HIGHER ORDER UNIFICATION REVISITED: COMPLETE SETS OF TRANSFORMATIONS

Wayne Snyder[1] and Jean H. Gallier[2]

[1]Computer Science Department, Room 280
Boston University
111 Cummington Street
Boston, MA 02215

[2]Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

November 13, 2012

This research was partially supported by the National Science Foundation under Grant No. DCR-86-07156 and by ONR under Grant No. N00014-88-K-0593.

HIGHER ORDER UNIFICATION REVISITED: COMPLETE SETS OF TRANSFORMATIONS

Wayne Snyder[1] **and Jean H. Gallier**[2]

Abstract: In this paper, we reexamine the problem of general higher-order unification and develop an approach based on the method of transformations on systems of terms which has its roots in Herbrand's thesis, and which was developed by Martelli and Montanari in the context of first-order unification. This method provides an abstract and mathematically elegant means of analyzing the invariant properties of unification in various settings by providing a clean separation of the logical issues from the specification of procedural information. Our major contribution is three-fold. First, we have extended the Herbrand-Martelli-Montanari method of transformations on systems to higher-order unification and pre-unification; second, we have used this formalism to provide a more direct proof of the completeness of a method for higher-order unification than has previously been available; and, finally, we have shown the completeness of the strategy of eager variable elimination. In addition, this analysis provides another justification of the design of Huet's procedure, and shows how its basic principles work in a more general setting. Finally, it is hoped that this presentation might form a good introduction to higher-order unification for those readers unfamiliar with the field.

1 Introduction

Higher-order unification is a method for unifying terms in the Simple Theory of Types [6], that is, given two typed lambda-terms e_1 and e_2, finding a substitution σ for the free variables of the two terms such that $\sigma(e_1)$ and $\sigma(e_2)$ are equivalent under the conversion rules of the calculus. This problem is fundamental to automating higher-order reasoning, as convincingly shown for example in the automated proof of Cantor's Theorem (that there is no surjection from a set to its powerset) found by the TPS system [3], where the higher-order unification procedure finds a term which corresponds to the diagonal set $\{a \in A \mid a \notin f(a)\}$ used in the standard proof (for details, see [3]). Higher-order unification has formed the basis for generalizations of the resolution principle to second-order logic [7, 40] and general ω-order logic [24, 36, 41] (but see also [1]), the generalization of the method of matings [2] to higher-order [4, 3, 31, 37], higher-order logic programming in the language λProlog [32, 35], a means for providing flexible implementations of logical inference rules in theorem provers [12,36], program synthesis, transformation, and development [27, 20, 21, 33, 39], and also has applications to type inferencing in polymorphic languages [38], computational linguistics [34], and certain problems in proof theory concerning the lengths of proofs [10]. Higher-order unification was studied by a number of researchers [7, 17, 18, 19, 40, 41] before Huet [25, 26] made a major contribution in showing that a restricted form of unification, called preunification, is sufficient for most refutation methods and in defining a method for solving this restricted problem which is used by most current higher-order systems.

In this paper, we reexamine the problem of general higher-order unification and develop an approach based on the method of transformations on systems of terms which has its roots in Herbrand's thesis, and which was developed by Martelli and Montanari [30] in the context of first-order unification. This method provides an abstract and mathematically elegant means of analyzing the invariant properties of unification in various settings by providing a clean separation of the logical issues from the specification of procedural information. The set of transformations for higher-order unification is developed from an analysis of the manner in which substitution and β-reduction make two terms identical, and shows clearly the relationship between first-order unification, higher-order preunification, and general higher-order unification. Our major contribution is three-fold. First, we have extended the Herbrand-Martelli-Montanari method of transformations on systems to higher-order unification and pre-unification; second, we have used this formalism to provide a more direct proof of the completeness of a method for higher-order unification than has previously been available; and, finally, we have shown the completeness of the strategy of eager variable elimination, which eliminates redundant computations while maintaining the ability to find complete sets of unifiers. In addition, this analysis provides another justification of the design of Huet's procedure, and shows how its basic principles work in a more general setting. Finally, it is hoped that this presentation might form a good introduction to higher-order unification for those readers unfamiliar with the field. To this end, and in order to motivate the use of transformations for higher-order unification, in the remainder of this introduction we provide an overview of our approach.

The method of transformations for solving unification problems is much like the well-known method used for solving systems of linear equations known as Gaussian elimination. In Gaussian elimination, the original system of equations is transformed step by step (by variable elimination) into a solved system, that is, a system whose solution is obvious. Similarly, a unification problem is a set $\{\langle u_1, v_1\rangle, \ldots, \langle u_n, v_n\rangle\}$ of pairs of terms (sometimes called a *disagreement set*) to be (jointly) unified. (We consider these pairs to be *unoriented*.) The method of transformations consists of applying simple transformations, some akin to variable elimination, until a "solved" system S' is obtained whose solution is obvious (in a sense to be made precise below).

Gaussian elimination and first-order unification are somewhat similar. For example, the transformations for first-order unification given in Section §3, like Gaussian elimination, must terminate and hence the existence of solutions is decidable. Also, these transformations preserve the set of solutions as an invariant, just as in Gaussian elimination the variable elimination step preserves solutions; and in both the set of solutions is either empty, a singleton, or infinite. But in the higher-order case the analogy breaks down. For example, unlike Gaussian elimination, it is undecidable whether a higher-order system has unifiers, and the transformations do not terminate in general. Also, the transformations used for higher-order unification do not necessarily preserve the set of solutions. In general, if a system S' is derived from a system S, it can only be claimed that the set of unifiers of S' is a subset of the set of unifiers of S. Thus, we face a completeness problem: we have to show that every unifier of S will be produced as the obvious solution of some system S' derivable from S. In fact, it is practically impossible to require that every unifier of S be produced, and normally we are only interested in whether a complete set of unifiers can be enumerated using the transformations. Roughly speaking, a complete set of unifiers for S is a set of unifiers for S from which every unifier for S can be generated.

Thus the interesting issue is in finding natural sets of transformations which present in an abstract form the fundamental operations of unification, but which are complete in this sense. In order to introduce the notion of higher-order unification, we shall first demonstrate the full method in the first-order case, and then sketch what changes need to be made to deal with higher-order terms. This will hopefully provide the necessary intuition for the more detailed treatment in the remainder of the paper.

Suppose we wish to find a unifier (if possible) for the two terms $f(x, f(h(x, gx), x'))$

and $f(x, f(h(fy, z), y'))$. Now any substitution which unifies these terms can not affect the topmost function symbol f, and so it is easy to see that a substitution θ unifies the terms if and only if it pairwise unifies each of the immediate subterms. For example, θ unifies the system

$$\{\langle f(x, f(h(x, gx), x')),\ f(x, f(h(fy, z), y'))\rangle\}$$

iff it unifies

$$\{\langle x, x\rangle, \langle f(h(x, gx), x'), f(h(fy, z), y')\rangle\}.$$

In general, we may define a transformation on systems which we call *term decomposition*:

$$\{\langle f(u_1, \ldots, u_n), f(v_1, \ldots, v_n)\rangle\} \cup S \implies \{\langle u_1, v_1\rangle, \ldots, \langle u_n, v_n\rangle\} \cup S,$$

where S is any system (possibly empty). After two more iterations of this transformation, we have

$$\{\langle x, x\rangle, \langle x, fy\rangle, \langle gx, z\rangle, \langle x', y'\rangle\}.$$

Now in this system, it is clear that the pair $\langle x, x\rangle$ is in fact already unified, and contributes no information about possible solutions, since *any* substitution unifies a pair $\langle u, u\rangle$ for some term u. Thus we may define a transformation which simply removes such trivial pairs:

$$\{\langle u, u\rangle\} \cup S \implies S.$$

In our example, we may derive the new system

$$\{\langle x, fy\rangle, \langle gx, z\rangle, \langle x', y'\rangle\}.$$

These two transformations simplify a system (by reducing the total number of symbols in the whole system) but do not in any way change the set of solutions; hence the set of solutions is *invariant* under the transformations. But it is not yet obvious what the set of solutions is. The reader may check for example that $[fy/x, gfy/z, x'/y']$, $[fy/x, gfy/z, y'/x']$, and $[fha/x, gfha/z, ha/y, a/x', a/y']$ are all unifiers of the system. In each of these however, the binding made for x has the form ft for some term t, since if a substitution unifies the pair $\langle x, fy\rangle$ then the binding for x must have f as a top symbol. In this case, we may provide a partial binding for x (since we do not yet know the entire binding, but only the top symbol) by transforming the previous system into a new one which contains this partial binding:

$$\{\langle x, fx_1\rangle, \langle x, fy\rangle, \langle gx, z\rangle, \langle x', y'\rangle\}.$$

Now we may eliminate the variable x from the rest of the system by replacing it by fx_1, i.e., by applying the substitution $[fx_1/x]$. After applying decomposition once more, we get the system

$$\{\langle x, fx_1\rangle, \langle x_1, y\rangle, \langle gfx_1, z\rangle, \langle x', y'\rangle\}.$$

In general, we may define an *imitation rule* for partially solving variables in systems: If x does not occur in the term $f(t_1, \ldots, t_n)$ then we have:

$$\{\langle x, f(t_1, \ldots, t_n)\rangle\} \cup S \implies \{\langle x, f(y_1, \ldots, y_n)\rangle, \langle f(y_1, \ldots, y_n), f(t_1, \ldots, t_n)\rangle\} \cup S',$$

where $y_1, \ldots, y_n$ are *new* variables occurring nowhere else, and S' is the result of replacing every occurrence of x in S by the partial binding $f(y_1, \ldots, y_n)$. Note that if x were to occur in the term $f(t_1, \ldots, t_n)$ then the system would not be unifiable.

The point of the imitation rule is that we find a partial solution for a variable x, and then *solve* x partially by substituting the partial solution for the remaining occurrences of x, thus reduced the problem of finding a binding to solving for the new variables in the partial binding for x. In general, if we transform a system using the rule

$$\{\langle x, t\rangle\} \cup S \implies \{\langle x, t\rangle\} \cup S[t/x],$$

where x is a variable occurring in S but not occurring in t and $S[t/x]$ represents the result of replacing every occurrence of x in S by t, then, as in Gaussian Elimination, we have *solved* the system for the variable x; hence this transformation is called *variable elimination*. As in the case of our first two transformations, the set of solutions is invariant under variable elimination. (Imitation does not preserve solutions, since it potentially introduces new variables.) In our example, we can eliminate the variable x_1 to obtain the system

$$\{\langle x, fy\rangle, \langle x_1, y\rangle, \langle gfy, z\rangle, \langle x', y'\rangle\}.$$

If we say that a pair $\langle x, t\rangle$ is in *solved form* in a system if x does not occur in the rest of the system and does not occur in t, then clearly the last system is *solved* in the sense that all its pairs are in solved form.

The basic idea of the transformation method as represented by these four transformations is to successively build up bindings for variables and simplify the systems produced by decomposing and eliminating trivial pairs. The intent is to transform a unification problem into a solved system, since a solved system $\{\langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle\}$ gives explicitly the bindings of a unifying substitution $[t_1/x_1, \ldots, t_n/x_n]$. In our example, we have the unifying substitution $[fy/x, y/x_1, gfy/z, x'/y']$, which, since we are only interested in bindings made for the variables in the original system, may be restricted to the form $[fy/x, gfy/z, x'/y']$. (It is interesting to note that we could also have extracted the substitution $[fy/x, gfy/z, y'/x']$.) This set of four transformations can be easily shown to be *sound* in the sense that if $S \implies S'$ and θ unifies S', then θ unifies S; thus the method is correct since any solution found will unify the original system. Showing that the method is *complete* is harder, since we must show that for *any* unifier θ of the original system S, we can find a sequence of transformations $S \stackrel{*}{\Longrightarrow} S'$ resulting in a solved form S' such that the substitution $\sigma_{S'}$ extracted from S' is more general than θ (over the set of variables in S). The intuitive reason that we can find *mgu*'s (and, more generally, we can find complete sets in the higher-order case) using this method is that imitation and variable elimination are capable of incrementally building up the bindings in the unifying substitution just as much as is necessary to unify the original system. The reader may check for example that each of the substitutions found above for S is more general than *any* unifier of the original system, i.e., they are *most general unifiers* or *mgu*'s.

There are several important things to note about this method. The first is that it is a *non-deterministic* set of abstract operations for unification; we can think of it as a set of *inference rules* for unification. This removal of control and data structure specification allows us to examine the fundamental properties of the problem more clearly. The notion of completeness is also non-deterministic, since we show only that for an arbitrary unifier θ there is *some* sequence of transformations which produces a unifier more general than θ. In order to design a practical procedure, we would have to specify data structures and a search strategy to explore the search tree of possible transformation paths. The second point is that if we need to find *all* unifiers, then in the case of a pair of two variables we would need to apply imitation by 'guessing' a partial binding for one of the variables or by guessing an arbitrary variable as a binding. For example, to find the unifier $[fz/x, fz/y]$ of the system $\{\langle x, y\rangle\}$ we would have to guess the function symbol f in the imitation pair $\langle x, fy_1\rangle$, then imitate for y, and finally guess that y_1 is bound to z. This is clearly a problem for implementation, but it turns out that for unification in theorem proving we need only find most general solutions, and so in the first-order case we can avoid this guessing by using variable elimination on such pairs. In fact, if we are interested in stopping as soon as the possibility of unification is detected, without necessarily transforming the system into a fully solved form, we may define the notion of a *presolved* system as one consisting of either solved pairs, as above, or pairs consisting of two variables, and stop the transformation process as soon as a presolved form is reached. For example, the system

$$\{\langle x, a\rangle, \langle y, fz\rangle, \langle x', y'\rangle, \langle x', z'\rangle, \langle z', x''\rangle\}$$

is presolved. It turns out that it is always possible to unify such systems, by applying variable elimination to the variable-variable pairs which are not yet solved. This shows that we need never apply the imitation rule to a variable-variable pair, since such pairs can always be eliminated using variable elimination; in the higher-order generalization of this case, this is not true, as we shall see, and the notion of presolved forms is crucial. It is interesting that in first-order, the presence of variable-variable pairs is the reason that *mgu*'s are, strictly speaking, not unique; recall that in our example above, we had two choices about the extraction of a binding from the pair $\langle x', y' \rangle$, resulting in the two *mgu*'s $[fy/x, gfy/z, x'/y']$ and $[fy/x, gfy/z, y'/x']$.

The other interesting point is that in the first-order case we have presented, we can in fact have a complete set of transformation rules if we exclude the imitation rule, i.e., if we find bindings by simply eliminating a variable all at once if we find a pair $\langle x, t \rangle$ where x does not occur in t. In our previous example, we could have 'short-circuited' the sequence of transformations by immediately eliminating the variable x to produce a solved form:

$$\begin{aligned} & \{\langle f(x, f(h(x, gx), x')),\ f(x, f(h(fy, z), y'))\rangle\} \\ \overset{*}{\Longrightarrow}\ & \{\langle x, fy\rangle, \langle gx, z\rangle, \langle x', y'\rangle\} \\ \Longrightarrow\ & \{\langle x, fy\rangle, \langle gfy, z\rangle, \langle x', y'\rangle\}. \end{aligned}$$

In Section §3 we shall develop this improved method in detail; the completeness of these transformations is particularly easy to prove. In the higher-order transformations, we can not do away with the imitation rule completely, but we can use variable elimination to more efficiently build up bindings whenever possible without sacrificing completeness.

The method we have just sketched can be generalized to higher-order unification with relatively few changes. The most important differences have to do with the imitation rule and the generalization of the notion of a partial binding to higher-order substitutions. Consider the system $S = \{\langle F(f(a)), f(F(a))\rangle\}$, where F is a variable of functional type (say $int \to int$). It is easily seen that $\theta = [\lambda x.f(x)/F]$ is a unifier for S, since

$$\theta(F(f(a))) = (\lambda x.f(x))\, f(a) \longrightarrow_\beta f(f(a)) \longleftarrow_\beta f((\lambda x.f(x))a) = \theta(f(F(a))),$$

where $\longrightarrow_\beta$ denotes β-reduction. (This is not the only solution, for example the reader may check that any substitution in the form $[\lambda x.\, f^k(x)/F]$ for $k \geq 0$ is also a unifier.) This time, it is a little more tricky to build up θ using partial bindings. In the first-order case, we generate bindings of the form $[f(y_1, \ldots, y_n)/x]$, where $y_1, \ldots, y_n$ are first-order variables. The generalization (roughly) is to consider partial bindings of the form

$$[\lambda x_1 \ldots x_k.\, a\big(Y_1(x_1, \ldots, x_k), \ldots, Y_n(x_1, \ldots, x_k)\big)/F],$$

where $Y_1, \ldots, Y_n$ are some higher-order variables of appropriate types and a is an atom (i.e., a constant, a free variable, or a bound variable x_i for $1 \leq i \leq k$). The idea is that we have to generalize the partial binding $f(y_1, \ldots, y_n)$ to higher-order, and so the top function symbol a may be a variable, and the variables $y_1, \ldots, y_n$ and the term itself may be of functional type; furthermore, each y_i must be generalized to a term $Y_i(x_1, \ldots, x_n)$ since the subterms of the binding may be some function of the bound variables $x_1, \ldots, x_n$. A further level of complexity is introduced by the constraints imposed by the type structure. The notion of higher-order partial bindings will be carefully defined in Section §4.

