A GENERAL COMPLETE E-UNIFICATION PROCEDURE

Jean H. Gallier and Wayne Snyder[1]
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pa 19104

1 Introduction

In this paper, a general unification procedure that enumerates a complete set of E-unifiers of two terms for any *arbitrary* set E of equations is presented. It is more efficient than the brute force approach using paramodulation, because many redundant E-unifiers arising by rewriting at or below variable occurrences are pruned out by our procedure, still retaining a complete set. This procedure can be viewed as a non-deterministic implementation of a generalization of the Martelli-Montanari method of transformations on systems of terms [13], which has its roots in Herbrand's thesis [7]. Remarkably, only two new transformations need to be added to the transformations used for standard unification. This approach differs from previous work based on transformations because, rather than sticking rather closely to the Martelli-Montanari approach using multi-equations [13] as in Kirchner [10,11], we introduce transformations dealing directly with rewrite rules.

As an example of the flexibility of this approach, we apply it to the problem of higher-order unification, and find an improved version of Huet's procedure [8]. Our major new result is the presentation and justification of a method for enumerating (relatively minimal) complete sets of unifiers modulo arbitrary sets of equations.

2 Preliminaries

It is assumed that the reader is familiar with the standard definitions and notations for many-sorted languages, tree domains, tree replacements, substitutions, and rewrite steps (as found in, e.g., [9] and [5]); we will present here only those definitions relevant to our approach to E-unification.

Definition 2.1 Given a set of equations E, we say that a substitution θ is an *E-unifier of u and v*, iff $\theta(u) \overset{*}{\longleftrightarrow}_E \theta(v)$, that is

$$\theta(u) = u_0 \longrightarrow_{[\alpha_1, l_1 \doteq r_1, \rho_1]} u_1 \ \cdots \ u_{n-1} \longrightarrow_{[\alpha_n, l_n \doteq r_n, \rho_n]} u_n = \theta(v),$$

where the α_i are the rewrite addresses, the $(l_i \doteq r_i)$ are variants of equations in $E \cup E^{-1}$, and the ρ_i are the matching substitutions used in each step. We will often find it useful to consider an *extension* of θ which incorporates all the matching substitutions, i.e., $\theta' = \theta \circ \rho_1 \circ \ldots \circ \rho_n$; note that $\theta'(u) \overset{*}{\longleftrightarrow}_E \theta'(v)$. We will assume that all substitutions are idempotent, i.e., that $\theta = \theta \circ \theta$.

The set of all E-unifiers of u and v is denoted $U_E(u, v)$. It is well known that the set $U_E(u, v)$ is only semi-decidable, and that if two terms are unifiable modulo E, there is in general no single *mgu*, but instead a possibly infinite set.

[1] This research was partially supported by the National Science Foundation under Grant No. DCR-86-07156.

Definition 2.2 Two substitutions σ and θ are termed *equal modulo E over W*, denoted by $\sigma =_E \theta[W]$, iff $\forall x \in W$, $\sigma(x) \stackrel{*}{\longleftrightarrow}_E \theta(x)$. We say that σ is *more general than θ over W*, denoted by $\sigma \leq_E \theta[W]$, iff there exists some substitution η such that $\theta =_E \sigma \circ \eta[W]$, i.e., that $\forall x \in W$, $\theta(x) \stackrel{*}{\longleftrightarrow}_E \eta(\sigma(x))$.

Now we generalize the concept of a *mgu* to E-unifiers; this formulation of a generating set for a set of E-unifiers is due to [16]. The following definition is from [3].

Definition 2.3 Let $Var(u)$ denote the set of variables occurring in the term u. Let $D(\theta) = \{x \mid \theta(x) \neq x\}$ and $I(\theta) = \bigcup_{x \in D(\theta)} Var(\theta(x))$. Given a finite set E of equations, for any two terms u, v, let $Var(u,v) = Var(u) \cup Var(v)$, and let W be any finite set of variables such that $(Var(u,v)) \cap W = \emptyset$. A set U of substitutions is a *complete set of E-unifiers for u and v away from W*, for short, a CSU_E of u and v iff

1. $\forall \theta \in U$, $D(\theta) \subseteq Var(u,v)$, and $I(\theta) \cap (D(\theta) \cup W) = \emptyset$;
2. $\forall \theta \in U$, $\theta(u) \stackrel{*}{\longleftrightarrow}_E \theta(v)$;
3. $\forall \theta \in U_E(u,v)$, $\exists \sigma \in U$ such that $\sigma \leq_E \theta[Var(u,v)]$.

We will consider the E-unification of *pairs* and *systems*, following the notations introduced by [13] and [10].

Definition 2.4 A (unordered) *pair* is simply a multiset of two terms, and a substitution θ is an E-unifier of a pair $\langle u, v \rangle$ iff $\theta(u) \stackrel{*}{\longleftrightarrow}_E \theta(v)$. A *system* is a set of pairs, and a substitution is an E-unifier of a system iff it is an E-unifier of each pair in the system.

In the sequel, we will denote constants by a; functions by f, g, h, and k; variables by w, x, y, and z; terms by l, r, u, v; and systems by S and R.

3 E-Unification via Transformations

We now develop the notion of an E-unification procedure as a series of transformations on systems which converts a pair of terms into a *solved form* which explicitly represents the E-unifier.[1] A pair $\langle x, t \rangle$ is in *solved form* in a system S if t is any term and x is a variable which does not occur anywhere else in S (in particular, $x \notin Var(t)$); a system is in solved form if all its pairs are in solved form. It should be clear that a solved system $S = \{\langle x_1, t_1 \rangle, \ldots, \langle x_n, t_n \rangle\}$ has a *mgu* $\theta_S = [t_1/x_1, \ldots, t_n/x_n]$, and that $\{\theta_S\}$ is a CSU_E for S. This allows us to effectively ignore any E-unifiers which use rewrite steps between solved pairs if we are just interested in complete sets of unifiers.

