993.

[FP91] T. Freeman and F. Pfenning. Refinement Types for ML. In *Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation*, ACM, pp. 268 – 277, 1991.

[Gol81] W. Goldfarb. The Undecidability of the Second-order Unification Problem. *Theoretical Computer Science* 13, pp. 225 – 230, 1981.

[Gor85] M. Gordon. HOL: A Machine Oriented Formulation of Higher-order Logic. University of Cambridge, Computer Laboratory, Report 68, 1985.

[Gou66] W. E. Gould. A Matching Procedure for Omega-order Logic. Dissertation, Princeton University, 1966.

[Hay71] P. Hayes. A Logic of Actions. *Machine Intelligence* 6, pp. 495 – 520, 1971.

[Her71] J. Herbrand. Sur la Théorie de la Démonstration. In *Logical Writings*, W. Goldfarb, ed., Cambridge University Press, Cambridge, 1971.

[HS86] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-Calculus*. Cambridge University Press, 1986.

[Hue73] G. Huet. The Unidecidability of Unification in Third-order Logic. *Information and Control* 22, pp. 257 – 267, 1973.

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

[JK93] Unification in an Extensional Lambda Calculus with Ordered Function Sorts and Constant Overloading. SEKI-Report SR-93-14 (SFB), Universität des Saarlandes, Saarbrücken, Germany. Extended Abstract submitted, Twelfth International Conference on Automated Deduction (1994).

[Joh91] P. Johann. Complete Sets of Transformations for Unification Problems. Dissertation, Wesleyan University, 1991.

[Klo91] J. W. Klop. Term Rewriting Systems. In *Handbook of Logic in Computer Science*, Volume 1, S. Abramsky and D. M. Gabbay, eds., Oxford University Press, Oxford, 1991.

[KP93] M. Kohlhase and F. Pfenning. Unification in a λ-calculus with Intersection Types. To appear in *Proceedings of the International Logic Programming Symposium*, 1993.

[LSBB92] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High Performance Theorem Prover. *Journal of Automated Reasoning* 8, pp. 183 – 212, 1992.

[Lus92] E. L. Lusk. Controlling Redundancy in Large Search Spaces: Argonne-style Theorem Proving Through the Years. In Springer-Verlag LNAI 624, pp. 96 – 106, 1992.

[Mil91] D. Miller. A Logic Programming Language with Lambda Abstraction, Function Variables, and Simple Unification. *Journal of Logic and Computation* 2, pp. 497 – 536, 1986.

[Mit91] J. C. Mitchell. Type Inference with Simple Types. *Journal of Functional Programming* 1, pp. 245 – 285, 1991.

[MM82] A. Martelli and U. Montanari. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems* 4, pp. 258 – 282, 1982.

[NQ92] T. Nipkow and Z. Qian. Reduction and Unification in Lambda Calculi with Subtypes. In Springer-Verlag LNAI 607, pp. 66 – 78, 1992.

[Obe62] A. Oberschelp. Untersuchung zur Mehrsortigen Quantorenlogik. *Mathematische Annalen* 145, pp. 297 – 333, 1962.

[OS89] H.-J. Ohlbach and J. Siekmann. The Markgraph Karl Resolution Procedure. In *Computational Logic — Essays in Honor of Alan Robinson*, J.-L. Lassez and G. Plotkin, eds., MIT Press, pp. 41 – 112, 1989.

[Pau90] L. C. Paulson. Isabelle: The Next 700 Theorem Provers. In *Logic and Computer Science*, P. Odifreddi, ed., Academic Press, 1990.

[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.

[Pla93] D. A. Plaisted. Equational Reasoning and Term Rewriting Systems. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, Volume 1, D. Gabbay and J. Siekmann, eds., Oxford University Press, Oxford. To appear.

[Qia90] Z. Qian. Higher-order Order-sorted Algebras. In Springer-Verlag LNCS 463, pp. 86 – 100, 1990.

[Qia91] Z. Qian. Extensions of Order-sorted Algebraic Specifications: Parameterization, Higher-order Functions and Polymorphism. Dissertation, Universität Bremen, 1991.

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

[Sil94] R. Silverman. An Implementation of Higher-order Unification via Combinatory Logic. Master's Thesis, Wesleyan University. To appear, 1994.

[Sti90] M. E. Stickel. A Prolog Technology Theorem Prover. In *Proceedings of the Tenth International Conference on Automated Decuction*, Springer-Verlag LNAI 449, pp. 673 – 674, 1990.

[Wal87] C. Walther. *A Many-sorted Calculus Based on Resolution and Paramodulation.* Pitman and Kaufman, 1987.

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