The imitation rule must accommodate this more complex form of partial binding. In the first order case, we applied imitation to a pair $\langle x, f(t_1, \ldots, t_k)\rangle$ using a partial binding $f(y_1, \ldots, y_n)$; in the higher-order case we must be able to apply imitation to pairs such as $\langle F(f(a)), f(F(a))\rangle$ to partially solve for F. A partial binding for F which imitates the symbol f in this case would have the form $\lambda x.\, f(Y(x))$, so that we would transform the system $\{\langle F(f(a)), f(F(a))\rangle\}$ into

$$\{\langle F, \lambda x.\, f(Y(x))\rangle, \langle f(Y(f(a))), f(f(Y(a)))\rangle\}$$

using the imitation rule; note that we have performed β-reduction after applying the substitution $[\lambda x.\, f(Y(x))/F]$. After decomposition we have

$$\{\langle F, \lambda x.\, f(Y(x))\rangle, \langle Y(f(a)), f(Y(a))\rangle\}.$$

Unfortunately, the imitation rule alone is not sufficient for building up bindings in higher-order unification. This is easy to see in considering the subproblem of finding a partial binding for Y, which is exactly the problem we faced with F; simply continuing to imitate will produce an infinite sequence of transformations. The problem arises because higher-order terms may have variables as their top-most symbol and so we must allow bindings such as $\lambda x.x$ to be found by our transformations. If we abbreviate a lambda binder $\lambda x_1 \ldots x_k$ into the form $\lambda\overline{x_k}$, the new rule for finding partial bindings has (roughly) the form:

$$\begin{aligned}&\{\langle \lambda\overline{x_k}.F(u_1, \ldots, u_n), \lambda\overline{x_k}.a(v_1, \ldots, v_m)\rangle\} \cup S \Longrightarrow \\ &\qquad\qquad \{\langle F, t\rangle\} \cup \sigma(\{\langle \lambda\overline{x_k}.F(u_1, \ldots, u_n), \lambda\overline{x_k}.a(v_1, \ldots, v_m)\rangle\} \cup S),\end{aligned}$$

where a is a function symbol, constant, or variable (either free or bound), and where t is either an imitation binding, i.e., $t = \lambda\overline{y_n}.\, a(Y_1(\overline{y_n}), \ldots, Y_m(\overline{y_n}))$, or a projection binding, i.e., $t = \lambda\overline{y_n}.\, y_i(Y_1(\overline{y_n}), \ldots, Y_q(\overline{y_n}))$ for some i, $1 \leq i \leq n$, and $\sigma = [t/F]$ (after applying σ, we also reduce the resulting terms to their normal form using β-conversion). For example, we can transform the system $\{\langle F(f(a)), f(F(a))\rangle\}$ by adding a projection binding to get

$$\{\langle F, \lambda x.x\rangle, \langle F(f(a)), f(F(a))\rangle\}$$

and then applying the substitution $[\lambda x.x/F]$ and β-reducing to get

$$\{\langle F, \lambda x.x\rangle, \langle f(a), f(a)\rangle\}.$$

After removing the trivial pair, gives us the solved system $\{\langle F, \lambda x.x\rangle\}$. The reader may check that a similar projection for the variable Y in our example above results in the solved system $\{\langle F, \lambda x.\, f(x)\rangle, \langle Y, \lambda x.x\rangle\}$.

Besides the more complicated form of the rule which finds partial bindings, there are several other things which make the higher-order case more complex than the first-order method outlined above. For example, unification is defined modulo the conversion rules of the lambda calculus, so that we shall have to carefully justify our method from an analysis of the means by which substitution and subsequent β-reduction makes terms equal. Another complication is that higher-order unification is undecidable in general and most general unifiers do not necessarily exist. The latter problem is solved by defining the notion of a *complete set of unifiers* (which may be infinite!) and the former simply prevents our transformation process from terminating in general. The notion of completeness therefore must be defined in terms of complete sets of unifiers; in fact, the completeness proof is not much harder than in first-order.

A final important difference from the first-order case has to do with the higher-order equivalent of a variable-variable pair of terms, namely, a pair of terms with variables at their heads, e.g. $\langle \lambda x.\, F(a,x), \lambda x.\, G(x,a)\rangle$. (These are called *flexible-flexible* pairs.) Unfortunately, it is not possible to avoid the arbitrary 'guessing' of bindings discussed above and preserve completeness, and so the search tree for unifiers may be infinitely branching. This posed an insurmountable problem for implementation until Huet showed that in the context of a refutation method, it is usually only necessary to determine the possibility of unification, and since such flexible-flexible pairs are always unifiable, we can stop after finding a *presolved* form. This restricted form of unification is termed *preunification*.

After reviewing a number of basic definitions and results in the next section, we then present the transformation method in detail for the first-order case, showing how decomposition, variable elimination, and the removal of trivial pairs gives us a method for finding most general unifiers. In Section §4 we extend this to the higher-order case, first presenting the fundamental concepts of higher-order unification, then giving the set of transformations for higher-order unification, and next proving the soundness and completeness of the set. Finally, we show how Huet's method for pre-unification from [26] can be described as a special case of this set.

2 Preliminaries

In order that this paper be self-contained, we present here a number of basic definitions and results related to the typed lambda calculus, including a detailed treatment of the notion of a substitution. Our notation and approach is basically consistent with [5], [13], [23], and

[26].

Definition 2.1 Given a set $\mathcal{T}_0$ of *base types* (e.g., such as *int*, *bool*, etc.) we define the set of types $\mathcal{T}$ inductively as the smallest set containing $\mathcal{T}_0$ and such that if $\alpha, \beta \in \mathcal{T}$, then $(\alpha \to \beta) \in \mathcal{T}$.

The type $(\alpha \to \beta)$ is that of a function from objects of type α to objects of type β. We assume that the type constructor $\to$ associates to the right, and we shall often write type expressions such as $(\alpha_1 \to (\alpha_2 \to \ldots (\alpha_n \to \beta) \ldots))$ in the form $\alpha_1, \ldots, \alpha_n \to \beta$, with β an arbitrary type.

Definition 2.2 Let us assume given a set Σ of symbols, which we call *function constants*, each symbol f having a unique type $\tau(f)$ from $\mathcal{T}$. For each type $\alpha \in \mathcal{T}$, we assume given a countably infinite set of variables of that type, denoted V_α, and let $V = \bigcup_{\tau \in \mathcal{T}} V_\tau$. Furthermore, let the set of *atoms* $\mathcal{A}$ be defined as $V \cup \Sigma$. The set $\mathcal{L}$ of lambda-terms is inductively defined as the smallest set containing $\mathcal{A}$ and closed under the rules of function application and lambda-abstraction, namely,

(i) If $e_1 \in \mathcal{L}$ has type $\alpha \to \beta$, and $e_2 \in \mathcal{L}$ has type α, then $(e_1 e_2)$ is a member of $\mathcal{L}$ of type β.

(ii) If $e \in \mathcal{L}$ has type β and $x \in V_\alpha$ then $(\lambda x.\, e)$ is a member of $\mathcal{L}$ of type $\alpha \to \beta$.

We shall denote the type of a term e by $\tau(e)$.

By convention, application associates to the left, so that a term $(\ldots((e_1 e_2) e_3) \ldots e_n)$ may be represented as $(e_1 e_2 \ldots e_n)$. In general we represent a sequence of lambda abstractions $\lambda x_1.\, (\lambda x_2.\, (\ldots (\lambda x_n.\, e) \ldots))$ in the form $\lambda x_1 \ldots x_n.\, e$, where e is either an application or an atom. We shall often drop superfluous parentheses when there is no loss of clarity, and will use square brackets if necessary; also we follow the convention that the dot includes as much right context as possible in the scope of its binder, so that, e.g., a term $\lambda x.\, stu$ is to be interpreted as $(\lambda x.\, ((st)u))$.

Definition 2.3 In a term $\lambda x_1 \ldots x_n.\, e$ where e is either an application or an atom, we call e the *matrix* of the term, the object $\lambda x_1 \ldots x_n$ is the *binder* of the term, and the occurrences of the variables are called *binding occurrences* of these variables. We define the *size* of a term u, denoted $|u|$, as the number of atomic subterms of u. A variable x occurs *bound* in a term e if e contains some subterm of the form $\lambda x.\, e'$, in which case the term e' is called the *scope* of this binding occurrence of x. A variable x occurs *free* in e if it is a subterm of e but does not occur in the scope of a binding occurrence of x. The set of free variables of a term e is denoted by $FV(e)$.

Definition 2.4 The *order* of a term or a variable is just the order of its type, where the *order* of a type φ is defined as

$$Ord(\varphi) = \begin{cases} 1, & \text{if } \varphi \in \mathcal{T}_0\,; \\ max(Ord(\alpha)+1, Ord(\beta)), & \text{if } \varphi \ = \ \alpha \to \beta\,. \end{cases}$$

A *language of order* n is one which allows contants of order at most $n+1$ and free and bound variables of order at most n.

This formalizes the usual convention that a first-order term denotes an individual, a term of second order denotes a function on individuals, etc.

Convention: In what follows we denote types by α, β, γ, and φ; constants of primitive type by b and c; constants of functional type by f, g, and h; variables of arbitrary type by x, y, and z, and arbitrary atoms by a. We shall often represent free variables of functional type by the letters F, G, H, and Y. Lambda terms will be denoted by e, r, s, t, u, v, and w. We shall, in the interest of clarity, omit type information whenever possible, since it is inferrable from context in the cases we consider.

The 'computation rules' of the lambda calculus are as follows.

Definition 2.5 Let $u[t/x]$ denote the result of replacing each free occurrence of x in u by t, and $BV(t)$ be the set of bound variables in t. We have three rules of *lambda conversion.*

(i) (α-conversion) If $y \notin FV(t) \cup BV(t)$, then

$$(\lambda x.\, t) \ \succ_\alpha \ (\lambda y.\, (t[y/x])).$$

(ii) (β-conversion)

$$((\lambda x.\, s)\, t) \ \succ_\beta \ s[t/x].$$

(iii) (η-conversion)[1] If $x \notin FV(t)$, then

$$(\lambda x.\, (t\, x)) \ \succ_\eta \ t.$$

The term on the left side of each of these rules is called a *redex*. A term t which contains no β-redices is called a *β-normal form*, and η-normal forms and $\beta\eta$-normal forms are defined similarly. If we denote by $e[s]$ a lambda term with some distinguished occurrence of a subterm s, then let $e[t]$ denote the result of replacing this single subterm by the term t, where $\tau(s) = \tau(t)$. We define the relation $\longrightarrow_\alpha$ as

$$e[s] \longrightarrow_\alpha e[t] \quad \text{iff} \quad s \succ_\alpha t,$$

[1] This rule is a special case of the the axiom of extensionality, viz., $\forall f, g(\forall x(f(x) = g(x)) \Longrightarrow f = g)$, which asserts that two functions are equal if they behave the same on all arguments, regardless of their syntactic representation.

and similarly for $\longrightarrow_\beta$ and $\longrightarrow_\eta$. We define $\longrightarrow_{\beta\eta}$ as $\longrightarrow_\beta \cup \longrightarrow_\eta$. We also define the symmetric closure $\longleftrightarrow$, the transitive closure $\xrightarrow{+}$, and the symmetric, reflexive, and transitive closure $\overset{*}{\longleftrightarrow}$ of each of these relations in the obvious fashion. The relations $\overset{*}{\longleftrightarrow}_\beta$, $\overset{*}{\longleftrightarrow}_\eta$, and $\overset{*}{\longleftrightarrow}_{\beta\eta}$ are called β-, η-, and $\beta\eta$-*equivalence* respectively.

It is easy to show that the type of a lambda term is preserved under these rules of lambda conversion.

Definition 2.6 We say that s *is substitutible for* x *in* t if, for every subformula $\lambda y.\, t'$ of t, if $y \in FV(s)$ then $x \notin FV(t')$.

The motivation for this notion is that no free variable capture will take place if s is substituted for x in t'. (The problem with this *free variable capture* is that it violates the fundamental meaning of scope and the binding of variables; in [5], for the untyped calculus it is shown that if this is allowed, the calculus becomes inconsistent in the sense that any two terms are equivalent.) In the β-conversion rule, in the pathological case that s is not substitutible for x in t, i.e., x occurs in t in the scope of some binding occurrence of a variable which is free in s, then there is always a sequence $(\lambda x.\, t)\, s \xrightarrow{+}_\alpha (\lambda x.\, t')\, s \longrightarrow_\beta t'[s/x]$, where s is substitutible for x in t'. Thus, for simplicity and without loss of generality we adopt the following assumption.

Convention: We assume in the following that in the set of terms being discussed, the set of all free variables is distinct from the set of all bound variables. (This allows us to be 'naive' in our use of β-conversion and substitution; for another approach, see [23].) In fact, in the rest of this paper, all comparisons of lambda terms are modulo α-conversion, which will allow us to represent lambda binders using 'generic' variables $x_1, \ldots, x_k$ unless confusion would result. By abuse of notation, using this naive approach and following our representation of a sequence of lambda abstractions as a term $\lambda x_1 \ldots x_k.\, u$, we shall consider the conversion of redices involving such terms as a *single* reduction step instead of k steps, e.g.,

$$(\lambda x_1 \ldots x_k.\, u)\, v_1 \ldots v_k \longrightarrow_\beta u[v_1/x_1, \ldots, v_k/x_k]$$

instead of $(\lambda x_1 \ldots x_k.\, u)\, v_1 \ldots v_k \xrightarrow{k}_\beta u[v_1/x_1, \ldots, v_k/x_k]$.

Definition 2.7 The calculus which admits only the β-rule as a computation rule we call the *typed* β*-calculus* and the calculus which also admits the η-rule is called the *typed* $\beta\eta$*-calculus*.

In this paper, we wish to give an abstract method for higher-order unification which presents the fundamental logical issues as clearly as possible, and for this purpose we feel it is sufficient to develop the notion of unification of terms in the typed $\beta\eta$-calculus. This is a natural assumption in practice, and all higher-order theorem proving systems known to the authors use this weak form of extensionality. The reader interested in the details of the non-extensional case may consult [26].

Two of the major results concerning this calculus are the following.

Theorem 2.8 (Strong Normalization) Every sequence of $\beta\eta$-reductions is finite.

Theorem 2.9 (Church-Rosser Theorem) If $s \overset{*}{\longleftrightarrow}_{\beta\eta} t$ for two lambda terms s and t, then there must exist some term u such that $s \overset{*}{\longrightarrow}_{\beta\eta} u \overset{*}{\longleftarrow}_{\beta\eta} t$.

(Proofs of these may be found in [23].) Each of these theorems remains true when restricted to just η-conversion or just β-conversion. One of the important consequences of these two results is that for each term t there exists a unique (up to α-conversion) term t' such that $t \overset{*}{\longrightarrow}_{\beta\eta} t'$ with t' in $\beta\eta$-normal form, and similarly for the restriction to just β- or just η-reduction. Another consequence is that the β-, η-, or $\beta\eta$-equivalence of two arbitrary terms may be decided by checking if the corresponding normal forms of the two terms are equal. For example, if we denote the unique β-normal form of a term t by $t\!\downarrow$, then $s \overset{*}{\longleftrightarrow}_{\beta} t$ iff $s\!\downarrow = t\!\downarrow$.

Convention: We shall in general assume that terms under discussion are in β-normal form unless otherwise stated. In particular, each term in β-normal form may be represented in the form $\lambda x_1 \dots x_n(a\, e_1 \dots e_m)$, where the *head* a is an atom, i.e., a is either a function constant, bound variable, or some variable free in this term, and the terms $e_1, \dots, e_m$ are in the same form. By analogy with first-order notation, such a term will be denoted $\lambda x_1 \dots x_n.\, a(e_1, \dots, e_m)$. As an abbreviation, we represent lambda terms using something like a 'vector' notation for lists, so that $\lambda x_1 \dots x_n.\, e$ will be represented by $\lambda \overline{x_n}.\, e$. Furthermore, this principle will be extended to lists of terms, so that $\lambda \overline{x_n}.\, f(e_1, \dots, e_m)$ will be represented as $\lambda \overline{x_n}.\, f(\overline{e_m})$, and we shall even sometimes represent a term such as

$$\lambda \overline{x_k}.\, a(y_1(\overline{x_k}), \dots, y_n(\overline{x_k}))$$

in the form $\lambda \overline{x_k}.\, a(\overline{y_n(\overline{x_k})})$.