We now show how E-unifiers can be obtained using transformations on systems. The relation $\Longrightarrow$ on systems is defined as follows:

Definition 3.1 (Transformation Rules) Let $|t|$ denote the *depth* of a term t (e.g., $|a| = 0$ and $|f(x)| = 1$). Let R denote any system (possibly empty), f be any function symbol of arity n, and u, v be two terms. We have the following transformations:

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

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

$$\{\langle x, v \rangle\} \cup R \Longrightarrow \{\langle x, v \rangle\} \cup R[v/x], \tag{3}$$

[1] It is remarkable that in his thesis, Herbrand gave all the steps of a (nondeterministic) unification 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 [7]. Apparently, this algorithm was rediscovered by a number of people, including Huet [8], Martelli and Montanari [13], and Kozen [12].

where x is a variable, $x \notin Var(v)$, $x \in Var(R)$, and $R[v/x]$ is the system obtained by substituting v for all occurrences of x in R.

Transformations (1)–(3) are essentially those given by Herbrand [7] and Martelli-Montanari [13], except that we need not orient the pairs, since they are unordered. We need two more transformations to deal with equations. Let $\langle u, v\rangle$ be a pair such that u is not a variable and either v is not a variable or v is a variable such that $v \in Var(u)$. Then

$$\{\langle u, v\rangle\} \cup R \Longrightarrow \{\langle u, l\rangle, \langle r, v\rangle\} \cup R, \tag{4}$$

where $(l \doteq r)$ is a variant of an equation in $E \cup E^{-1}$ such that $Var(l, r) \cap (Var(R) \cup Var(u, v)) = \emptyset$, and if l is not a variable, then $Root(u) = Root(l)$. Transformation (4) may not be applied hereafter to the pair $\langle u, l\rangle$. This transformation represents a *leftmost* rewrite step at the root, and prohibits rewriting a variable occurrence, unless the *occur check* fails. Finally, if $x \in Var(f(v_1, \ldots, v_n))$, then

$$\{\langle x, f(v_1, \ldots, v_n)\rangle\} \cup R \Longrightarrow \{\langle x, f(y_1, \ldots, y_n)\rangle, \langle f(y_1, \ldots, y_n), f(v_1, \ldots, v_n)\rangle\} \cup R \tag{5},$$

where the $y_1, \ldots, y_n$ are new variables, and rule (3) is immediately applied to the pair $\langle x, f(y_1, \ldots, y_n)\rangle$ and then (2) is applied to the pair $\langle f(y_1, \ldots, y_n), f(v_1, \ldots, v_n)\rangle$.

The motivation for rule (5) is in E-unifying a pair of the form $\langle x, f(v_1, \ldots, v_n)\rangle$, where the *occur check* fails for x. Although such a pair cannot have a *mgu*, it is potentially E-unifiable by rewriting at the root (e.g., $[a/x] \in U_{\{a \doteq f(a)\}}(x, f(x))$) or by rewriting below the root (e.g., $[f(a)/x] \in U_{\{a \doteq g(f(a))\}}(x, f(g(x)))$). In the latter case, any E-unifier must match the root symbol of the second term, and so be of the form $[f(t_1, \ldots, t_n)/x]$ for some terms $t_1, \ldots, t_n$. We simulate rewriting below the root by successively imitating the root of v and decomposing, thus distributing the occur check into at least one of the pairs $\langle y_1, v_1\rangle, \ldots, \langle y_n, v_n\rangle$, whereupon we may apply (4) or (5) again to that pair. At some point we must find an application of (4) if we are to eliminate the occur check.[2]

Unfortunately, it is possible to create an infinite series of pairs isomorphic up to renaming by repeatedly applying (5): $\{\langle x, f(x)\rangle\} \stackrel{*}{\Rightarrow} \{\langle x, f(x)\rangle, \langle y_1, f(y_1)\rangle\} \stackrel{*}{\Rightarrow} \{\langle x, f(x)\rangle, \langle y_1, f(y_1)\rangle, \langle y_2, f(y_2)\rangle\} \ldots$. In Lemmas 4.4 – 4.7, we shall prove that we can preserve completeness if we restrict the applications of (5) used in finding an address at which to apply (4) to (the finite number of) addresses along the path from the root to an occurrence of x in the original term $f(v_1, \ldots, v_n)$. Our choice of (5) was motivated by the general *top-down* flavor of our method. A rule combining rules (4) and (5) is also used in [14].

Thus, given a set of equations E and a system S to be E-unified, we have the following

Non-Deterministic Procedure: Transform the system S into a solved form, if one exists:

$$S \Longrightarrow S_1 \Longrightarrow \ldots \Longrightarrow S_n.$$

Return $mgu(S_n)|_{Var(S)}$.[3]

Example 3.2 Let $E = \{\ [1]\, f(z_1) = g(z_1),\ \ [2]\, h(f(z_2)) \doteq k(f(z_2)), [3]\, z_3 \doteq f(g(z_3))\ \}$, and $S = \{\langle f(x, h(g(y))),\ f(g(f(x)), k(g(y)))\rangle\}$. The following sequence of transformations leads to a system in solved form.

$$\begin{aligned} &\{\langle f(x, h(g(y))),\ f(g(f(x)), k(g(y)))\rangle\} \\ &\Longrightarrow \{\langle x, g(f(x))\rangle, \langle h(g(y)), k(g(y))\rangle\} \qquad \text{by (2)} \end{aligned}$$

[2] C.f. transformation (5) with the use of the *functional reflexivity* axioms in paramodulation.