Definition 2.10 A term whose head is a function constant or a bound variable is called a *rigid* term; if the head is a free variable it will be called a *flexible* term. (For example, the term $\lambda x.\, F(\lambda y.\, y(x,a), c)$ is flexible, but both of its immediate subterms are rigid.)

As remarked above, we consider in this paper only the problem of unifying terms in the $\beta\eta$-calculus, and since our analysis proceeds by examining the manner in which substitution and subsequent β-reduction makes two terms identical, we need not explicitly consider the role of η-reduction. The formal justification for this is given by the following result.

Lemma 2.11 For any two terms s and t, we have $s \overset{*}{\longrightarrow}_{\beta\eta} t$ iff there exists a term u such that $s \overset{*}{\longrightarrow}_{\beta} u \overset{*}{\longrightarrow}_{\eta} t$.

(For a proof see [5].) As a consequence, we can decide $\beta\eta$-equivalence by reducing terms to their β-normal forms, and then testing for η-equivalence, that is, $s \overset{*}{\longleftrightarrow}_{\beta\eta} t$ iff $s{\downarrow} \overset{*}{\longleftrightarrow}_{\eta} t{\downarrow}$. This allows us to 'factor out' η-conversion, by considering only η-equivalence classes of terms. We shall use the following means of representing such classes by canonical representatives (due to [26]).

Definition 2.12 Let $e = \lambda x_1 \ldots x_n.\, a(e_1, \ldots, e_m)$ be a term in β-normal form of type $\alpha_1, \ldots, \alpha_n, \alpha_{n+1}, \ldots, \alpha_{n+k} \to \beta$, with $\beta \in \mathcal{T}_0$. The *η-expanded form* of e, denoted by $\eta[e]$, is produced by adding k new variables of the appropriate types to the binder and the matrix of the term, and (recursively) applying the same expansion to the subterms, to obtain

$$\lambda x_1 \ldots x_n x_{n+1} \ldots x_{n+k}.\, a(\eta[e_1], \ldots, \eta[e_m], \eta[x_{n+1}], \ldots, \eta[x_{n+k}]),$$

where $\tau(x_{n+i}) = \alpha_{n+i}$ for $1 \le i \le k$.

This is effectively the normal form of a term under the converse of the η-reduction rule (so that $\eta[e] \overset{*}{\longrightarrow}_{\eta} e$) and is *only* defined on a term already in β-normal form. It is easy to show that in an η-expanded form, every atom appears applied to as many arguments as allowed by its type, and that the matrices of all subterms are of base types. This form is more useful than the η-normal form because it makes the type of the term and all its subterms more explicit, and is therefore a convenient syntactic convention for representing the congruence class of all terms equal modulo the $\eta-$rule. It is easy to show, by structural induction on terms, that these expanded forms always exist and are unique (up to α-conversion), so that for any two terms s and t in β-normal form, we have $s \overset{*}{\longleftrightarrow}_{\eta} t$ iff $\eta[s] = \eta[t]$ (see [26], lemma 4.3). Thus, we have a Church-Rosser theorem in the following form.

Theorem 2.13 For every two terms s and t, we have $s \overset{*}{\longleftrightarrow}_{\beta\eta} t$ iff $\eta[s{\downarrow}] = \eta[t{\downarrow}]$.

Definition 2.14 Let $\mathcal{L}_{exp}$ be defined as the set of all η-expanded forms, i.e., $\mathcal{L}_{exp} = \{\eta[e{\downarrow}] \mid e \in \mathcal{L}\}$. Define the set $\mathcal{L}_{\eta}$ as the smallest subset of $\mathcal{L}$ containing $\mathcal{L}_{exp}$ and closed under application and lambda abstraction, i.e., $(e_1 e_2)$ and $\lambda x.\, e_1$ are in $\mathcal{L}_{\eta}$ whenever $e_1 \in \mathcal{L}_{\eta}$ and $e_2 \in \mathcal{L}_{\eta}$.

The essential features of $\mathcal{L}_{exp}$ and $\mathcal{L}_{\eta}$ which will allow us to restrict our attention to η-expanded forms are proved in the next lemma, which is from [26].

Lemma 2.15 For every variable x and every pair of terms e and e' of the appropriate types:

(1) $e, e' \in \mathcal{L}_{exp}$ implies that $(\lambda x.\, e) \in \mathcal{L}_{exp}$ and $(ee')\!\downarrow \in \mathcal{L}_{exp}$;
(2) $e \in \mathcal{L}_\eta$ implies that $e\!\downarrow \in \mathcal{L}_{exp}$;
(3) $e, e' \in \mathcal{L}_\eta$ implies that $(\lambda x.\, e) \in \mathcal{L}_\eta$ and $(ee') \in \mathcal{L}_\eta$;
(4) $e \in \mathcal{L}_\eta$ and $e \overset{*}{\longrightarrow}_\beta e'$ implies that $e' \in \mathcal{L}_\eta$;
(5) $e, e' \in \mathcal{L}_\eta$ implies that $e'[e/x] \in \mathcal{L}_\eta$.

These closure conditions for $\mathcal{L}_\eta$ (not all of which are satisfied by the set of η-normal forms) formally justify our leaving the η-rule implicit in the following sections by developing our method for higher-order unification in the language $\mathcal{L}_\eta$ and considering explicitly only β-conversion as a computation rule.[2] The reader interested in a more detailed treatment of these matters, including proofs of the previous results, is referred to [26] for details.

We now formalize the general notion of substitution of lambda terms for free variables in the $\beta\eta$-calculus, after which we show how this may be specialized to substitutions over $\mathcal{L}_{exp}$.

Definition 2.16 A *substitution* is any (total) function $\sigma : V \to \mathcal{L}$ such that $\sigma(x) \neq x$ for only finitely many $x \in V$ and for every $x \in V$ we have $\tau(\sigma(x)) = \tau(x)$. Given a substitution σ, the *support* (or *domain*) of σ is the set of variables $D(\sigma) = \{x \mid \sigma(x) \neq x\}$. A substitution whose support is empty is termed the *identity substitution*, and is denoted by Id. The set of variables *introduced by* σ is $I(\sigma) = \bigcup_{x \in D(\sigma)} FV(\sigma(x))$.

A subtle point of this definition is that substitutions are total functions which are non-trivial over only a finite number of variables; over the rest of V they simply map variables to themselves. Given a substitution σ, if its support is the set $\{x_1, \ldots, x_n\}$, and if $t_i = \sigma(x_i)$ for $1 \leq i \leq n$, then σ is also denoted by listing its bindings explicitly: $[t_1/x_1, \ldots, t_n/x_n]$. Given a term u, we may also denote $\sigma(u)$ as $u[t_1/x_1, \ldots, t_n/x_n]$.

Definition 2.17 A substitution ρ is a *renaming substitution away from* W if $\rho(x)$ is a variable (modulo η-conversion) for every $x \in D(\rho)$, $I(\rho) \cap W = \emptyset$, and for every x and y in $D(\rho)$, $\rho(x) \overset{*}{\longleftrightarrow}_\eta \rho(y)$ implies that $x = y$. If W is unimportant, then ρ is simply called a *renaming*. The *restriction* of a substitution σ to some W', denoted $\sigma|_{W'}$, is the substitution σ' such that

$$\sigma'(x) = \begin{cases} \sigma(x), & \text{if } x \in W'; \\ x, & \text{otherwise.} \end{cases}$$

[2] In fact, we shall depart from our convention in the interests of simplicity only when representing terms which are (up to η-conversion) variables, e.g., $\lambda xy.\, F(x,y)$. In some contexts, such as solved form systems, we wish to emphasize their character as variables, and will represent them as such, e.g., just F. In these cases, we shall be careful to say that 'F is (up to η-conversion) a variable,' etc.

Since $\mathcal{L}$ is freely generated, every substitution $\sigma : V \to \mathcal{L}$ has a unique extension $\widehat{\sigma} : \mathcal{L} \to \mathcal{L}$ defined recursively as follows.

Definition 2.18 Let σ_{-x} denote the substitution $\sigma|_{D(\sigma)-\{x\}}$. For any substitution σ,

$$\begin{aligned}
\widehat{\sigma}(x) &= \sigma(x) \text{ for } x \in V;\\
\widehat{\sigma}(a) &= a \text{ for } a \in \Sigma;\\
\widehat{\sigma}(\lambda x.\, e) &= \lambda x.\, \widehat{\sigma_{-x}}(e);\\
\widehat{\sigma}(\,(e_1\, e_2)\,) &= (\,\widehat{\sigma}(e_1)\,\widehat{\sigma}(e_2)\,).
\end{aligned}$$

Thus a substitution has an effect only on the *free* variables of a term. In the sequel, we shall identify σ and its extension $\widehat{\sigma}$. Note that by our assumption that the sets of bound variables and free variables in any context are disjoint, no variable capture will ever take place by application of a substitution. It is easy to show that the type of a term is unchanged by application of an arbitrary substitution.

Remark: It is important to note that by $\sigma(e)$ we denote the result of applying the substitution σ to e *without* β-reducing the result; we shall denote by $\sigma(e){\downarrow}$ the result of applying the substitution and then reducing the result to β-normal form. This rather non-standard separation we impose between substitution and the subsequent β-reduction is useful because we wish to examine closely the exact effect of substitution and β-reduction on lambda terms in a later section.

Definition 2.19 The *union* of two substitutions σ and θ, denoted by $\sigma \cup \theta$, is defined by

$$\sigma \cup \theta(x) = \begin{cases} \sigma(x), & \text{if } x \in D(\sigma)\,;\\ \theta(x), & \text{if } x \in D(\theta)\,;\\ x, & \text{otherwise,}\end{cases}$$

and is only defined if $D(\sigma) \cap D(\theta) = \emptyset$. The *composition* of σ and θ is the substitution denoted by $\sigma \circ \theta$ such that for every variable x we have $\sigma \circ \theta(x) = \widehat{\theta}(\sigma(x))$. Note carefully that we denote composition from *left to right*.

Definition 2.20 Given a set W of variables, we say that two substitutions σ and θ are *equal over* W, denoted $\sigma = \theta[W]$, iff $\forall x \in W,\ \sigma(x) = \theta(x)$. Two substitutions σ and θ are β*-equal over* W, denoted $\sigma =_\beta \theta[W]$ iff $\forall x \in W,\ \sigma(x) \overset{*}{\longleftrightarrow}_\beta \theta(x)$, or, equivalently, $\sigma(x){\downarrow} = \theta(x){\downarrow}$. The relations $=_\eta$ and $=_{\beta\eta}$ are defined in the same way but using $\overset{*}{\longleftrightarrow}_\eta$ and $\overset{*}{\longleftrightarrow}_{\beta\eta}$. We say that σ is *more general than* θ *over* W, denoted by $\sigma \leq \theta[W]$, iff there exists a substitution η such that $\theta = \sigma \circ \eta[W]$, and we have $\sigma \leq_\beta \theta[W]$ iff there exists some η' such that $\theta =_\beta \sigma \circ \eta'[W]$, and $\leq_\eta$ and $\leq_{\beta\eta}$ are defined analogously. When W is the set of all variables, we drop the notation $[W]$. If neither $\sigma \leq_{\beta\eta} \theta$ nor $\theta \leq_{\beta\eta} \sigma$ then σ and θ are said to be *independent*.

The comparison of substitutions modulo β-, η-, and $\beta\eta$-conversion is formally justified by the following lemma, which is easily proved by structural induction on terms:

Lemma 2.21 If σ and θ are arbitrary substitutions such that either $\sigma =_\beta \theta$, $\sigma =_\eta \theta$, or $\sigma =_{\beta\eta} \theta$, then for any term u we have either $\sigma(u) \overset{*}{\longleftrightarrow}_\beta \theta(u)$, $\sigma(u) \overset{*}{\longleftrightarrow}_\eta \theta(u)$, or $\sigma(u) \overset{*}{\longleftrightarrow}_{\beta\eta} \theta(u)$, respectively.

We now show that we can develop the notion of substitution wholly within the context of the language $\mathcal{L}_\eta$ developed above without loss of generality.

Definition 2.22 A substitution θ is said to be *normalized* if $\theta(x) \in \mathcal{L}_{exp}$ for every variable $x \in D(\theta)$.

We can assume without loss of generality that no normalized substitution has a binding of the form $\eta[x]/x$ for some variable x. A normalized renaming substitution has the form $[\eta[y_1]/x_1, \ldots, \eta[y_n]/x_n]$; the effect of applying such a substitution and then β-reducing is to rename the variables $x_1, \ldots, x_n$ to $y_1, \ldots, y_n$. The justification for using normalized substitutions is given by the following corollary of Lemma 2.15.

Corollary 2.23 If θ is a normalized substitution and $e \in \mathcal{L}_{exp}$, then $\theta(e) \in \mathcal{L}_\eta$ and $\theta(e)\!\downarrow \in \mathcal{L}_{exp}$.

It is easy to show that if σ and θ are normalized, then $\sigma =_{\beta\eta} \theta$ iff $\sigma = \theta$ and if θ' is the result of normalizing θ, then $\theta' =_{\beta\eta} \theta$.

Convention: In general, substitutions are assumed to be normalized in the rest of this paper, allowing us to factor out η-equivalence in comparing substitutions, so that we may, e.g., use $\leq_\beta$ instead of $\leq_{\beta\eta}$. In fact, the composition of two normalized substitutions could be considered to be a normalized substitution as well, so that $\sigma \leq_\beta \theta$ iff $\sigma \leq \theta$, but this need *not* be assumed in what follows. For example, the composition $[\lambda x.\, G(a)/F] \circ [\lambda y.\, y/G]$ is defined as $[\lambda x.\, ((\lambda y.\, y)a)/F, \lambda y.\, y/G]$, *not* as $[\lambda x.\, a/F, \lambda y.\, y/G]$. We shall continue to use $=_\beta$ and $\leq_\beta$ to compare normalized substitutions, although strictly speaking the subscript could be omitted if no composition is involved.

Definition 2.24 A substitution σ is *idempotent* if $\sigma \circ \sigma =_{\beta\eta} \sigma$.

A sufficient condition for idempotency is given by[3]

[3] In the first-order case, this condition is necessary as well, but in our more general situation we have counter-examples such as $\sigma = [\lambda x.\, F(a)/F]$.

Lemma 2.25 A substitution σ is idempotent whenever $I(\sigma) \cap D(\sigma) = \emptyset$.

That in most contexts we may restrict our attention to idempotent substitutions without loss of generality is demonstrated by our next result, which shows that any substitution is equivalent (over an arbitrarily chosen set of variables) up to renaming to an idempotent substitution. (For a proof see [43].)

Lemma 2.26 For any substitution σ and set of variables W containing $D(\sigma)$, there exists an idempotent substitution σ' such that $D(\sigma) = D(\sigma')$, $\sigma \leq_{\beta\eta} \sigma'$, and $\sigma' \leq_{\beta\eta} \sigma[W]$.

In general the assumption of idempotency simplifies matters. We shall provide specific motivations for the use of idempotent unifiers in the appropriate sections.

The net effect of these definitions, conventions, and results is that we can develop our method for unification of terms in the $\beta\eta$-calculus wholly within $\mathcal{L}_\eta$, leaving η-equivalence implicit in the form of the terms under consideration.

Before we proceed with the transformation method for the first-order case, we present the notion of a *multiset*.

Definition 2.27 Given a set A, a *multiset* over A is an unordered collection of elements of A which may have multiple occurrences of identical elements. More formally, a multiset over A is a function $M : A \to \mathbf{N}$ (where $\mathbf{N}$ is the set of natural numbers) such that an element a in A has exactly n occurrences in M iff $M(a) = n$. In particular, a does not belong to M when $M(a) = 0$, and we say that $a \in M$ iff $M(a) > 0$. The *union* of two multisets M_1 and M_2, denoted by $M_1 \cup M_2$, is defined as the multiset M such that for all $a \in A$, $M(a) = M_1(a) + M_2(a)$.

To avoid confusion between multisets and sets, we shall always state carefully when an object is considered to be a multiset. Note that multiset union is a distinct notion from the union of sets, since for example, if A is a non-empty multiset, then $A \cup A \neq A$.

3 Unification by Transformations on Systems

We now define unification of first-order terms and present an abstract view of the unification process as a set of non-deterministic rules for transforming a unification problem into an explicit representation of its solution, if such exists; in the next section this will be extended to the higher-order case. This elegant approach is due to [30], but was implicit in Herbrand's thesis [22].[4] Note that all terms in this section are purely first-order, so that there are no

[4] It is remarkable that in his thesis, Herbrand gave all the steps of a (nondeterministic) unification

lambda-abstractions, no variables at the head of terms, and for any term t, $FV(t)$ represents the set of *all* variables in t. Every first-order term is trivially in $\mathcal{L}_{exp}$.

Our representation for unification problems is the following.

Definition 3.1 A *term pair* or just a *pair* is a *multiset* of two terms, denoted, e.g., by $\langle s, t\rangle$, and a substitution θ is called a *standard unifier* (or just a *unifier*) of a pair $\langle s, t\rangle$ if $\theta(s) = \theta(t)$. A *term system* (or *system*) is a *multiset* of such pairs, and a substitution θ is a unifier of a system if it unifies each pair. The set of unifiers of a system S is denoted $U(S)$, and if S consists of only a single pair $\langle s, t\rangle$, the set of unifiers is denoted by $U(s, t)$.

Definition 3.2 A substitution σ is a *most general unifier*, or *mgu*, of a system S iff

(i) $D(\sigma) \subseteq FV(S)$;

(ii) $\sigma \in U(S)$;

(iii) For every $\theta \in U(S)$, $\sigma \leq \theta$.

It is well known that *mgu*'s always exist for unifiable systems, and it can be shown that *mgu*'s are unique up to composition with a renaming substitution, and so we shall follow the common practice of glossing over this distinction by referring to *the mgu* of a system, denoted by $mgu(S)$.

Definition 3.3 A pair $\langle x, t\rangle$ is in *solved form* in a system S and x in this pair is called a *solved variable* if x is a variable which does not occur anywhere else in S; in particular, $x \notin FV(t)$. A system is in solved form if all its pairs are in solved form; a variable is *unsolved* if it occurs in S but is not solved.

Note that a solved form system is always a *set* of solved pairs. The importance of solved form systems is shown by

Lemma 3.4 Let $S = \{\langle x_1, t_1\rangle, \dots, \langle x_n, t_n\rangle\}$ be a system in solved form. If $\sigma = [t_1/x_1, \dots, t_n/x_n]$, then σ is an idempotent *mgu* of S. Furthermore, for any substitution $\theta \in U(S)$, we have $\theta = \sigma \circ \theta$.

Proof. We simply observe that for any such θ, $\theta(x_i) = \theta(t_i) = \theta(\sigma(x_i))$ for $1 \leq i \leq n$, and $\theta(x) = \theta(\sigma(x))$ otherwise. Clearly σ is an *mgu*, and since $D(\sigma) \cap I(\sigma) = \emptyset$ by the definition of solved forms, it is idempotent. $\square$

Strictly speaking the substitution σ here is ambiguous in the case that there is at least one pair in S consisting of two solved variables; but since *mgu*'s are considered unique

algorithm based on transformations on systems of equations. These transformations are given at the end of the section on property A, page 148 of Herbrand [22].

up to renaming, and such pairs can be arbitrarily renamed, we denote this substitution by σ_S. As a special case, note that $\sigma_\emptyset = Id$.

We may analyse the process of finding *mgu*'s as follows. If $\theta(u) = \theta(v)$, then either (i) $u = v$ and no unification is necessary; or (ii) $u = f(u_1, \ldots, u_n)$ and $v = f(v_1, \ldots, v_n)$ for some $f \in \Sigma$, and $\theta(u_i) = \theta(v_i)$ for $1 \leq i \leq n$; or (iii) u is a variable not in $FV(v)$ or vice versa. If u is a variable not in $FV(v)$, then $[v/u] \in U(u,v)$ and $[v/u] \leq \theta$. By extending this analysis to account for systems of pairs, we have a set of transformations for finding *mgu*'s.

Definition 3.5 (The set of transformation rules $\mathcal{ST}$) Let S denote any system (possibly empty), $f \in \Sigma$, and u and v be two terms. We have the following transformations.

$$\{\langle u, u \rangle\} \cup S \implies S \tag{1}$$

$$\{\langle f(u_1, \ldots, u_n), f(v_1, \ldots, v_n) \rangle\} \cup S \implies \{\langle u_1, v_1 \rangle, \ldots, \langle u_n, v_n \rangle\} \cup S \tag{2}$$

$$\{\langle x, v \rangle\} \cup S \implies \{\langle x, v \rangle\} \cup \sigma(S), \tag{3}$$

where $\langle x, v \rangle$ is not a solved pair in S such that $x \notin FV(v)$, and $\sigma = [v/x]$.

Recall that systems are multisets, so the unions here are multiset unions; the intent of the left-hand side of each of these rules is to isolate a single pair to be transformed. Transformation (2) is called *term decomposition* and (3) is called *variable elimination*. We shall say that $\theta \in Unify(S)$ iff there exists some sequence of transformations

$$S \implies \ldots \implies S',$$

where S' is in solved form and $\theta = \sigma_{S'}$. (If no transformation applies, but the system is not in solved form, the procedure given here fails.)

Clearly, by choosing $S = \{\langle u, v \rangle\}$, we can attempt to find a unifier for two terms u, and v, as the following example shows.[5]

Example 3.6

$$\begin{aligned}
&\langle f(x, g(a, y)),\ f(x, g(y, x)) \rangle \\
&\implies_2 \ \langle x, x \rangle,\ \langle g(a, y), g(y, x) \rangle \\
&\implies_1 \ \langle g(a, y), g(y, x) \rangle \\
&\implies_2 \ \langle a, y \rangle,\ \langle y, x \rangle \\
&\implies_3 \ \langle a, y \rangle,\ \langle a, x \rangle\,.
\end{aligned}$$

[5] In examples, we shall often drop set brackets around systems, e.g., $S = \langle x_1, t_1 \rangle, \ldots, \langle x_n, t_n \rangle$.

The sense in which these transformations preserve the logically invariant properties of a unification problem is shown by

Lemma 3.7 If $S \Longrightarrow S'$ using any transformation from $\mathcal{ST}$, then $U(S) = U(S')$.

Proof. The only difficulty is in transformation (3). Suppose $\{\langle x, v\rangle\} \cup S \Longrightarrow_3 \{\langle x, v\rangle\} \cup \sigma(S)$ with $\sigma = [v/x]$. For any substitution θ, if $\theta(x) = \theta(v)$, then $\theta = \sigma \circ \theta$, since $\sigma \circ \theta$ differs from θ only at x, but $\theta(x) = \theta(v) = \sigma \circ \theta(x)$. Thus,

$$\begin{aligned} & \theta \in U(\{\langle x, v\rangle\} \cup S) \\ \text{iff } & \theta(x) = \theta(v) \text{ and } \theta \in U(S) \\ \text{iff } & \theta(x) = \theta(v) \text{ and } \sigma \circ \theta \in U(S) \\ \text{iff } & \theta(x) = \theta(v) \text{ and } \theta \in U(\sigma(S)) \\ \text{iff } & \theta \in U(\{\langle x, v\rangle\} \cup \sigma(S)). \end{aligned}$$

□

The point here is that the most important feature of a unification problem—its set of solutions—is preserved under these transformations, and hence we are justified in our method of attempting to transform such problems into a trivial (solved) form in which the existence of an *mgu* is evident.

We may now show the soundness and completeness of these transformations following [30].

Theorem 3.8 (Soundness) If $S \overset{*}{\Longrightarrow} S'$ with S' in solved form, then $\sigma_{S'} \in U(S)$.

Proof. Using the previous lemma and a trivial induction on the length of transformation sequences, we see that $U(S) = U(S')$, and so clearly $\sigma_{S'} \in U(S)$. □

Theorem 3.9 (Completeness) Every sequence of transformations

$$S = S_0 \Longrightarrow S_1 \Longrightarrow S_2 \Longrightarrow \ldots S'$$

must eventually terminate. Furthermore, S is unifiable iff every irreducible system S' derivable from S is in solved form, and for every $\theta \in U(S)$, $\sigma_{S'} \leq \theta$.

Proof. We first show that every transformation sequence terminates. For any system S, let us define a complexity measure $\mu(S) = < m, n >$, where m is the number of unsolved variables in the system, and n is the sum of the sizes of all the terms in the system. Then the lexicographic ordering on $< m, n >$ is well-founded, and each transformation produces a new system with a measure strictly smaller under this ordering: (1) and (2) must decrease n and can not increase m, and (3) must decrease m.

Therefore the relation $\Longrightarrow$ is well-founded, and every transformation sequence must end in some system to which no transformation applies. Suppose a given sequence ends in a system S'. Now by Lemma 3.7, $\theta \in U(S)$ iff $\theta \in U(S')$. Thus, S is unifiable iff S' contains no pairs of the form $\langle f(t_1, \ldots, t_n), g(t'_1, \ldots, t'_m)\rangle$ or of the form $\langle x, t\rangle$ with $x \in FV(t)$. But since no transformation applies, all pairs in S' must be in solved form. Finally, since $\theta \in U(S')$, by Lemma 3.4 we must have $\sigma_{S'} \leq \theta$. $\square$

Putting these two theorems together, we have that the set $\mathcal{ST}$ can always find an *mgu* for a unifiable system of terms; as remarked in [30], this abstract formulation can be used to model many different unification algorithms, by simply specifying data structures and a control strategy.

In fact, we have proved something stronger than necessary in Theorem 3.9: it has been shown that all transformation sequences terminate and that *any* sequence of transformations issuing from a unifiable system must eventually result in a solved form. This is possible because the problem is decidable. Strictly speaking, it would have been sufficient for completeness to show that if S is unifiable then there exists *some* sequence of transformations which results in a solved form, since then a complete search strategy, such as breadth-first search, could find the solved form. This form of completeness, which might be termed *non-deterministic completeness*, will be used in finding results on higher-order unification, where the general problem is undecidable.

In some contexts it may be useful to deal with idempotent unifiers which are renamed away from some set of 'protected' variables but which are most general over the set of variables in the original system. The next definition makes this precise. (In the next section we shall offer a variation of this notion for higher-order unification.)

Definition 3.10 Given a system S and a finite set W of 'protected' variables, a substitution σ is a *most general unifier of S away from W* (abbreviated $mgu(S)[W]$) iff

(i) $D(\sigma) \subseteq FV(S)$ and $I(\sigma) \cap (W \cup D(\sigma)) = \emptyset$;

(ii) $\sigma \in U(S)$;

(iii) For every $\theta \in U(S)$, $\sigma \leq \theta[FV(S)]$.

That such substitutions may always be found for unifiable systems is shown by the following lemma, whose proof may be found in [43].

Lemma 3.11 If S is a unifiable system and W a protected set of variables, then there exists a substitution σ which is a $mgu(S)[W]$.

4 Higher Order Unification via Transformations

In this section we extend the methods of the previous section to a more general context. Higher-order unification is more complex than first-order unification due to the presence of variables of functional type, the notion of scope and bound variables, and the fact that unification is defined in terms of $\beta\eta$-equivalence. This additional syntactic complexity has several serious consequences. First of all, the unification of terms of second-order and higher is undecidable in general [16]. Next, most general unifiers do not exist any more, and a more complex notion, that of a *complete set of unifiers*, is necessary. Finally, due to the complexity of the subproblem of unifying two flexible terms, the search space for a complete unification procedure may be infinitely branching, which forbids any reasonable implementation. Our analysis of the problem proceeds by examining the exact fashion in which substitution and β-reduction makes two terms identical from the top-down (i.e., from the head to the innermost subterms). We develop from this a set of non-deterministic transformations extending those of the previous section, and prove their non-deterministic completeness in an analogous fashion. In the next section, this is restricted to the problem of preunification.

Definition 4.1 The notion of pairs and systems of terms carries over from the first-order case. A substitution θ is a unifier of two lambda terms e_1 and e_2 iff $\theta(e_1) \overset{*}{\longleftrightarrow}_{\beta\eta} \theta(e_2)$.[6] A substitution is a unifier of a system S if it unifies each pair in S. The set of all unifiers of S is denoted $U(S)$ and if S consists of a single pair $\langle s, t\rangle$ then it is denoted $U(s,t)$.

This definition is more general than we shall need, in fact, since we shall develop our approach in $\mathcal{L}_\eta$ in order to factor out η-conversion, as was formally justified in Section §2. Thus for two terms $s, t \in \mathcal{L}_\eta$, we say that a normalized substitution θ is in $U(s,t)$ iff $\theta(s) \overset{*}{\longleftrightarrow}_{\beta} \theta(t)$, or, alternately, if $\theta(s){\downarrow} = \theta(t){\downarrow}$.

A pair of terms is solved in a system S if it is in the form $\langle \eta[x], t\rangle$, for some variable x which occurs only once in S; a system is solved if each of its pairs is solved. Our only departure from the use of η-expanded form is that we shall represent pairs of the form $\langle \eta[x], t\rangle$ as $\langle x, t\rangle$ in order to emphasize their correspondence to bindings t/x in substitutions, as in the first-order case of the previous section.

Example 4.2 If $u = f(a, g(\lambda x.\, G(\lambda y.\, x(b))))$ and $v = F(\lambda x.\, x(z))$, then $\theta = [\lambda x_2.\, f(a, g(x_2))/F, \lambda x_3.\, x_3(z_2)/G, b/z]$ is in $U(u,v)$, since $\theta(u){\downarrow} = \theta(v){\downarrow}$:

$$\theta(u) = f(a, g(\lambda x.\, [(\lambda x_3.\, x_3(z_2))(\lambda y.\, x(b))]))$$

[6] This is in the context of the $\beta\eta$-calculus; in the β-calculus the condition would be $\theta(e_1) \overset{*}{\longleftrightarrow}_{\beta} \theta(e_2)$.

$$\begin{aligned} &\longrightarrow_\beta\ f(a, g(\lambda x.\, [(\lambda y.\, x(b))z_2])) \\ &\longrightarrow_\beta\ f(a, g(\lambda x.\, x(b))) \\ &\longleftarrow_\beta\ (\lambda x_2.\, f(a, g(x_2)))(\lambda x.\, x(b))\ = \theta(v). \end{aligned}$$

The basic decidability results concerning higher-order unification are as follows.

Definition 4.3 For a given set of function constants Σ, the *unification problem* for the language $\mathcal{L}$ generated by Σ is to decide, for any arbitrary terms $e, e' \in \mathcal{L}$, whether the set $U(e, e')$ is non-empty. The n^{th}*-order unification problem* is to decide the unification problem for an arbitrary language of order n.

For example, in Section §3 we showed that the first-order unification problem is decidable. Unfortunately, this does not hold for higher-orders.

Theorem 4.4 The second-order unification problem is undecidable.

This result was shown by Goldfarb [16] using a reduction from Hilbert's Tenth Problem; previously, Huet [28] showed the undecidability of the third-order unification problem, using a reduction from the Post Correspondence Problem. These results show that there are second-order (and therefore arbitrarily higher-order) languages where unification is undecidable; but in fact there exist *particular* languages of arbitrarily high-order which have a decidable unification problem. Interestingly, Goldfarb's proof requires that the language to which the reduction is made contain at least one 2-place function constant. It has been shown in [11] that the unification problem for second-order monadic languages (i.e., no function constant has more than one argument place) is decidable, which has applications in certain decision problems concerning the lengths of proofs. A different approach to decidability is taken in [46], where decidable cases of the unification problem are found by showing that the search tree for some problems, although infinite, is regular, and that the set of unifiers can be represented by a regular expression. More generally, it has been shown by Statman [44] that the set of all decidable unification problems is polynomial-time decidable.

Besides the undecidability of higher-order unification, another problem is that *mgu*'s may no longer exist, a result first shown by [17]. For example, the two terms $F(a)$ and a have the unifiers $[\lambda x.\, a/F]$ and $[\lambda x.\, x/F]$, but there is no unifier more general than both of these. This leads us to extend the notion of a $mgu(S)[W]$ to the higher-order case by considering *complete sets* of unifiers. Our definition is a generalization of the one found in [26] to term systems.[7]

[7] We also generalize slightly the Huet definition by allowing the protected set of variables to be arbitrary.

Definition 4.5 Given a system S and a finite set W of 'protected' variables, a set U of normalized substitutions is a *complete set of unifiers for S away from W* (which we shall abbreviate by $CSU(S)[W]$) iff

(i) For all $\sigma \in U$, $D(\sigma) \subseteq FV(S)$ and $I(\sigma) \cap (W \cup D(\sigma)) = \emptyset$;

(ii) $U \subseteq U(S)$;

(iii) For every normalized $\theta \in U(S)$, there exists some $\sigma \in U$ such that $\sigma \leq_\beta \theta[FV(S)]$.

The first condition is called the *purity condition*, the second the *coherence condition*, and the last the *completeness condition*. If S consists of a single pair $\langle u, v\rangle$ then we use the abbreviation $CSU(u,v)[W]$. When W is not significant, we drop the notation $[W]$.