[3] This denotes the restriction of the substitution to the set $Var(S)$.

$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle g(y_1), g(f(x))\rangle, \langle h(g(y)), k(g(y))\rangle\} \quad \text{by (5)}$$
$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle g(y_1), g(f(g(y_1)))\rangle, \langle h(g(y)), k(g(y))\rangle\} \quad \text{by (3)}$$
$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle y_1, f(g(y_1))\rangle, \langle h(g(y)), k(g(y))\rangle\} \quad \text{by (2)}$$
$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle f(g(y_1)), f(g(z_3))\rangle, \langle z_3, y_1\rangle, \langle h(g(y)), k(g(y))\rangle\} \quad \text{by (4), eq.[3]}$$
$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle f(g(y_1)), f(g(y_1))\rangle, \langle z_3, y_1\rangle, \langle h(g(y)), k(g(y))\rangle\} \quad \text{by (3)}$$
$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle z_3, y_1\rangle, \langle h(g(y)), k(g(y))\rangle\} \quad \text{by (1)}$$
$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle z_3, y_1\rangle, \langle h(g(y)), h(f(z_2))\rangle, \langle k(f(z_2)), k(g(y))\rangle\} \quad \text{by (4), eq.[2]}$$
$$\stackrel{*}{\Longrightarrow} \{\langle x, g(y_1)\rangle, \langle z_3, y_1\rangle, \langle g(y), f(z_2)\rangle\} \quad \text{by (2)}$$
$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle z_3, y_1\rangle, \langle g(y), g(z_1)\rangle, \langle f(z_1), f(z_2)\rangle\} \quad \text{by (4), eq.[1]}$$
$$\stackrel{*}{\Longrightarrow} \{\langle x, g(y_1)\rangle, \langle z_3, y_1\rangle, \langle y, z_1\rangle, \langle z_1, z_2\rangle\} \quad \text{by (2)}$$
$$\Longrightarrow \{\langle x, g(y_1)\rangle, \langle z_3, y_1\rangle, \langle y, z_2\rangle, \langle z_1, z_2\rangle\} \quad \text{by (3)}$$

Hence, $[g(y_1)/x, z_2/y]$ is an E-unifier of S. The substitution $[g(y_1)/x, y_1/z_3, z_2/y, z_2/z_1]$ is the *extended* E-unifier of S. In the fourth line from the bottom note that, since pairs are unordered and the system is a set, two pairs have collapsed into one.

4 Soundness and Completeness

It is a testament to the power and elegance of the technique of unification by transforming systems of terms that it can be adapted to E-unification by adding only two additional transformations, and that this method, as we prove in this section, can non-deterministically find a CSU_E of u and v for *arbitrary* theories E.

The obvious strategy for proving completeness is to take some "proof" of the fact that $\theta(u) \stackrel{*}{\longleftrightarrow}_E \theta(v)$, and let its structure determine the sequence of transformations; we then show that the substitution σ found by the procedure is such that $\sigma \leq_E \theta[Var(u, v)]$. However, in arbitrary theories we cannot assume (as in [14]) that substitutions are reduced, and furthermore it is not clear what is decreasing in the rewrite proof as transformations are applied. Finding the appropriate measure for the induction turned out to be non-trivial.[4] Two essential issues in proving completeness were (A) finding a *variable pure* representation for rewrite proofs in which transformation (3) does not increase the number of rewrite steps in the representation and does not allow variables introduced by rewrites to interfere with each other, and (B) removing the potential for infinite recursion on (5) in attempting to eliminate the occur check. We shall present our soundness result after explaining the proof representation we have developed, and finish the section with our completeness theorem.

Given a system $S = \{\langle u_1, v_1\rangle, \ldots, \langle u_n, v_n\rangle\}$ and a substitution θ, we will represent the "proof" that $\theta \in U_E(S)$ as a certain kind of DAG (Directed Acyclic Graph). The nodes of these proof DAGs are labeled with two terms connected by one of the symmetric relations $\approx$ or $\sim$.

Definition 4.1 Given an idempotent substitution θ, the set of *proof DAGs* associated with θ is defined inductively as follows.

(i) (Axioms) For every term u, the one node tree labeled with $u \approx u$ is a proof DAG associated with θ. For every variable x and term v such that $x \notin Var(v)$ and $\theta(x) = \theta(v)$, the one node tree labeled with $x \approx v$ is a proof DAG associated with θ.

[4] For example, (4) and (5) decrease the number of rewrite rules in the rewrite proof and increase the number of unsolved variables; (3) eliminates unsolved variables in the rewrite proof but potentially increases the number of rewrites.

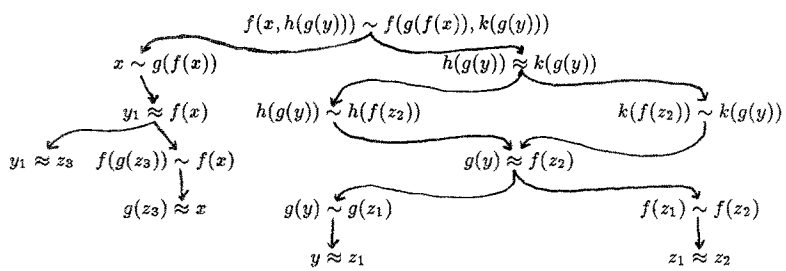
(ii) (Term decomposition) Given a pair $\langle u, v\rangle$ such that u and v are compound terms whose roots are labeled with the same symbol f, let $\theta(u) = f(\theta(u_1), \ldots, \theta(u_k))$ and $\theta(v) = f(\theta(v_1), \ldots, \theta(v_k))$, where $u_i = u/i$ (resp. $v_i = v/i$). Given any k proof DAGs $T_1, \ldots, T_k$ associated with θ, where each T_i is a proof DAG whose root is labeled with $u_i \approx v_i$ (or $u_i \sim v_i$), the DAG T whose root is labeled with $u \sim v$ and such that $T/i = T_i$, $1 \leq i \leq k$, is a proof DAG associated with θ.