That there is no loss of generality in considering only normalized substitutions may be seen by the fact that any substitution is $\beta\eta$-equal to a normalized substitution. By providing a version of Lemma 3.11 for this new context, we see that condition (i) is without loss of generality as well.

Lemma 4.6 For any system S, substitution θ, and set of protected variables W, if $\theta \in U(S)$ then there exists some normalized substitution σ such that

(i) $D(\sigma) \subseteq FV(S)$ and $I(\sigma) \cap (W \cup D(\sigma)) = \emptyset$;

(ii) $\sigma \in U(S)$;

(iii) $\sigma \leq_{\beta\eta} \theta[FV(S)]$ and $\theta \leq_{\beta\eta} \sigma[FV(S)]$.

Proof. If $\sigma = \theta|_{FV(S)}$ satisfies condition (i), then we have our result trivially. Otherwise, if $I(\theta) = \{x_1, \ldots, x_n\}$ then let $\{y_1, \ldots, y_n\}$ be a set of new variables disjoint from the variables in W, $I(\theta)$, and $FV(S)$ such that $\tau(x_i) = \tau(y_i)$ for $1 \leq i \leq n$. Now define the renaming substitutions $\rho_1 = [\eta[y_1]/x_1, \ldots, \eta[y_n]/x_n]$ and $\rho_2 = [\eta[x_1]/y_1, \ldots, \eta[x_n]/y_n]$, let $\sigma' = \theta \circ \rho_1|_{FV(S)}$, and then let σ be the normalized version of σ'. Clearly σ satisfies (i), and since $\sigma =_{\beta\eta} \theta \circ \rho_1[FV(S)]$ we have the second part of (iii). Now, because $\rho_1 \circ \rho_2 =_{\beta\eta} Id[FV(S) \cup I(\theta)]$, we must have $\theta =_{\beta\eta} \theta \circ \rho_1 \circ \rho_2[FV(S) \cup I(\theta)]$. But then by the fact that $\sigma =_{\beta\eta} \theta \circ \rho_1[FV(S)]$ we have $\theta =_{\beta\eta} \sigma \circ \rho_2[FV(S)]$, and so $\sigma \leq_{\beta\eta} \theta[FV(S)]$, proving the first part of (iii). To show (ii), observe that for any $\langle u, v\rangle \in S$ we have $\theta(u)\!\downarrow = \theta(v)\!\downarrow$, and for any term t, we have $\sigma'(t) \overset{*}{\longleftrightarrow}_{\beta\eta} \sigma(t)$, and so

$$\sigma(u) \overset{*}{\longleftrightarrow}_{\beta\eta} \sigma'(u) = \rho_1(\theta(u)) \overset{*}{\longrightarrow}_{\beta\eta} \rho_1(\theta(u)\!\downarrow) = \rho_1(\theta(v)\!\downarrow) \overset{*}{\longleftarrow}_{\beta\eta} \rho_1(\theta(v)) = \sigma'(v) \overset{*}{\longleftrightarrow}_{\beta\eta} \sigma(v),$$

The original definition imposed the restriction that $W \cap FV(S) = \emptyset$ in order that variable renaming not be necessary. We relax this restriction so that we have a true generalization of a $mgu(S)[W]$ to higher-order unifiers, and allow renaming to be imposed or not, by setting W appropriately. Note that our definition is based on our use of $\mathcal{L}_\eta$; in the version for the $\beta\eta$-calculus, condition (iii) would use $\leq_{\beta\eta}$, and substitutions would not have to be normalized. The original Huet definition of a complete set may also be found in [9] in the context of E-unification.

which shows that $\sigma \in U(S)$. $\square$

This shows us that for any S and W, the set of all normalized unifiers satisfying condition (i) and (ii) of Definition 4.5 is a $CSU(S)[W]$, and so in particular there is no loss of generality in considering only normalized, idempotent unifiers θ such that $D(\theta) \cap I(\theta) = \emptyset$ in what follows. This will simplify our presentation.

Finally, we examine the relevance of solved form systems in $\mathcal{L}_\eta$.

Lemma 4.7 If $S = \{\langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle\}$ is a system in solved form, then $\{\sigma_S\}$ is a $CSU(S)[W]$ for any W such that $W \cap FV(S) = \emptyset$.

Proof. The first two conditions in Definition 4.5 are satisfied, since σ_S is an idempotent *mgu* of S, $W \cap FV(S) = \emptyset$, and $I(\sigma_S) \subseteq FV(S)$. Now, if $\theta \in U(S)$, then $\theta =_\beta \sigma_S \circ \theta$, since $\theta(x_i) \overset{*}{\longleftrightarrow}_\beta \theta(t_i) = \theta(\sigma_S(x_i))$ for $1 \le i \le n$, and $\theta(x) = \theta(\sigma_S(x))$ otherwise. Thus $\sigma_S \le_\beta \theta$ and so obviously $\sigma_S \le_\beta \theta[FV(S)]$. $\square$

4.1 Transformations for Higher Order Unification

We may analyze the process of higher-order unification as follows. Let us assume, without loss of generality, that u and v are two lambda terms in $\mathcal{L}_{exp}$ and that θ is an idempotent, normalized unifier of u and v. Thus there exists some sequence of reductions to a β-normal form: $\theta(u) \overset{*}{\longrightarrow}_\beta w \overset{*}{\longleftarrow}_\beta \theta(v)$. (Note that if all the terms instantiated by the substitution are first-order, then this sequence is trivial, since there are no β-reductions.) We may analyse this sequence *top-down*, examining the way in which each binding in the substitution (with its subsequent β-reduction, if the binding is higher-order) makes the two terms identical at each level of the terms. We have the following five cases (which are not intended to be mutually exclusive).

(A) $u = v$ and no unification is necessary. (Assume $u \neq v$ in the remaining cases.)

(B) No substitution takes place at the head in either term. In this case, $Head(u) = Head(v)$ and, since $u \neq v$, we must have $|u|, |v| > 0$. Thus, suppose $u = \lambda\overline{x_k}.\, a(\overline{u_n})$, $w = \lambda\overline{x_k}.\, a(\overline{w_n})$, and $v = \lambda\overline{x_k}.\, a(\overline{v_n})$, where $n > 0$ and either $a \in \Sigma$, or $a = x_i$ for some i, $1 \le i \le k$, or a is a free variable not in $D(\theta)$. In this case we must have $\theta(\lambda\overline{x_k}.\, u_i) \overset{*}{\longrightarrow}_\beta \lambda\overline{x_k}.\, w_i \overset{*}{\longleftarrow}_\beta \theta(\lambda\overline{x_k}.\, v_i)$ for $1 \le i \le n$, that is, the subterms of u and v are pair-wise unifiable by θ.

(C) Our two terms are $u = \lambda\overline{x_k}.\, F(\overline{x_k})$ and $v = \lambda\overline{x_k}.\, v'$, for some variable F and some term v', and where $F \notin FV(v)$. In this case, we must have

$$\theta(\lambda\overline{x_k}.\, F(\overline{x_k})) \overset{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.\, v'),$$

where $F \notin FV(v)$, and, if $\theta = [\lambda\overline{y_k}.\, t/F] \cup \theta'$, then since $\theta(F) \longleftarrow_\beta \theta(\lambda\overline{x_k}.\, F(\overline{x_k}))$,[8] we have $\theta(F) \overset{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.\, v')$, where F does not occur in v', so that we may use the same argument we used in the first-order case. If we let $\sigma = [\lambda\overline{x_k}.\, v'/F]$ then $\theta =_\beta \sigma \circ \theta$, since θ and $\sigma \circ \theta$ differ only at F, but

$$\theta(F) \overset{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.\, v') = \sigma \circ \theta(F).$$

This in fact shows that a pair of terms in this form has a single *mgu*. (For example, $\lambda x.\, F(x)$ and $\lambda x.\, f(x, z)$ are unified by $\theta = [\lambda y.\, f(y, a)/F, a/z]$, but $\sigma = [\lambda y.\, f(y, z)/F]$ is an *mgu*.) It should be obvious that this is a generalization of variable elimination to higher-order, since u is (up to η-equivalence) simply a variable not occurring in $FV(v)$.

(D) Some substitution takes place at the head of only one term; assume that this term is u (so that $Head(w) = Head(v)$). Then let $u = \lambda\overline{x_k}.\, F(\overline{u_n})$ and $v = \lambda\overline{x_k}.\, a(\overline{v_m})$ for some atom $a \neq F$ which is either a function constant, a bound variable, or a free variable not in $D(\theta)$. Now in order for the two terms to unify, we must make the head of u become a at some point in the sequence of β-reductions from $\theta(u)$ to w. There are two possibilities: either we imitate the head of v by substituting a term for F whose head is a, or we substitute a term for F which projects up a subterm of u. (The latter case is only possible if F is of higher-order type.) We consider each of these in turn.

(Imitation) The substitution for F matches the head symbol of v by imitating the head symbol a, where $a \in \Sigma$ or a is a free variable not in $D(\theta)$, as we saw in Example 4.2.[9] Thus we have $\theta(F) = \lambda\overline{z_n}.\, a(\overline{r_m})$ for some terms $\overline{r_m}$ and we have a reduction sequence of the form

$$\theta(u) = \theta(\lambda\overline{x_k}.\, ((\lambda\overline{z_n}.\, a(\overline{r_m}))\overline{u_n})) \longrightarrow_\beta \theta(\lambda\overline{x_k}.\, a(\overline{r'_m})) \overset{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.\, a(\overline{v_m})),$$

where $r'_i = r_i[u_1/z_1, \ldots, u_n/z_n]$ for $1 \le i \le m$. (Notice that by the idempotency of θ, for illustration we can partially instantiate the term u with just the binding for the head F in this sequence.)

(Projection) The substitution for F attempts to match the head symbol a of v by projecting up a subterm of u. There are three ways to do this, depending upon the head symbol of the term projected up. First of all, perhaps a subterm of u has a head a which provides the match; for example, $F(\lambda x.\, f(x, a))$ and $f(b, a)$ will be unified by the substitution $[\lambda y.\, y(b)/F]$ in this fashion (note that we had to provide an argument b to the subterm

[8] Note that the β-reduction simply replaces the bound variables $y_1, \ldots, y_k$ with $x_1, \ldots, x_k$, a useless operation in view of our assumption of α-equivalence.

[9] Note that it is impossible to imitate a bound variable, since the rules of the calculus disallow free variable capture.

$\lambda x.\, f(x,a)$ for the projection to work). The second reason to project is that perhaps a subterm of u is flexible, allowing us to start all over again in attempting to match the head of this new term to v. For example $F(\lambda x.\, G(x,a))$ and b can be unified by the substitution $[\lambda y.\, y(b)/F, \lambda x_1 x_2.\, x_1/G]$, where the binding for F works in this way. The third motivation for projection is that perhaps the subterm is itself a projection, and after some sequence of reductions, we have a term which is either flexible (and so we continue), or whose head is a and the match succeeds. For example, $\theta = [\lambda y_1.\, y_1(\lambda y_2.\, y_2(a))/F]$ unifies the two terms $u = F(\lambda x_1.\, x_1(\lambda x_2.\, f(x_2)))$ and $v = f(a)$ in this manner:

$$\begin{aligned}
\theta(u) &= [\lambda y_1.\, y_1(\lambda y_2.\, y_2(a))]\, \lambda x_1.\, x_1(\lambda x_2.\, f(x_2)) \\
&\longrightarrow_\beta\ [\lambda x_1.\, x_1(\lambda x_2.\, f(x_2))]\, \lambda y_2.\, y_2(a) \\
&\longrightarrow_\beta\ (\lambda y_2.\, y_2(a))\, \lambda x_2.\, f(x_2) \\
&\longrightarrow_\beta\ (\lambda x_2.\, f(x_2))\, a \\
&\longrightarrow_\beta\ f(a) = \theta(f(a)).
\end{aligned}$$

In substituting a projection for the head of a flexible term $u = \lambda \overline{x_k}.\, F(\overline{u_n})$, we are restricted by the type of F to projecting up a subterm u_i which will preserve the type of u. In particular, since we can only substitute a term of the same type as F, and since unification is only defined between terms of the same type, if $\tau(u) = \tau(v) = \alpha_1, \ldots, \alpha_k \to \beta$, then $\tau(u_i)$ must be some type $\gamma_1, \ldots, \gamma_{m'} \to \beta$ in order that the result of the projection preserves the type of u. Thus the type of the matrix of u_i must be the same as the matrix of u, and the substitution must provide arguments for each of the variables in the lambda binder of u_i. Thus if $\theta(F) = \lambda \overline{z_n}.\, z_i(\overline{r_{m'}})$ for some i, $1 \le i \le n$, then u_i must be in the form $u_i = \lambda \overline{y_{m'}}.\, u'_i$ where the type of the matrix u'_i is the same as the type of the matrices of u and v. In this case, the head a of u can be a function constant, a free variable, or a bound variable (i.e., one of the x_i), and thus we have a reduction sequence of the form

$$\theta(u) = \theta(\lambda \overline{x_k}.\, [(\lambda \overline{z_n}.\, z_i(\overline{r_{m'}}))\overline{u_n}]) \longrightarrow_\beta \theta(\lambda \overline{x_k}.\, [(\lambda \overline{y_{m'}}.\, u'_i)\overline{r'_{m'}}])$$

$$\xrightarrow{*}_\beta \theta(\lambda \overline{x_k}.\, a'(\overline{t_p})) \xleftrightarrow{*}_\beta \theta(\lambda \overline{x_k}.\, a(\overline{v_m})),$$

where $r'_i = r_i[u_1/z_1, \ldots, u_n/z_n]$ for $1 \le i \le m'$, $\lambda \overline{x_k}.\, a'(\overline{t_p}) = (\lambda \overline{x_k}.\, [(\lambda \overline{y_{m'}}.\, u'_i)\overline{r'_{m'}}])\!\downarrow$, and either $a' = a$ or a' is a free variable in $D(\theta)$.

(E) Substitutions take place at the heads of both terms. Then let $u = \lambda \overline{x_k}.\, F(\overline{u_n})$ and $v = \lambda \overline{x_k}.\, G(\overline{v_m})$, where both F and G are in $D(\theta)$. Here we must eventually match the heads of the two terms, but we can do it in a large number of ways. In order to simplify our analysis, we attempt to reduce it to the previous case if we can. Let us (without loss of generality) focus on the binding made for the variable F. There are two subcases.

(i) θ substitutes a non-projection term for F, e.g., $\theta(F) = \lambda\overline{z_n}.\, a(\overline{s_p})$, where $a \neq G$ is not a bound variable (and by idempotency is not a variable in $D(\theta)$), and then (possibly) causes a β-reduction, after which we can analyse the result using case (D).

(ii) θ substitutes a projection term for F (which obeys the typing constraints discussed above), e.g., $\theta(F) = \lambda\overline{z_n}.\, z_j(\overline{t_q})$, and then, after we reduce to normal form, if the head symbol is either a function constant, a bound variable, or a variable not in $D(\theta)$, we may analyse the result using case (D); if the head is a variable in $D(\theta)$, then we (recursively) apply case (E) to these new terms.

By recursively applying this analysis to the subproblems generated we may account for every binding made by θ and every β-reduction in the original sequence. This forms the basis for the set of transformation rules below, which find unifiers by 'incrementally' building up bindings using partial bindings, as informally shown in the introduction. In case (D) above, this means that there will only be a finite number of choices for a partial binding, since there is only one possible imitation and only a finite number of possible projections. In case (E), unfortunately, this is not true. As shown in [26], the problem is that two flexible terms may not possess a finite CSU, and in fact there may be an infinite number of independent unifiers which contain flexible terms as bindings, so that even if we only attempt to find the top function symbol of the binding, there are potentially an infinite number of choices, since for each type there is always an infinite number of function variables. Thus, even if there is only a finite number of function constants in the language, it is not possible to reduce the non-determinism of this case in general to a finite number of choices of partial bindings, and so the search tree must be infinitely branching.[10]

Given a system S of terms from $\mathcal{L}_{exp}$ and some normalized $\theta \in U(S)$, a complete unification procedure must always be able to find some substitution σ such that $\sigma \in U(S)$ and $\sigma \leq_\beta \theta[FV(S)]$. Recall from the introduction that the basic idea of the transformation method is that, given some $\theta \in U(S)$, we attempt to find 'pieces' of θ by finding solved pairs $\langle x, t\rangle$ such that $\theta(x) \overset{*}{\longleftrightarrow}_\beta \theta(t)$; in this case, we know by an argument similar to that used in Lemma 4.7 that $\theta =_\beta [t/x] \circ \theta$, and by finding enough such pairs, we eventually have a $\sigma =_\beta [t_1/x_1] \circ \ldots \circ [t_n/x_n]$, where σ is a unifier of S more general than (or equivalent to) θ. In other words, we may successively approximate θ until we have built up just enough of the substitution to unify the system. We do this by 'solving' variables (as in case (C) above) or using approximations to individual bindings, as in Huet's method and in [15], which we call *partial bindings*.

Definition 4.8 A *partial binding of type* $\alpha_1, \ldots, \alpha_n \to \beta$ (where β is a base type) is a

[10] See Section §5, where we discuss Huet's solution to this problem.

term of the form

$$\lambda \overline{y_n}.\, a\big(\lambda \overline{z^1_{p_1}}.\, H_1(\overline{y_n}, \overline{z^1_{p_1}}),\, \dots\, , \lambda \overline{z^m_{p_m}}.\, H_m(\overline{y_n}, \overline{z^m_{p_m}})\big)$$

for some atom a, where

(1) $\tau(y_i) = \alpha_i$ for $1 \leq i \leq n$,

(2) $\tau(a) = \gamma_1, \dots, \gamma_m \to \beta$, where $\gamma_i = \varphi^i_1, \dots, \varphi^i_{p_i} \to \gamma'_i$ for $1 \leq i \leq m$,

(3) $\tau(z^i_j) = \varphi^i_j$ for $1 \leq i \leq m$ and $1 \leq j \leq p_i$;

(4) $\tau(H_i) = \alpha_1, \dots, \alpha_n, \varphi^i_1, \dots, \varphi^i_{p_i} \to \gamma'_i$ for $1 \leq i \leq m$,

where $\gamma'_1, \dots, \gamma'_m$ are of base types. The immediate subterms of a partial binding (i.e., the arguments to the atom a) will be called *general flexible terms*.

Note that these partial bindings are uniquely determined (up to renaming of the free variables) by their type and by their head symbol a.

Definition 4.9 For a partial binding as in the previous definition, if a is either a function constant or a free variable, then such a binding is called an *imitation binding for* a; if a is a bound variable y_i for some i, $1 \leq i \leq n$, then it is called an i^{th} *projection binding*. A *variant* of a partial binding t is a term $\rho(t){\downarrow}$, where ρ is a renaming of the set $H_1, \dots, H_m$ of free variables at the heads of the general flexible terms in t away from all variables in the context in which t will be used. For any variable F, a partial binding t is *appropriate to* F if $\tau(t) = \tau(F)$. An imitation binding is *appropriate to* $\lambda \overline{x_k}.\, F(\overline{u_n})$ iff it is appropriate to F.

In the case of an i^{th} projection binding t for some i, $1 \leq i \leq n$, appropriate to a term $\lambda \overline{x_k}.\, F(\overline{u_n})$ of type $\alpha_1, \dots, \alpha_k \to \beta$, the reader may check that $\tau(u_i) = \varphi_1, \dots, \varphi_q \to \beta$ for some types $\varphi_1, \dots, \varphi_q$, so that the result of substituting the binding and β-reducing will preserve the type of the term.

For notational brevity we shall extend our vector style notation to represent partial bindings in the form

$$\lambda \overline{y_n}.\, a(\overline{\lambda \overline{z_{p_m}}.\, H_m(\overline{y_n}, \overline{z_{p_m}})}).$$

Following our analysis of higher-order unification given above, we have the following set of transformations.

Definition 4.10 (The set of transformations $\mathcal{HT}$.) Let S be a system of lambda-terms (possibly empty). We have the following transformations.

$$\{\langle u, u\rangle\} \cup S \Longrightarrow S \tag{1}$$

$$\{\langle \lambda\overline{x_k}.\, a(\overline{u_n}), \lambda\overline{x_k}.\, a(\overline{v_n})\rangle\} \cup S \Longrightarrow \bigcup_{1\leq i\leq n} \{\langle \lambda\overline{x_k}.\, u_i,\, \lambda\overline{x_k}.\, v_i\rangle\} \cup S, \tag{2}$$

where a is an arbitrary atom.

If $u = \lambda\overline{x_k}.\, F(\overline{x_k})$ and $v = \lambda\overline{x_k}.\, v'$, for some k, some variable F, and some term v', where $F \notin FV(v)$, then

$$\{\langle u, v\rangle\} \cup S \Longrightarrow \{\langle F, \lambda\overline{x_k}.\, v'\rangle\} \cup \sigma(S){\downarrow}, \tag{3}$$

where $\sigma = [\lambda\overline{x_k}.\, v'/F]$.

These three transformations are analogous to the set $\mathcal{ST}$. To provide for function variables, we need one more transformation, which is divided into three cases.

$$\{\langle \lambda\overline{x_k}.\, F(\overline{u_n}),\, \lambda\overline{x_k}.\, a(\overline{v_m})\rangle\} \cup S \Longrightarrow \{\langle F,\, t\rangle,\, \langle \lambda\overline{x_k}.\, F(\overline{u_n}), \lambda\overline{x_k}.\, a(\overline{v_m})\rangle\} \cup S, \tag{4a}$$

where a is either a function constant or a free variable not equal to F and t is a variant of an imitation binding for a appropriate to F, e.g., $t = \lambda\overline{y_n}.\, a(\overline{\lambda\overline{z_{p_m}}.\, H_m(\overline{y_n}, \overline{z_{p_m}})})$.

$$\{\langle \lambda\overline{x_k}.\, F(\overline{u_n}),\, \lambda\overline{x_k}.\, a(\overline{v_m})\rangle\} \cup S \Longrightarrow \{\langle F,\, t\rangle,\, \langle \lambda\overline{x_k}.\, F(\overline{u_n}), \lambda\overline{x_k}.\, a(\overline{v_m})\rangle\} \cup S, \tag{4b}$$

where a is some arbitrary atom (possibly bound) and t is a variant of an i^{th} projection binding for some i, $1 \leq i \leq n$, appropriate to the term $\lambda\overline{x_k}.\, F(\overline{u_n})$, that is, $t = \lambda\overline{y_n}.\, y_i(\overline{\lambda\overline{z_{p_q}}.\, H_q(\overline{y_n}, \overline{z_{p_q}})})$, such that if $Head(u_i)$ is a function constant, then $Head(u_i) = a$.

$$\{\langle \lambda\overline{x_k}.\, F(\overline{u_n}),\, \lambda\overline{x_k}.\, G(\overline{v_m})\rangle\} \cup S \Longrightarrow \{\langle F,\, t\rangle,\, \langle \lambda\overline{x_k}.\, F(\overline{u_n}), \lambda\overline{x_k}.\, G(\overline{v_m})\rangle\} \cup S, \tag{4c}$$

where $t = \lambda\overline{y_n}.\, a(\overline{\lambda\overline{z_{p_m}}.\, H_m(\overline{y_n}, \overline{z_{p_m}})})$ is a variant of some arbitrary partial binding appropriate to the term $\lambda\overline{x_k}.\, F(\overline{u_n})$ such that $a \neq F$ and $a \neq G$.

As a part of the transformations (4a)–(4c), we immediately apply transformation (3) to the new pair $\langle F, t\rangle$, which effectively amounts to just applying the substitution $[t/F]$ to the rest of the system. As in the set $\mathcal{ST}$, note that the unions above are *multiset unions*.

Henceforth we say that $\theta \in Unify(S)$ iff there exists a series of transformations $S \overset{*}{\Longrightarrow} S_n$, with S_n in solved form, and $\theta = \sigma_{S_n}|_{FV(S)}$.

Example 4.11 For example, the following series of transformations leads to a system in solved form.[11]

$$\begin{aligned}
&\langle F(f(a)), f(F(a))\rangle \\
&\Longrightarrow_{4a} \langle F, \lambda x.\, f(Y(x))\rangle,\ \langle \tfrac{(\lambda x.\, f(Y(x)))f(a)}{f(Y(f(a)))}, f(\tfrac{(\lambda x.\, f(Y(x)))a}{f(Y(a))})\rangle \\
&\Longrightarrow_{2} \langle F, \lambda x.\, f(Y(x))\rangle,\ \langle Y(f(a))), f(Y(a))\rangle \\
&\Longrightarrow_{4b} \langle F, \lambda x.\, f(\tfrac{(\lambda x.\, x)x}{x})\rangle,\ \langle Y, \lambda x.\, x\rangle,\ \langle \tfrac{(\lambda x.\, x)f(a)}{f(a)}, f(\tfrac{(\lambda x.\, x)a}{a})\rangle \\
&\Longrightarrow_{1} \langle F, \lambda x.\, f(x)\rangle,\ \langle Y, \lambda x.\, x\rangle
\end{aligned}$$

Hence, $[\lambda x.\, f(x)/F] \in Unify(F(f(a)), f(F(a)))$.

4.2 Soundness of the Transformations

The following lemmas will enable us to prove the soundness of this set of transformations.

Lemma 4.12 If $S \Longrightarrow S'$ using transformation (1) or (3), then $U(S) = U(S')$.

Proof. As in the first-order case, the only difficulty is in transformation (3). We must show that $U(\{\langle x, v\rangle\} \cup S) = U(\{\langle x, v\rangle\} \cup \sigma(S)\downarrow)$ where $\sigma = [v/x]$ and $x \notin FV(v)$. For any substitution θ, if $\theta(x) \overset{*}{\longleftrightarrow}_\beta \theta(v)$, then $\theta =_\beta \sigma \circ \theta$, since $\sigma \circ \theta$ differs from θ only at x, but $\theta(x) \overset{*}{\longleftrightarrow}_\beta \theta(v) = \sigma \circ \theta(x)$. But then, using Lemma 2.21, it is easy to see that $\theta \in U(S)$ iff $\sigma \circ \theta \in U(S)$. Furthermore, since for any term u we must have $\sigma \circ \theta(u) = \theta(\sigma(u)) \overset{*}{\longrightarrow}_\beta \theta(\sigma(u)\downarrow)$, it can easily be shown that $\sigma \circ \theta \in U(S)$ iff $\theta \in U(\sigma(S)\downarrow)$. Thus,

$$\begin{aligned}
&\theta \in U(\{\langle x, v\rangle\} \cup S) \\
\text{iff}\ &\theta(x) \overset{*}{\longleftrightarrow}_\beta \theta(v) \text{ and } \theta \in U(S) \\
\text{iff}\ &\theta(x) \overset{*}{\longleftrightarrow}_\beta \theta(v) \text{ and } \sigma \circ \theta \in U(S) \\
\text{iff}\ &\theta(x) \overset{*}{\longleftrightarrow}_\beta \theta(v) \text{ and } \theta \in U(\sigma(S)\downarrow) \\
\text{iff}\ &\theta \in U(\{\langle x, v\rangle\} \cup \sigma(S)\downarrow).
\end{aligned}$$

□

This lemma shows that the invariant properties of a problem are preserved under these two transformations, as they were in the first-order case.

[11] In order to show the effect of the β-reductions which follow the application of substitutions in (3), we often explicitly represent these reductions using an 'inference' style notation, e.g., we represent the effect of the substitution θ on the term e as $\frac{\theta(e)}{\theta(e)\downarrow}$, to illustrate both the effect of the substitution and the subsequent β-normal form.

Lemma 4.13 Let $S \Longrightarrow_2 S'$ where the pair in S transformed is $\langle \lambda\overline{x_n}.\, a(\overline{u_n}), \lambda\overline{x_n}.\, a(\overline{v_n})\rangle$. For any substitution θ,

(i) if a is either a constant or a bound variable or a free variable *not* in $D(\theta)$, then $\theta \in U(S)$ iff $\theta \in U(S')$;

(ii) if $a \in D(\theta)$ then $\theta \in U(S')$ implies that $\theta \in U(S)$.

Proof. If $\theta(\lambda\overline{x_k}.\, u_i) \overset{*}{\longleftrightarrow}_\beta \theta(\lambda\overline{x_k}.\, v_i)$ for $1 \leq i \leq n$, then clearly we must have

$$\theta(\lambda\overline{x_k}.\, a(\overline{u_n})) = \lambda\overline{x_k}.\, \theta(a)(\theta(u_1), \ldots, \theta(u_n)) \overset{*}{\longleftrightarrow}_\beta \lambda\overline{x_k}.\, \theta(a)(\theta(v_1), \ldots, \theta(v_n)) = \theta(\lambda\overline{x_k}.\, a(\overline{v_n})),$$

and so for any atom a we have $\theta \in U(S)$ whenever $\theta \in U(S')$. If a is either a function constant, a bound variable, or a variable not in $D(\theta)$, then $\theta(a) = a$ and it is easy to see that the reverse direction holds as well. □

Lemma 4.14 If $S \Longrightarrow S'$ using transformation (2) or (4), then $U(S') \subseteq U(S)$.

Proof. For (2) the result is a consequence of our previous lemma. Transformation (4) is in two parts, first adding a pair $\langle F, t\rangle$ to the system S, and then applying (3) to this new pair. Clearly, since $S \subseteq \{\langle F, t\rangle\} \cup S$ we must have $U(\{\langle F, t\rangle\} \cup S) \subseteq U(S)$. That the subsequent application of (3) to the new pair is sound has been shown by lemma 4.12. □

Since in transformation (4) we effectively commit ourselves to a particular approximation of a solution, it is hardly surprising that the inclusion $U(S') \subseteq U(S)$ is in general proper. Similarly, in the case of (2), decomposing flexible pairs may eliminate unifiers; for example $\langle F(a, b), F(c, d)\rangle$ has an infinite number of unifiers, but the system $\langle a, c\rangle, \langle b, d\rangle$ has none. These results show us that in higher-order unification, the set of solutions is invariant only under transformations (1), (3), and (2) in the case of two rigid terms.

Finally, using these lemmas we have

Theorem 4.15 (Soundness) If $S \overset{*}{\Longrightarrow} S'$, with S' in solved form, then the substitution $\sigma_{S'}|_{FV(S)} \in U(S)$.

Proof. By a simple induction on the length of transformation sequences, and using the previous lemmas in the induction step, we may show that $\sigma_{S'} \in U(S)$. But since the restriction has no effect as regards the effect of the substitution on the terms in S, we see that $\sigma_{S'}|_{FV(S)} \in U(S)$. □

4.3 Completeness of the Transformations

The completeness of our set of transformations will be proved along the lines of the proof of completeness of the set of transformations $\mathcal{ST}$ given earlier, except that now the transformation relation is not terminating in general, so we shall prove only the *non-deterministic* *completeness* of the set, i.e., we show that for any system S, if $\theta \in U(S)$, then there exists *some* sequence of transformations which finds a unifier σ such that $\sigma \leq_\beta \theta[FV(S)]$.

First we show the exact sense in which partial bindings can be considered to be approximations to bindings in substitutions.

Lemma 4.16 If $s = \lambda\overline{x_n}.\, a(\overline{s_m})$ is any term, then there exists a variant of a partial binding t and a substitution η such that $\eta(t) \overset{*}{\longrightarrow}_\beta s$.

Proof. If $m = 0$, i.e. $s = \lambda\overline{x_k}.\, a$, then the result is trivial by taking $t = s$ and $\eta = Id$. Otherwise, assume $m > 0$, and let $t = \lambda\overline{x_n}.\, a(\overline{\lambda\overline{z_{p_m}}.\, H_m(\overline{x_n}, \overline{z_{p_m}})})$ and $\eta = [\lambda\overline{x_n}.\, s_1/H_1, \ldots, \lambda\overline{x_n}.\, s_m/H_m]$. Then by the type of the head a, the i^{th} subterm s_i must be in the form $\lambda z_{p_i}.\, s'_i$, so that

$$\eta\big(\lambda\overline{z_{p_i}}.\, H_i(\overline{x_n}, \overline{z_{p_i}})\big) \longrightarrow_\beta s_i,$$

for each i, $1 \leq i \leq m$. Thus $\eta(t) \overset{*}{\longrightarrow}_\beta s$. □

Lemma 4.17 If $\theta = [s/F] \cup \theta'$ then there exists a variant of a partial binding t appropriate to F and a substitution η such that

$$\begin{aligned} \theta \;&=\; [s/F] \cup \eta \cup \theta'[D(\theta)] \\ &=_\beta\; [t/F] \circ \eta \cup \theta'[D(\theta)]. \end{aligned}$$

Furthermore, if $D(\theta) \cap I(\theta) = \emptyset$, then $\theta'' = [s/F] \cup \eta \cup \theta'$ is a unifier of the pair $\langle F, t\rangle$ and $D(\theta'') \cap I(\theta'') = \emptyset$.