(iii) (Rewrite rule insertion) Given a pair $\langle u, v\rangle$ of terms, given any m variants $(l_i \doteq r_i)$, $1 \leq i \leq m$, of equations from $E \cup E^{-1}$, given any $m+1$ proof DAGs $T_1, \ldots, T_{m+1}$ associated with θ, where T_1 is either an axiom labeled with $u \approx l_1$, or a proof DAG whose root is labeled with $u \sim l_1$, for each i, $2 \leq i \leq m$, T_i is either an axiom labeled with $r_{i-1} \approx l_i$, or a proof DAG whose root is labeled with $r_{i-1} \sim l_i$, and T_{m+1} is either an axiom labeled with $r_m \approx v$, or a proof DAG whose root is labeled with $r_m \sim v$, the DAG T whose root is labeled with $u \approx v$ and such that $T/i = T_i$, $1 \leq i \leq m+1$, is a proof DAG associated with θ.

A *proof DAG* for a pair $\langle \theta, u \approx v\rangle$ or $\langle \theta, u \sim v\rangle$ is a DAG obtained by merging identical nodes in any proof DAG associated with θ and whose root is labeled with $u \approx v$ or $u \sim v$.[5] (We will omit θ when available from context. Note that, since these relations are symmetric, $u \approx v$ is the same node as $v \approx u$.) A *proof DAG* for a pair $\langle \theta, S\rangle$, where θ is a substitution and S is a system, is the DAG obtained by merging identical nodes in any proof DAGs for the pairs in S. A proof DAG T for a system S is *variable pure* iff the sets of variables occurring in the equations used in the rewrite steps implicit in T are pairwise disjoint, and these variables are also disjoint from $Var(S)$.

Note that in case (ii) and case (iii), $\langle u' \sim v'\rangle$, as contrasted with $\langle u' \approx v'\rangle$, indicates that no rewrite step is applied at the root of any DAG in the sequence $\theta(u') \overset{*}{\longleftrightarrow}_E \theta(v')$. We omit here the formal proofs of the soundness and completeness of this representation, since this should be intuitively obvious. A minor technical detail of the latter is that $\theta(u) \overset{*}{\longleftrightarrow}_E \theta(v)$ implies that there exists a variable pure DAG associated with an extended θ' incorporating all the rewrite substitutions in $\overset{*}{\longleftrightarrow}_E$.

For example, the DAG associated with the substitution $[g(y_1)/x, y_1/z_3, z_2/y, z_2/z_1]$ found in Example 3.2 is as follows.



We need one lemma before we state our soundness result.

Lemma 4.2 Let E be a set of equations, D_2 be any system of DAGs with $x \in Var(D_2)$, and let D_1 be a DAG whose root is labeled with $\langle x \approx v\rangle$ and such that $x \notin Var(v)$. Then $\{D_1 \cup D_2\}$ is a variable pure DAG system associated with θ iff $\{D_1 \cup D_2[v/x]\}$ is a variable pure DAG system associated with θ.

Proof. This result assures us (A) that transformation (3) produces equivalent systems, and (B) that this process will not increase the number of rewrite steps implicit in the proof. The proof depends on the

[5] This structure sharing is not just a way of reducing the size of the DAGs; it is used to prevent the number of rewrite steps implicit in the DAG from increasing in transformation (3).

(recursive) definition of substitution in a DAG, where $D_2[v/x]$ is the DAG system obtained by substituting v for x in interior positions in the nodes and sharing the structure of the DAG D_1 elsewhere (the details are tedious and involve the examination of nine cases on the structure of the nodes in D_1 and D_2). □

The following lemma shows that our procedure is sound.

Theorem 4.3 (Soundness) If $S \stackrel{*}{\Longrightarrow} S_n$, with S_n in solved form, then $mgu(S_n)|_{Var(S)} \in U_E(S)$.

Proof. We show that if $S \Longrightarrow S'$ using transformation (1) or (3), then $U_E(S) = U_E(S')$; and if $S \Longrightarrow S'$ using one of (2), (4), or (5), then $U_E(S') \subseteq U_E(S)$. The only difficulty might arise in transformation (3); we use Lemma 4.2. The soundness of the method follows by a trivial induction. □

We now discuss the bound on the number of applications of (5) in eliminating the occur check. The following lemma, based on a simpler result of Kozen,[6] is the key to the elimination of the variable x in the case of a pair $\langle x, v\rangle$, where $|v| \geq 1$ and $x \in Var(v)$.

Lemma 4.4 Given a set E of equations, given any term v containing some occurrence of a variable x, and such that $|v| \geq 1$, if there is a term t with no occurrence of x such that $v[t/x] \stackrel{*}{\longleftrightarrow}_E t$, then there is some subterm r of t such that $r \stackrel{*}{\longleftrightarrow}_E t$, $v[r/x] \stackrel{*}{\longleftrightarrow}_E r$, and, in the sequence of rewrite steps $v[r/x] \stackrel{*}{\longleftrightarrow}_E r$, for every occurrence α of the variable $x \in dom(v)$, some rewrite rule is applied to a proper ancestor β of α.

We omit the proof, which proceeds by induction on $(k, |t|)$, where k is the number of occurrences of x in v (using the lexicographic ordering).

The condition on the application of rewrite rules given by lemma 4.4 implies the next lemma. The following definition will be helpful.

Definition 4.5 Given a tree u, given any two sequences of independent addresses $A = \langle \beta_1, \ldots, \beta_n\rangle$ and $B = \langle \alpha_1, \ldots, \alpha_k\rangle$ of addresses in $dom(u)$, where $n \leq k$, A is a *proper cover* of B iff for every $\beta \in B$, there exists some $\alpha \in A$ such that α is a proper prefix of β. (That is, every path from the root to some β contains a $\alpha \neq \beta$.)

Lemma 4.6 Let E be a set of equations, $\langle x, v\rangle$ be a pair, where $|v| \geq 1$ and $x \in Var(v)$, and let $\langle \alpha_1, \ldots, \alpha_k\rangle$ be the sequence of all occurrences of x in v. The following properties are equivalent.

(1) $\{\langle x \approx v\rangle\} \cup D$ is a variable pure DAG associated with θ.

(2) There is some substitution σ, some sequence of addresses $\langle \beta_1, \ldots, \beta_n\rangle$ that is a proper cover of $\langle \alpha_1, \ldots, \alpha_k\rangle$, and some sequence $(l_1 \doteq r_1), \ldots, (l_n \doteq r_n)$ of variants of equations from $E \cup E^{-1}$ renamed away from $Var(v) \cup Var(D)$, such that $\sigma =_E \theta[Var(v) \cup Var(D)]$, and σ is the substitution associated with the DAG system

$$\{\langle x \approx v[\beta_1 \leftarrow r_1, \ldots, \beta_n \leftarrow r_n]\rangle\} \cup \{\langle v/\beta_1 \approx l_1\rangle, \ldots, \langle v/\beta_n \approx l_n\rangle\} \cup D',$$

where $v[\beta_1 \leftarrow r_1, \ldots, \beta_n \leftarrow r_n]$ denotes the result of replacing each subtree at β_i in v by r_i, and D' is obtained by altering the structure of the corresponding DAGs in D.

Proof. We use lemma 4.4 to show that there is always a substitution σ *small enough* so that there is a sequence of rewrite steps between $\sigma(v)$ and $\sigma(x)$ such that for every occurrence α of x in v, some rewrite rule is applied to a proper ancestor β of α. The technical difficulties have to do with showing that a variable pure proof DAG can be obtained, without using new variants of equations. □

The next lemma shows that rules (3), (4) and (5) are sufficient to eliminate a variable causing the occur check condition.

[6] In [12], lemma 7. His result is sufficient only for the case of ground equations.

Lemma 4.7 If σ is an E-unifier of the system

$$S = \{(x, v[\beta_1 \leftarrow r_1, \ldots, \beta_n \leftarrow r_n])\} \cup \{(v/\beta_1, l_1), \ldots, (v/\beta_n, l_n)\} \cup R,$$

then, letting $v' = v[\beta_1 \leftarrow y_1, \ldots, \beta_n \leftarrow y_n]$, where $\{y_1, \ldots, y_n\}$ is a set of new variables, using a finite number of applications of the transformations (3), (4), and (5), one can obtain a system S' and a substitution σ' extending σ, such that σ' is an E-unifier of

$$S' = \{(x, v')\} \cup R' \cup \{((v/\beta_1)[v'/x], l_1), (r_1, y_1), \ldots, ((v/\beta_n)[v'/x], l_n), (r_n, y_n)\} \cup R.$$

(R' is a system in solved form involving some new variables.) The variables $y_1, \ldots, y_n$ can now be eliminated using rule (3).

Finally, we are ready to state the major result of the paper. The completeness of our method is shown in the following theorem.

Theorem 4.8 (Completeness) For every $\theta \in U_E(S)$, there exists a series of transformations $S \stackrel{*}{\Longrightarrow} \widehat{S}$ such that $\widehat{S}$ is in solved form, and $mgu(\widehat{S})|_{Var(S)} \leq_E \theta[\mathrm{Var(S)}]$.

Proof. Given $\theta \in U_E(S)$, first we extend the substitution θ, and then construct a variable pure proof DAG D for $\langle \theta, S \rangle$. We use S to dictate the sequence of transformations in generating $\widehat{S}$. We assign a measure of complexity $\mu(D, S, \theta)$ to triples of the form (D, S, θ), where $\mu(D, S, \theta)$ is an element of a well-founded set (with ordering $\prec$), and we show that if S is not in solved form, it is possible to construct a triple (D', S', θ') such that, $S \stackrel{*}{\Longrightarrow} S'$, $\mu(D', S', \theta') \prec \mu(D, S, \theta)$, and θ' extends θ. The well founded set is the set of quintuples $\langle M, n_1, n_2, n_3, n_4 \rangle$, where M is a multiset of natural numbers measuring the complexity of θ, n_1 the total number of variables in S, n_2 the total number of equations in D, n_3 the size of the terms occurring in the pairs labeling root nodes of D, and n_4 the number of pairs $\langle u, u \rangle$ in S. The ordering $\prec$ is the lexicographic ordering on quintuples, where the ordering on the first component is the multiset ordering, and the ordering on the other components is the ordering on the natural numbers. There is a subtlety regarding the measure M associated with a substitution θ: M is the multiset of depths of trees of the form $\theta(x)$, for every variable x in the domain of θ that does *not* belong to a pair $\langle x, v \rangle$ which is already solved in the system S. Similarly, n_1 only takes into account variables that do not belong to solved pairs in S. □

Combining theorem 4.3 and theorem 4.8, we obtain the fact that our method yields complete sets of E-unifiers.