Proof. Given the term s, let t and η be as in the previous lemma. Since t is a variant, $D(\eta) \cap D(\theta) = \emptyset$, and since furthermore $\eta(t) \overset{*}{\longrightarrow}_\beta s$, we have $[s/F] = [s/F] \cup \eta =_\beta [t/F] \circ \eta[D(\theta)]$, from which the first part follows. If $D(\theta) \cap I(\theta) = \emptyset$ (so that θ is idempotent), then since t is a variant, $D(\eta) \cap I(\theta) = \emptyset$, so that $D(\theta'') \cap I(\theta'') = \emptyset$ (and $\theta''(s) = s$) and finally, $\theta''(F) = s \overset{*}{\longleftarrow}_\beta \eta(t) = \theta''(t)$. □

Note that if $D(\theta) \cap I(\theta) \neq \emptyset$ in this lemma, then potentially θ has a binding for the head of s and t, and so possibly $\theta''(t) \neq \eta(t)$. Also, notice that $[s/F] \cup \eta$ and $[t/F] \circ \eta$ are only β-equal (over $D(\theta)$) because we do not assume that the implicit β-reductions are performed when substitutions are composed. These lemmas show the motivation for the term 'partial binding' and provide the formal justification for the assertion that partial bindings can be used to build up substitutions incrementally.

Next we define a set of transformations on pairs θ, S which shows how the structure of a substitution θ can determine an appropriate sequence of transformations.

Definition 4.18 (The set $\mathcal{CT}$) Let θ be a normalized substitution and S be an arbitrary system. The first three transformations are essentially from the set $\mathcal{HT}$:

$$\theta, S \implies_i \theta, S'$$

for $1 \leq i \leq 3$ iff $S \implies_i S'$ in the set $\mathcal{HT}$, with the restriction that (2) is only applied to a pair $\langle u, v\rangle$ if the top function symbol in u and v is *not* a free variable in $D(\theta)$. Also, we have

$$[s/F] \cup \theta, \{\langle \lambda\overline{x_k}.\, F(\overline{u_n}),\, \lambda\overline{x_k}.\, v\rangle\} \cup S \implies_4 [s/F] \cup \eta \cup \theta, \{\langle F,\, t\rangle,\, \langle \lambda\overline{x_k}.\, F(\overline{u_n}), \lambda\overline{x_k}.\, v\rangle\} \cup S,$$

where F is not solved in the system on the left side, s is some term $\lambda\overline{y_n}.\, a(\overline{s_m})$,

$$t = \lambda\overline{y_n}.\, a(\overline{\lambda\overline{z_{p_m}}.\, H_m(\overline{y_n}, \overline{z_{p_m}})})$$

is a partial binding appropriate to F with the same (up to α-conversion) head as s, and

$$\eta = [\lambda\overline{y_n}.\, s_1/H_1, \ldots, \lambda\overline{y_n}.\, s_m/H_m].$$

(Note that perhaps $m = 0$ in which case η is omitted.) Transformation (3) is immediately applied as a part of (4), as in the set $\mathcal{HT}$. Again, notice that $[s/F] \cup \eta \cup \theta =_\beta [t/F] \circ \eta \cup \theta$.

Example 4.19 Let $\theta = [\lambda x.\, f(x)/F]$ and $S = \{\langle F(f(a)), f(F(a))\rangle\}$. We have the following sequence of $\mathcal{CT}$-transformations.

$$\begin{aligned}
&[\lambda x.\, f(x)/F], \{\langle F(f(a)), f(F(a))\rangle\}\\
&\implies_4 [\lambda x.\, f(x)/F, \lambda x.\, x/Y], \{\langle F, \lambda x.\, f(Y(x))\rangle,\, \langle \tfrac{(\lambda x.\, f(Y(x)))f(a)}{f(Y(f(a)))}, f(\tfrac{(\lambda x.\, f(Y(x)))a}{f(Y(a))})\rangle\}\\
&\implies_2 [\lambda x.\, f(x)/F, \lambda x.\, x/Y], \{\langle F, \lambda x.\, f(Y(x))\rangle,\, \langle Y(f(a)), f(Y(a))\rangle\}\\
&\implies_4 [\lambda x.\, f(x)/F, \lambda x.\, x/Y], \{\langle F, \lambda x.\, f(\tfrac{(\lambda x.\, x)x}{x})\rangle,\, \langle Y, \lambda x.\, x\rangle,\, \langle \tfrac{(\lambda x.\, x)f(a)}{f(a)}, f(\tfrac{(\lambda x.\, x)a}{a})\rangle\}\\
&\implies_1 [\lambda x.\, f(x)/F, \lambda x.\, x/Y], \{\langle F, \lambda x.\, f(x)\rangle,\, \langle Y, \lambda x.\, x\rangle\}
\end{aligned}$$

The next lemma shows us how these transformations are useful for proving completeness.

Lemma 4.20 If $\theta \in U(S)$ for some system S not in solved form, and W is a set of variables, then there exists some transformation $\theta, S \implies \theta', S'$ such that

(i) $\theta = \theta'[W]$;
(ii) If $D(\theta) \cap I(\theta) = \emptyset$ then $\theta' \in U(S')$ and $D(\theta') \cap I(\theta') = \emptyset$; and
(iii) $S \implies S'$ with respect to the set $\mathcal{HT}$.

Proof. Since S is not in solved form, there must exist some pair $\langle u, v\rangle$ which is not solved in S. We have three cases: (A) If $u = v$ then we may apply (1) or (2); (B) if $Head(u) = Head(v) \notin D(\theta)$, then we can apply (2); otherwise, (C) we have $u \neq v$ and either $Head(u) \neq Head(v)$ or $Head(u) = Head(v) \in D(\theta)$. In case (C), either u or v has an unsolved variable from $D(\theta)$ at its head; without loss of generality, assume that u has. Thus, we have $u = \lambda\overline{x_k}.\, F(\overline{u_n})$ and $v = \lambda\overline{x_k}.\, v'$ with $F \in D(\theta)$ and F not solved in S and (4) must apply, and in the special case that $u \xrightarrow{*}_\eta F$ and $F \notin FV(v)$, we can alternately apply (3). Although there may not be a unique choice about which transformation to apply, at least one must apply, and thus we have some transformation $\theta, S \Longrightarrow_i \theta', S'$. In the case that $1 \leq i \leq 3$, (i) holds because $\theta' = \theta$, by our soundness lemmas of the previous section we have (ii), and (iii) holds by the definition of the set $\mathcal{CT}$. If $i = 4$ then by our previous corollary we have extended $\theta = [s/F]\cup\varphi$ to a substitution $\theta' = [s/F]\cup\eta\cup\varphi =_\beta [t/F]\circ\eta\cup\varphi$ where we can assume that $D(\eta) \cap W = \emptyset$ (showing (i)), and we have added a pair $\langle F, t\rangle$ to S to form S'. From the definition of $\mathcal{CT}$ and the previous lemma it is clear that we have $D(\theta') \cap I(\theta') = \emptyset$ and $\theta'(F) = s \xleftarrow{*}_\beta \eta(t) = \theta'(t)$, so that $\theta' \in U(S')$, showing (ii). Finally, since S is unifiable it is not hard to see that the conditions imposed on (4) in $\mathcal{CT}$ are consistent with (4) in $\mathcal{HT}$. If $Head(v)$ is not a variable in $D(\theta)$, then we have two cases: if $Head(s) = Head(v)$, then $S \Longrightarrow_{4a} S'$ (i.e., this is an imitation case); otherwise, s is a projection, and $S \Longrightarrow_{4b} S'$. If $Head(v) \in D(\theta)$ then $S \Longrightarrow_{4c} S'$. $\square$

Corollary 4.21 If $\theta \in U(S)$ and no transformation applies to θ, S then S is in solved form.

Finally, we may present our completeness proof.

Theorem 4.22 (Completeness of $\mathcal{HT}$) For any system S, if $\theta \in U(S)$ then there exists some sequence of transformations

$$S = S_0 \Longrightarrow S_1 \Longrightarrow S_2 \Longrightarrow \ldots \Longrightarrow S_n,$$

where S_n is in solved form and $\sigma_{S_n} \leq_\beta \theta[FV(S)]$.

Proof. By Lemma 2.26 we may assume without loss of generality that $D(\theta) \cap I(\theta) = \emptyset$ (since if not we may find a substitution $\theta'' = \theta[FV(S)]$ fulfilling these conditions). We prove this result using the set $\mathcal{CT}$, first showing that every sequence of $\mathcal{CT}$ transformations terminates. For any θ and S, define the complexity measure $\mu(\theta, S) = < M, n >$, where n is the sum of the sizes (i.e., the number of atomic subterms) of all terms in S, and M is the sum of the sizes of the bindings in θ for variables which are not solved in S:

$$M = \sum\{\, |\theta(x)| \mid x \in D(\theta) - Sol(S)\,\},$$

where $Sol(S)$ is the set of all variables solved in S. The standard lexicographic ordering on pairs of natural numbers is well-founded, and any $\mathcal{CT}$-transformation produces a pair strictly smaller under the ordering: (1) and (2) reduce n without affecting M, (3) reduces M by removing a variable from $D(\theta) - Sol(S)$, and (4) reduces M. In (4), for some variable F in $D(\theta) - Sol(S)$, the binding $[s/F]$ is deleted from θ where s is some term of the form $\lambda\overline{y_n}.\, a(\overline{s_m})$, and some new bindings $[\lambda\overline{y_n}.\, s_1/H_1, \ldots, \lambda\overline{y_n}.\, s_m/H_m]$ associated with new unsolved variables are added to θ to form θ'. However, the sum of the sizes of the new bindings in θ' is strictly smaller than the size of s (since s also contains a). Hence every sequence of $\mathcal{CT}$-transformations is finite.

Thus there must exist a sequence of transformations

$$\theta, S = \theta_0, S_0 \implies \theta_1, S_1 \implies \ldots \implies \theta_m, S_m$$

such that no transformation applies, and by induction on m using the previous lemma, with $FV(S)$ for the set W, we have $\theta = \theta_m[FV(S)]$, $\theta_m \in U(S_m)$, and there is a corresponding sequence of $\mathcal{HT}$-transformations

$$S = S_0 \implies S_1 \implies \ldots \implies S_m$$

and by the corollary we know that S_m is in solved form. Finally, by Lemma 4.7 we have $\sigma_{S_m} \leq_\beta \theta_m = \theta[FV(S)]$. □

The reader should note that this proof is essentially similar to that of Theorem 3.9. Finally, combining our soundness and completeness results, we have that this method is capable of non-deterministically finding a unifier of S more general than any given unifier. More formally, we may characterize the set of substitutions non-deterministically found by the set of transformations $\mathcal{HT}$ as follows.

Theorem 4.23 For any system S, the set

$$\{\sigma_{S'}|_{FV(S)} \mid S \overset{*}{\implies} S', \text{ and } S' \text{ is in solved form}\}$$

is a $CSU(S)$. By application of the appropriate renaming substitution away from W, this set is a $CSU(S)[W]$ for any W.

Proof. We must simply verify the conditions in Definition 4.5. Coherence was shown in Theorem 4.15 and our previous result demonstrated completeness. By restricting the idempotent substitution $\sigma_{S'}$ to $FV(S)$ we satisfy purity for W empty. If W is not empty, we may suitably rename the variables introduced by each of the substitutions $\sigma_{S'}$ away from W, using Lemma 4.6. □

The careful reader will note that we have made no assumptions about the order in which transformations are performed, and so these results apply in a very general way to the derivation of solved form systems from initial systems of terms. In particular, we see that the strategy of *eager variable elimination*, in which transformation (3) is performed as soon as possible on any pair to which it applies, is complete (in the case of general E-unification this problem is still open, see [15]). The search space is thereby reduced, since we do not need to build up such solved pairs one symbol at a time. In addition, it shows how this set of transformations is a true generalization of the transformations used for first-order unification.

5 Huet's Procedure Revisited

The set of transformations given in the previous section were proved to be complete for the problem of general higher-order unification, that is, they can non-deterministically find *any* higher-order unifier of two arbitrary terms. Unfortunately, as remarked above, the 'don't know' non-determinism of this set causes severe implementation problems in the case of two flexible terms (case (E) in our analysis), and, as discussed above, this 'guessing' of partial bindings in this case can not be avoided without sacrificing completeness, and so the search tree of all transformation sequences may be infinitely branching at certain nodes, causing a disastrous explosion in the size of the search space.

Huet's well-known solution to this problem [25, 26] was to redefine the problem in such a way that such flexible-flexible pairs are considered to be already solved; this partial solution of the general higher-order unification problem turns out to be sufficient for refutation methods (see [24]), and this is the method used in most current systems. We show here how to explain this approach in terms of transformations on systems. The only changes have to do with redefining the notion of a solved system and restricting the set of transformations.

Definition 5.1 A pair of terms $\langle x, e \rangle$ is in *presolved form* in a system S if it is in solved form in S (as above) *or* if it is a pair consisting of two flexible terms. A system is in presolved form if each member is in presolved form. For a set S in presolved form, define the associated substitution σ_S as the *mgu* $\sigma_{S'}$ of the set S' of solved pairs of S.

Definition 5.2 Let $\cong$ be the least congruence relation on $\mathcal{L}$ containing the set of pairs $\{(u, v) \mid u, v \text{ are both flexible terms } \}$. A substitution θ is a *preunifier* of u and v if $\theta(u)\!\downarrow\, \cong \theta(v)\!\downarrow$.

The importance of pre-unifiers is shown by our next definition and lemma.

Definition 5.3 For every $\phi = \alpha_1, \ldots, \alpha_n \to \beta \in \mathcal{T}$, with $n \geq 0$, define a term

$$\widehat{e}_\phi = \lambda x_1 \ldots x_n. v,$$

where $\tau(x_i) = \alpha_i$ for $1 \leq i \leq n$ and $v \in V_\beta$ is a new variable which will never be used in any other term. Let ζ be an (infinite) set of bindings

$$\zeta = \{\widehat{e}_{\tau(x)}/x \mid x \in V\}.$$

Finally, if S' is a pre-solved system containing a set S'' of flexible-flexible pairs, then define the substitution

$$\zeta_{S'} = \zeta|_{FV(S'')}.$$

As in [26], it is easy to show this next result.

Lemma 5.4 If S is a system in pre-solved form then the substitution $\sigma_S \cup \zeta_S$ is a unifier of S.

This lemma asserts that pre-unifiers may always be extended to true unifiers by finding trivial unifiers for the flexible-flexible terms in the pre-solved system.

The set of transformations for finding preunifiers is a slightly restricted version of the set of transformations $\mathcal{HT}$.

Definition 5.5 (The set of transformations $\mathcal{PT}$) Let S be a system, possibly empty. To the transformations (1) and (3) from $\mathcal{HT}$ we add three (restricted) transformations:

$$\{\langle \lambda \overline{x_k}.\, a(\overline{u_n}), \lambda \overline{x_k}.\, a(\overline{v_n})\rangle\} \cup S \Longrightarrow \bigcup_{1 \leq i \leq n} \{\langle \lambda \overline{x_k}.\, u_i,\, \lambda \overline{x_k}.\, v_i\rangle\} \cup S, \tag{$2'$}$$

where $a \in \Sigma$ or $a = x_j$ for some j, $1 \leq j \leq k$.

$$\{\langle \lambda \overline{x_k}.\, F(\overline{u_n}),\, \lambda \overline{x_k}.\, a(\overline{v_m})\rangle\} \cup S \implies \{\langle F,\, t\rangle,\, \langle \lambda \overline{x_k}.\, F(\overline{u_n}), \lambda \overline{x_k}.\, a(\overline{v_m})\rangle\} \cup S, \tag{$4'a$}$$

where $a \in \Sigma$ and t is a variant of an imitation binding for a appropriate to F.

$$\{\langle \lambda \overline{x_k}.\, F(\overline{u_n}),\, \lambda \overline{x_k}.\, a(\overline{v_m})\rangle\} \cup S \implies \{\langle F,\, t\rangle,\, \langle \lambda \overline{x_k}.\, F(\overline{u_n}), \lambda \overline{x_k}.\, a(\overline{v_m})\rangle\} \cup S, \tag{$4'b$}$$

where either $a \in \Sigma$ or $a = x_j$ for some j, $1 \leq j \leq k$, and t is a variant of an i^{th} projection binding for some i, $1 \leq i \leq n$, appropriate to the term $\lambda \overline{x_k}.\, F(\overline{u_n})$.

After each of ($4'$a) and ($4'b$), we apply transformation (3) to the new pair introduced. As in our previous definitions, recall that the unions are multiset unions.

We say that $\theta \in PreUnify(S)$ iff there exists a series of transformations from $\mathcal{PT}$

$$S = S_0 \implies S_1 \implies \ldots \implies S_n,$$

with S_n in pre-solved form, and $\theta = \sigma_{S_n}|_{FV(S)}$.

In terms of Huet's procedure (see the Appendix) the first two transformations represent approximately the effect of Simplify, and ($4'$a) and ($4'$b) represent the processes of imitation and projection respectively in Match. Transformation (3) represents the effect of applying substitutions in Simplify, but also allows variable elimination, which was remarked upon by Huet (see [26], p. 3-57) but not emphasized.[12] Note that the transformations (1), ($2'$), and (3) in $\mathcal{PT}$ preserve the set of solutions invariant, as discussed in Section §4.2.