Theorem 4.9 The set $\{mgu(\widehat{S})|_{Var(S)} \mid S \stackrel{*}{\Longrightarrow} \widehat{S}$, and $\widehat{S}$ is in solved form$\}$ is a CSU_E for any system S.

5 Implementation of a Deterministic Procedure

In this section, we design a deterministic procedure by emphasizing a distinction implicit in our transformations, viz., whether a rewrite takes place at the root (transformation (4)) or not (the other transformations). Specifically, if u and v are E-unifiable, then $\forall \theta \in U_E(u, v)$ there exists a sequence $\theta(u) = u_0 \longrightarrow_E u_1 \longrightarrow_E \ldots \longrightarrow_E u_n = \theta(v)$. Each such θ can be classified into at least one, and possibly both, of the following two cases. (Case 1 is further divided into five mutually exclusive cases based on the stucture of the terms.)

1. No rewrite rule is applied at the root of any u_i.
   (a) Both u and v are compound terms, e.g., $u = f(u_1, \ldots, u_n)$ and $v = f(v_1, \ldots, v_n)$. Thus $\theta(u_i) \stackrel{*}{\longleftrightarrow}_E \theta(v_i)$ for $1 \leq i \leq n$.

(b) Either u or v is a variable; assume u is a variable.

i. v is a constant or a variable.

ii. v is a compound term.

A. $u \notin Var(v)$.

{ Then $[v/u] = mgu(u, v) \leq_E \theta$}

B. $u \in Var(v)$. Thus, if $v = f(v_1, \ldots, v_n)$ then $\theta(u) = f(t_1, \ldots, t_n)$ for some terms $t_1, \ldots, t_n$.

(c) Both u and v are constants, i.e., $u = v$.

2. Some rewrite rule is applied at the root of some u_i. Thus

$$\theta(u) \overset{*}{\longleftrightarrow}_E \rho(l) \longleftrightarrow_E \rho(r) \overset{*}{\longleftrightarrow}_E \theta(v),$$

where $(l \doteq r)$ is a variant of an equation in $E \cup E^{-1}$, ρ is the matching substitution used in the rewrite step, and no rewrite at the root takes place between $\theta(u)$ and $\rho(l)$.

The following Pseudo-Pascal procedure recursively applies this classification to two terms, and it may be seen that it is both an implementation of the set of transformations given earlier and an extension of Robinson's original algorithm for standard unification [17] to the case of E-unification.

```
global variables
  currDepth, maxDepth : integer;   E : eqSet;

procedure E-Unifiers( u, v : term );
  begin
    for maxDepth := 1 to ∞ do
      begin
        currDepth := 0;
        output( E-Unifs(u, v, false, false ) )
      end
  end;


function E-Unifs( u, v : term; occur, noRootRW : boolean ) : unifSet;
var
  unifs1, unifs2, subUnifs : unifSet;  i, n : integer;  θ, σ : unifier;
  f : funcSymbol; y1, ..., yn : variables;
begin
  currDepth := currDepth + 1;
  if currDepth > maxDepth
    then return(∅);              { Terminate this call and return ∅ }

{ Case 1:  Find unifiers of u and v which don't involve rewriting root and collect in unifs1 }

  if u = v
    then unifs1 := {Id}              { This includes Case 1.(c) }
  else if |u| > 0 and |v| > 0 and (Root(u)=Root(v))      { Case 1.(a) }
    then begin
      unifs1 := E-Unifs( u/1, v/1, false, false );
      for i := 2 to Arity(Root(u)) do
        begin
          subUnifs := ∅;
          for each θ ∈ unifs1 do
            subUnifs := subUnifs ∪ θ ∘ E-Unifs(θ(u/i), θ(v/i), false, false);
          unifs1 := subUnifs
        end
    end
  else if Variable(u) or Variable(v)
    then begin
      if not Variable(u)
        then Swap(u, v);
      if |v| = 0 or (|v| > 0 and (u ∉ Vars(v)))
        then begin            { Cases 1.(b).i to 1.(b).ii.B }
```

```
        unifs1 := {[v/u]};
        noRootRW := true
      end
      else begin                { Case 1.(b).ii.B }
        if not occur            { start of new occur check case found }
          then Mark all addresses α ∈ Dom(v) where v(α) = u;
        if Marked( v )
          then begin
            unifs1 := ∅;
            noRootRW := true
          end
          else begin
            f := Root(v);
            n := Arity(f);
            θ := [f(y1, ..., yn)/u];        { where the yi are new variables }
            unifs1 := θ ∘ E-Unifs(θ(u), θ(v), true, true);
          end
      end {else}
  end {then}
  else unifs1 := ∅;

{ Case 2: Find unifiers which involve rewriting u and v at the root and collect in unifs2 }

  if Id ∈ unifs1 or noRootRW
    then unifs2 := ∅
    else begin
      if Variable(u)
        then Swap(u, v);
      unifs2 := ∅;
      for each (l ≐ r) ∈ E ∪ E^-1 where l is a variable or Root(u) = Root(l) do
        for each θ ∈ E-Unifs(u, l, false, true ) do
          unifs2 := unifs2 ∪ θ ∘ E-Unifs(θ(r), θ(v), false, false );
    end;

  currDepth := currDepth - 1;
  return( unifs1 ∪ unifs2 )
end; { E-Unifs }
```

The procedure works roughly as follows. *MaxDepth* controls the maximum recursion depth in each loop iteration and *currDepth* contains the depth of the current call; this amounts to a breadth-first traversal of the search space of all E-unifiers. The procedure will generate an infinite chain $U_1 \subseteq U_2 \subseteq U_3 \subseteq \ldots$ of sets of E-unifiers indexed by *maxDepth*. The flag *occur* indicates that the current call is part of a (finitely recursing) occur check case; the marking of addresses serves to prevent recursion past the occurrence of a variable which caused an occur check: as shown in lemmas 4.4 – 4.7, we can guarantee that no new E-unifiers will be found after this imitation reaches the occur check variable.

Note that the flag *noRootRW* is set to **true** in the recursive call in case 2 to force a leftmost root rewrite, and is also used to eliminate attempts to rewrite if *Id* is found in case 1 (c.f. transformation (1)) or if either case 1.b.i or 1.b.ii.A has occurred (c.f. transformation (3)), since in fact this is unnecessary for completeness. The code contains other efficiency heuristics. For example, in case 1.(a), we successively apply the substitutions found for the previous subterms to later ones, possibly reducing the set *subUnifs* each time. In case 2 we attempt to rewrite non-variables to variables (rather than the reverse) whenever possible, and many useless rewrite sequences are therefore eliminated. These measures reduce considerably the running time of the procedure for the various levels of the search space, and allow us to find E-unifiers for non-trivial problems.

6 Higher-Order Unification via Transformations

Another testament to the power of this representation for unification problems is that Higher Order Unification may be described in terms of transformations, and that our results concerning the elimination of the occur check suggest a means of eliminating useless paths in the search for unifiers in a novel manner. Higher Order Unification is a method for unifying terms in the Simple Theory of Types [2], 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 $\sigma(e_1) = \sigma(e_2)$, modulo α-conversion (renaming of bound variables). In this section, we extend our method to higher order unification.

We follow [8] and assume the standard definition of terms, bound and free variables, substitutions, reductions, normal forms, and the simple types.[7] We consider here only unification assuming *η-equivalence*, i.e., $\lambda x(ex) = e$ if $x \notin FreeVar(e)$. All terms are assumed to be expanded into *η-normal form*; for example, a term $\lambda x_1 \ldots x_n.@(e_1, \ldots, e_m)$ will be of some type $\alpha_1, \ldots, \alpha_n \rightarrow \beta$, with β a base type.

Definition 6.1 We now present Huet's procedure cast in the form of transformations on a system S of pairs of λ-terms. (Assume that the type of the pair mentioned in each transformation is $\alpha_1, \ldots, \alpha_n \rightarrow \beta$.) We have the following rules:

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

$$\{\langle e_1, e_2\rangle\} \cup S \Longrightarrow \{\langle e_2, e_1\rangle\} \cup S, \tag{2}$$

where e_1 is rigid and e_2 is flexible;

$$\{\langle \lambda x_1 \ldots x_n.@_1(e_1^1 \ldots e_{p_1}^1), \lambda y_1 \ldots y_n.@_2(e_1^2 \ldots e_{p_2}^2)\rangle\} \cup S \Longrightarrow$$

$$\bigcup_{1 \le i \le p_1} \{\langle \lambda x_1 \ldots x_n.e_i^1, \lambda y_1 \ldots y_n.e_i^2\rangle\} \cup S, \tag{3}$$

where $@_2 = \left((\lambda x_1 \ldots x_n.@_1)\, y_1 \ldots y_n\right)$. If $\langle \lambda x_1 \ldots x_n.F(e_1^1 \ldots e_{p_1}^1), \lambda y_1 \ldots y_n.@(e_1^2 \ldots e_{p_2}^2)\rangle \in S$, where $@$ is a rigid head,

$$S \Longrightarrow \{\langle F, \lambda z_1 \ldots z_{p_1}.@(G_1(z_1 \ldots z_{p_1}) \ldots G_{p_2}(z_1 \ldots z_{p_1}))\rangle\} \cup S, \tag{4}$$

where the G_i are new variables of the appropriate type, and rule (6) is immediately applied to the new pair. If $\langle \lambda x_1 \ldots x_n.F(e_1^1 \ldots e_{p_1}^1), \lambda y_1 \ldots y_n.@(e_1^2 \ldots e_{p_2}^2)\rangle \in S$, where $@$ is a rigid head and for some subterm e_i^1, $\tau(e_i^1) = \gamma_1 \ldots \gamma_k \rightarrow \beta$,

$$S \Longrightarrow \{\langle F, \lambda z_1 \ldots z_{p_1}.z_i(H_1(z_1 \ldots z_{p_1}) \ldots H_k(z_1 \ldots z_{p_1}))\rangle\} \cup S, \tag{5}$$

where the H_i are new variables of the appropriate type. Apply rule (6) to the new pair. Finally, we have

$$\{\langle x, e\rangle\} \cup S \Longrightarrow \{\langle x, e\rangle\} \cup S[e/x], \tag{6}$$

where x is a variable not in $FreeVar(e)$. (We assume that β-reduction is performed as part of the substitution process.) Since Huet's procedure uses ordered pairs, we have included rule (2). Rule (4) corresponds to Imitation, and (5) to Projection.