We now present the major results concerning this formulation of higher-order unification, following [26]. Their proofs are simple modifications of our previous results, and are left to the reader.

Theorem 5.6 (Soundness) If $S \overset{*}{\implies} S'$, with S' in presolved form, then the substitution $\sigma_{S'}|_{FV(S)}$ is a preunifier of S.

Theorem 5.7 (Completeness) If θ is some preunifier of the system S, then there exists a sequence of transformations $S \overset{*}{\implies} S'$, with S' in presolved form, such that

$$\sigma_{S'}|_{FV(S)} \leq_\beta \theta.$$

The search tree for this method consists of all the possible sequences of systems created by transforming the original two terms. Leaves consist of pre-solved systems or systems where no transformation can be applied. These correspond to the **S** and **F** nodes in Huet's algorithm; in fact, the search trees generated are essentially the same as the *matching trees* defined in [25], except that here an explicit representation of the matching substitutions found so far is carried along in the system (see the Appendix). The set of pre-unifiers potentially found by our procedure is the set of pre-solved leaves in the search tree.

As in the case of general higher-order unification, the strategy of eager variable elimination is complete, allowing a reduction in the size of the search space, since we do not need to build up the terms using partial bindings. This rule had been suggested as a heuristic

[12] Jensen and Pietrzykowski [41] suggest a similar rule as a heuristic improvement.

in [26] and [41], but not emphasized as an essential part of the method of building up substitutions, as here. We note also as a minor point that in some cases it is possible to apply variable elimination to a presolved system so that that this binding is incorporated into the *mgu* of the final solved form system. For example, the following initial system is presolved, but in fact has a *mgu* $[\lambda x.\, G(a,x)/F]$:

$$\begin{aligned}&\langle \lambda x.\, F(x), \lambda x.\, G(a,x)\rangle, \langle F(b), G(a,b)\rangle\\ &\Longrightarrow_3 \langle F, \lambda x.\, G(a,x)\rangle, \langle \tfrac{(\lambda x.\, G(a,x))\, b}{G(a,b)}, G(a,b)\rangle\\ &\Longrightarrow_1 \langle F, \lambda x.\, G(a,x)\rangle.\end{aligned}$$

We give a pseudo-code version of Huet's method for the typed $\beta\eta$-calculus in an appendix as an example of the way in which these transformations can be used to design more practical procedures.

6 Conclusion

We have presented in this paper a reexamination of the problem of general higher-order unification, using the abstract approach of transformations on systems of terms. We feel that this kind of analysis provides the right level of abstraction by revealing the logical issues in their purest form. As shown in our application of this method to general E-unification [15], this abstract approach allows us to derive complete sets of abstract transformations for unification in various contexts from an analysis of what it means for two terms to be 'the same' (e.g., modulo a set of equations in E-unification and modulo β-reduction in higher-order unification). We claim that this approach is more perspicuous than those previously advanced, permits more direct soundness and completeness proofs, and unifies and justifies the various approaches taken to unification problems. This abstract characterization of the process of unification in various settings clarifies the basic similarities and differences of the problems by removing the notion of control and showing exactly where non-determinism arises and where it may be eliminated. The three sets of transformations $\mathcal{ST}$, $\mathcal{PT}$, and $\mathcal{HT}$ thus represent an (inclusion) hierarchy of abstract methods for unification. One result that came out of this is that variable elimination can be extended from first-order unification to both general higher-order unification and to pre-unification; in particular, the strategy of eager variable elimination is still complete. This work is part of a project [43] which attempts to provide a general theory of complete sets of transformations for unification, including higher-order unification and general E-unification; we hope to extend this approach to higher-order E-unification and unification of polymorphic lambda terms. It is our hope that the abstract method of transformations on systems will yield still further insights into the nature of unification problems in the future as well.

Acknowledgment: We would like to thank Dale Miller, Frank Pfenning, Rick Statman, Eric Tiden, and the anonymous referees for many helpful comments which improved the paper immeasurably.

Appendix

The basic idea of the seminal higher-order preunification procedure developed by Huet [26] is to search for preunifiers of two lambda-terms one substitution at a time by alternately decomposing terms and finding matching substitutions for the heads, stopping when the subterms are found to be either trivially unifiable, or not unifiable. More specifically, the procedure generates a tree (of OR branches) from a root consisting of the original pair of terms, whose nodes are disagreement sets of pairs of terms not yet unified, and whose arcs are labelled by substitutions found and applied to generate new descendants. The tree is explored and unifiers incrementally created by decomposing pairs of terms until their heads are no longer equal and then finding substitutions which match the heads of pairs, if possible. Identical pairs of terms are fully decomposed and eventually removed from the disagreement set. When either a trivially unifiable disagreement set, composed only of flexible-flexible pairs, is found (success) or an un-unifiable pair, i.e., a rigid-rigid pair with dissimilar top function symbols, is found (failure), a branch is terminated. In general this process may not terminate, since whether two lambda terms are unifiable is only semi-decidable.

We now present a pseudo-Pascal version of Huet's non-deterministic procedure for pre-unifying two terms in the $\beta\eta-$calculus.

```
global variable  T : searchTree;

procedure LambdaUnifiers( $e_1, e_2$ : $\lambda$-terms );
{ This procedure enumerates a complete set of pre-unifiers for
  two $\lambda$-terms of the same type. }
var
  $N$, $N'$ : treeNodes; $e'_1, e'_2$ : $\lambda$-terms; $\Sigma$ : substSet; $\sigma$, $\rho$, $\theta$ : unifier;
begin
  $T$ := the one node tree consisting of Simplify($\{(e_1, e_2)\}$);
  while exists an unmarked leaf node $N$ in T do
    begin
      Pick some flexible-rigid pair $(e_1, e_2) \in N$;
      $\Sigma$ := Match($e_1$,$e_2$, $FV(N)$);
      if $\Sigma = \emptyset$
        then mark $N$ with "F"
        else
          for each $\sigma \in \Sigma$ do
            begin
              $N'$ := Simplify($\sigma(N)$);
              Add a descendant arc from $N$ to $N'$ labelled by $\sigma$;
              if $N'$ is labelled "S"
                then begin
                  $\theta := Id$;
                  for each $\rho$ on path from $N'$ to root of $T$ do
                    $\theta := \rho \circ \theta$;
                  Output($\theta$)
                end
            end
    end
end.

function Simplify( $N$ : disSet ) : node;
{ Takes a disagreement set of pairs of terms of the same type and returns
  a node marked with "F" or "S", or a new disagreement set containing
  at least one flexible-rigid pair. }
begin
  { Dissolve all rigid-rigid pairs. }
  while exists rigid-rigid pair $(e_1, e_2)$ in $N$ do
    begin
      { Suppose $e_1 = \lambda x_1 \dots x_n.\, a_1(e^1_1, \dots, e^1_{p_1})$
            and $e_2 = \lambda y_1 \dots y_n.\, a_2(e^2_1, \dots, e^2_{p_2})$ }
      { See if heads same. }
      if not $(\lambda x_1 \dots x_n.\, a_1 \overset{*}{\longleftrightarrow}_\alpha \lambda y_1, \dots, y_n.\, a_2)$
        then Return($N$ marked with "F");
      { Else we know $\tau(a_1) = \tau(a_2)$ and thus $p_1 = p_2$ }
      Replace $(e_1, e_2)$ by the pairs
        $(\lambda x_1 \dots x_n.\, e^1_i, \lambda y_1 \dots y_n.\, e^2_i)$   for $1 \le i \le p_1$
    end;
  { Orient pairs. }
  while exists rigid-flexible pair $(e_1, e_2) \in N$ do
    Replace $(e_1, e_2)$ by $(e_2, e_1)$;
  if exists some flexible-rigid pair in $N$
    then Return($N$)
    else Return($N$ marked with"S")
end;
```

```
function Match($e_1$, $e_2$ : $\lambda-$terms; $V$ : setOfVars) : substSet;
{ Returns a set of substitutions which matches head of $e_1$ to head of $e_2$.
  $e_1$ is a flexible term $\lambda x_1 \ldots x_n . F(e^1_1, \ldots, e^1_{p_1})$
  and $e_2$ is a rigid term $\lambda y_1 \ldots y_n . a(e^2_1, \ldots, e^2_{p_2})$,
  where $\tau(e_1) = \tau(e_2) = \alpha_1, \ldots, \alpha_n \to \beta$. The set of unifiers
  returned is obtained by imitating the head of $e_2$ and
  by projecting $e_1$ on each of its arguments which preserves the type. }
var $\Sigma$ : substSet; i : integer;
begin
  { Imitate heading of $e_2$ if possible. }
  if Constant(a)
     then $\Sigma := \{ [\lambda z_1 \ldots z_{p_1} . a(G_1(z_1, \ldots, z_{p_1}), \ldots, G_{p_2}(z_1, \ldots, z_{p_1}))/F] \}$;
       { Where $\tau(z_i) = \tau(e^1_i)$, for $1 \le i \le p_1$, and the $G_j$ are
          variables not in V such that $\tau(G_j) = \tau(e^1_1), \ldots, \tau(e^1_{p_1}) \to \tau(e^2_j)$}
     else $\Sigma := \emptyset$;
  { Next project $F$ on each of its arguments which has appropriate type. }
  for $i := 1$ to $p_1$ do
     if $\tau(e^1_i) = \gamma_1, \ldots, \gamma_{m_i} \to \beta$ for some $\gamma_j$ { Note that possibly $m_i = 0$. }
       then
         $\Sigma := \Sigma \cup \{ [\lambda z_1 \ldots z_{p_1} . z_i(H^i_1(z_1, \ldots, z_{p_1}), \ldots, H^i_{m_i}(z_1, \ldots, z_{p_1}))/F] \}$;
           { Where $\tau(z_i) = \tau(e^1_i)$ for $1 \le i \le p_1$
              and the $H^i_j$ for $1 \le j \le m_i$ are variables not in V
              of type $\tau(H^i_j) = \tau(e^1_1), \ldots, \tau(e^1_{p_1}) \to \tau(e^2_j)$}
  Return($\Sigma$)
end;
```

7 References

[1] Andrews, P.B., "Resolution in Type Theory," JSL 36:3 (1971) 414-432.

[2] Andrews, P.B., "Theorem Proving via General Matings," JACM 28:2 (1981) 193-214.

[3] Andrews, P.B., D. Miller, E. Cohen, F. Pfenning, "Automating Higher-Order Logic," Contemporary Mathematics 29 (1984) 169-192.

[4] Andrews, P.B., *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, Academic Press, Inc. (1986).

[5] Barendregt, H.P., *The Lambda Calculus*, North-Holland (1984).

[6] Church, A., "A Formulation of the Simple Theory of Types," JSL 5 (1940) 56-68.

[7] Darlington, J.L., "A Partial Mechanization of Second-Order Logic," Machine Intelligence 6 (1971) 91-100.

[8] Elliot, C., and Pfenning, F., "A Family of Program Derivations for Higher-Order Unification," Ergo Report 87-045, CMU, November 1987.

[9] Fages, F., and Huet, G., "Complete Sets of Unifiers and Matchers in Equational Theories," TCS 43 (1986) 189-200.

[10] Farmer, W., *Length of Proofs and Unification Theory*, Ph.D. Thesis, University of Wisconsin—Madison (1984).

[11] Farmer, W. "A Unification Algorithm for Second-Order Monadic Terms," Unpublished Technical Report, MITRE Corporation, Bedford, MA.

[12] Felty, A., and Miller, D., "Specifying Theorem Provers in a Higher-Order Logic Programming Language," Ninth International Conference on Automated Deduction, Argonne, Illinois (1988).

[13] Gallier, J.H. *Logic for Computer Science: Foundations of Automatic Theorem Proving*, Harper and Row, New York (1986).

[14] Gallier, J.H., and Snyder, W., "A General Complete E-Unification Procedure," RTA, Bordeaux, 1987.

[15] Gallier, J.H., and Snyder, W., "Complete Sets of Transformations for General *E*-Unification," to appear in TCS (1989).

[16] Goldfarb, W., "The Undecidability of the Second-Order Unification Problem," TCS 13:2 (1981) 225-230.

[17] Gould, W.E., *A Matching Procedure for Omega-Order Logic*, Ph.D. Thesis, Princeton University, 1966.

[18] Guard, J.R., "Automated Logic for Semi-Automated Mathematics," Scientific Report 1, AFCRL 64-411, Contract AF 19 (628)-3250 AD 602 710.

[19] Guard, J., Oglesby, J., and Settle, L., "Semi-Automated Mathematics," JACM 16 (1969) 49-62.

[20] Hannan, J. and Miller, D., "Enriching a Meta-Language with Higher-Order Features," Workshop on Meta-Programming in Logic Programming, Bristol (1988).

[21] Hannan, J. and Miller, D., "Uses of Higher-Order Unification for Implementing Program Transformers," Fifth International Conference on Logic Programming, MIT Press (1988).

[22] Herbrand, J., "Sur la Théorie de la Démonstration," in: *Logical Writings*, W. Goldfarb, ed., Cambridge, 1971.

[23] Hindley, J., and Seldin, J., *Introduction to Combinators and Lambda Calculus*, Cambridge University Press (1986).

[24] Huet, G., "A Mechanization of Type Theory," Proceedings of the Third International Joint Conference on Artificial Intelligence (1973) 139-146.

[25] Huet, G., "A Unification Algorithm for Typed λ-Calculus," TCS 1 (1975) 27-57.

[26] Huet, G., *Résolution d'Equations dans les Langages d'Ordre* $1, 2, \ldots, \omega$, Thèse d'Etat, Université de Paris VII (1976).

[27] Huet, G., and Lang, B., "Proving and Applying Program Transformations Expressed with Second-Order Patterns," Acta Informatica 11 (1978) 31-55.

[28] Huet, G., "The Undecidability of Unification in Third-Order Logic," Information and Control 22 (1973) 257-267.

[29] Lucchesi, C.L., "The Undecidability of the Unification Problem for Third Order Languages," Report CSRR 2059, Dept. of Applied Analysis and Computer Science, University of Waterloo (1972).

[30] Martelli, A., Montanari, U., "An Efficient Unification Algorithm," ACM Transactions on Programming Languages and Systems 4:2 (1982) 258-282.

[31] Miller, D., *Proofs in Higher-Order Logic*, PhD. Dissertation, Carnegie-Mellon University, 1983.

[32] Miller, D., and Nadathur, G., "Higher-Order Logic Programming," Proceedings of the Third International Conference on Logic Programming, London (1986).

[33] Miller, D., and Nadathur, G., "A Logic Programming Approach to Manipulating Formulas and Programs," IEEE Symposium on Logic Programming, San Franciso (1987).

[34] Miller, D., and Nadathur, G., "Some Uses of Higher-Order Logic in Computational Linguistics," 24th Annual Meeting of the Association for Computational Linguistics (1986) 247—255.

[35] Nadathur, G., *A Higher-Order Logic as the Basis for Logic Programming*, Ph.D. Dissertation, Department of Computer and Information Science, University of Pennsylvania (1986).

[36] Paulson, L.C., "Natural Deduction as Higher-Order Resolution," Journal of Logic Programming 3:3 (1986) 237-258.

[37] Pfenning, F., *Proof Transformations in Higher-Order Logic*, Ph.D. thesis, Department of Mathematics, Carnegie Mellon University, Pittsburgh, Pa. (1987).

[38] Pfenning, F., "Partial Polymorphic Type Inference and Higher-Order Unification," in *Proceedings of the 1988 ACM Conference on Lisp and Functional Programming*, ACM, July 1988.

[39] Pfenning, F., and Elliott, C., "Higher-Order Abstract Syntax," *Proceedings of the SIGPLAN '88 Symposium on Language Design and Implementation*, ACM, June 1988.

[40] Pietrzykowski, T., "A Complete Mechanization of Second-Order Logic," JACM 20:2 (1971) 333-364.

[41] Pietrzykowski, T., and Jensen, D., "Mechanizing ω-Order Type Theory Through Unification," TCS 3 (1976) 123-171.

[42] Robinson, J.A., "Mechanizing Higher-Order Logic," Machine Intelligence 4 (1969) 151-170.

[43] Snyder, W.S., *Complete Sets of Transformations for General Unification*, Ph.D. Dissertation, Department of Computer and Information Science, University of Pennsylvania (1988).

[44] Statman, R., "On the Existence of Closed Terms in the Typed λ-Calculus II: Transformations of Unification Problems," TCS 15:3 (1981) 329-338.

[45] Winterstein, G., "Unification in Second-Order Logic," Electronische Informationsverarbeitung und Kybernetik 13 (1977) 399-411.

[46] Zaionc, Marek, "The Set of Unifiers in Typed λ-Calculus as a Regular Expression," Proceedings of the RTA 1985.