Note that the left term in the new pairs introduced by 4 and 5 is not in η-normal form; since these are immediately transformed into solved forms, this should cause no confusion. A pair is in solved form

[7] Our notation will differ from [8] in that we will denote constants of functional type by f and variables ranging over functions by F and G. Following Huet, lambda expressions will be represented by e, $@$ will denote either a constant or a variable, and $\tau(e)$ will denote the type of the term e. A term whose head $@$ is a free variable is termed *flexible*; if $@$ is a constant or a bound variable it is termed *rigid*.

if it is flexible-flexible or if it is of the form $\langle x, e\rangle$ for some variable x (functional or otherwise) not in $FreeVar(e)$.

The procedure based on these transformations must of course account for β-reduction, and trivial unifiers must be extracted from flexible-flexible pairs in solved forms; the details are omitted. The interesting issue here is that the pruning of the search space in the occur check case carries over to the higher-order case. It is possible to generate pairs isomorphic (up to renaming) to previous pairs by a series of transformations by (4) or by (5), since if applied to a flexible-rigid pair $\langle e_1, e_2\rangle$, where the head of e_1 occurs in e_2, the occur check situation will be distributed into the subterms in a manner similar to E-unification. An implementation which marks addresses of occur check variables to stop useless recursion caused by this phenomenon would be an improvement on Huet's original formulation, and appears to be new.

7 Conclusion

We have developed a general approach to E-unification based on transformations on systems and designed a complete E-unification procedure which implements the search for a system in solved form. It can be viewed as a top-down recursive procedure generalizing Robinson's recursive algorithm [17].

Much work has gone into the design of unification procedures for specialized theories [18], and into procedures based on narrowing [4], which depends on a completion procedure. Both [10,11] and [14] have developed procedures based on transformations on systems, but both of these depend on a completion procedure as a preprocessing phase. We have designed a complete procedure for the general case which improves the brute-force method of enumerating all rewrite sequences in the belief that such an approach might reveal fundamental insights into the problem of E-unification, and that current procedures might be able to be embedded in such a framework. For example, the *syntactic theories* described in [10,11], such as commutativity, are easily handled in our procedure by allowing at most one rewrite at each address; other special purpose algorithms are currently being studied. We have shown how higher-order unification can be implemented, and we are examining the extension of our procedure to higher-order unification using combinators.

We have also designed a version of the procedure taking advantage of confluence, but due to the lack of space, we cannot discuss it here. Our procedure "guesses" the confluence point and alternately tries to make progress from the left and the right; it reduces the search space considerably, since only oriented equations need to be tried (as opposed to all equations in $E \cup E^{-1}$ in the general case).

We are also examining a generalization of Paterson and Wegman's *unification closure* algorithm [15], which amounts to using structure sharing [1] to avoid explicitly applying substitutions and which maintains the solved systems in a triangular form; since backtracking may occur, a *trace stack* (as in Prolog) must be stored. (An implementation using this technique is being written.)

In addition to defining more closely the relationship of our method to previous procedures, we are currently examining a fundamentally new form of E-unification, termed Rigid E-unification, which has come up in the study of equational matings [6]. This method assumes that once equations are used, they are "frozen" with their substitutions, and can not be used again with another substitution (and are essentially ground thereafter). We conjecture that rigid unification is decidable, and are studying the application of the methods of this paper to its solution. Several decidable subcases have been isolated and are being studied using the implementation described in this paper.

Acknowledgment: We wish to thank Stan Raatz for much helpful discussion concerning both the general content of this paper and its presentation.

8 References

[1] Boyer, R.S., Moore, J.S., The Sharing of Structure in Theorem-proving Programs, *Machine Intellegence* 7: 101-116 (1972).

[2] Church, A., A Formulation of the Simple Theory of Types, *Journal of Symbolic Logic*, 5: 56-68 (1940).

[3] Fages, F. and Huet, G., Complete Sets of Unifiers and Matchers in Equational Theories, *TCS* 43(2,3), pp. 189-200, 1986.

[4] Fay, M., First-order Unification in an Equational Theory, *Proc. 4th Workshop on Automated Deduction*, Austin Texas, 1979.

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

[6] Gallier, J.H., Raatz, S., and Snyder, W., Theorem Proving using Rigid E-Unification: Equational Matings, submitted to LICS 1987.

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

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

[9] Huet, G. and Oppen, D. C., Equations and Rewrite Rules: A Survey, in: R. V. Book (ed.), *Formal Languages: Perspectives and Open Problems*, Academic Press, NY, 1982.

[10] Kirchner, C., Méthodes et Outils de Conception Systematique d'Algorithmes d'Unification dans les Theories Equationnelles, Thèse d'Etat, Université de Nancy I, 1985.

[11] Kirchner, C. Computing Unification Algorithms. *1st IEEE Symposium on Logic in Computer Science*, Cambridge, Massachusetts, 206-216, June 1986.

[12] Kozen, D. Positive First-Order Logic is NP-Complete. *IBM Journal of Research and Development*, 25(4), 327-332, 1981.

[13] Martelli, A., Montanari, U., An Efficient Unification Algorithm, *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 2 (April 1982), pp. 258-282.

[14] Martelli, A., Moiso, C., Rossi, G.F., An Algorithm for Unification in Equational Theories, *Third IEEE Symposium on Logic Programming*, Salt Lake City, Utah, pp. 180-186, September 1986.

[15] Paterson, M.S., Wegman, M.N., Linear Unification, *Journal of Computer and System Sciences* 16 (1978), pp. 158-167.

[16] Plotkin, G., Building in Equational Theories, in: *Machine Intelligence* 7: 73-90 (1972).

[17] Robinson, J.A., A Machine Oriented Logic Based on the Resolution Principle, *JACM* 12 (Jan. 1965), pp. 23 -41.

[18] Siekmann, J. H., Universal Unification, *Proc. CADE-7*, Napa 1984, 1-42.