Unification Theory

Franz Baader
German Research Center for AI (DFKI)
Postfach 2080
W-6750 Kaiserslautern, Germany
e-mail: baader@dfki.uni-kl.de

Abstract

The purpose of this paper is not to give an overview of the state of art in unification theory. It is intended to be a short introduction into the area of equational unification which should give the reader a feeling for what unification theory might be about. The basic notions such as complete and minimal complete sets of unifiers, and unification types of equational theories are introduced and illustrated by examples. Then we shall describe the original motivations for considering unification (in the empty theory) in resolution theorem proving and term rewriting. Starting with Robinson's first unification algorithm it will be sketched how more efficient unification algorithms can be derived.

We shall then explain the reasons which lead to the introduction of unification in non-empty theories into the above mentioned areas theorem proving and term rewriting. For theory unification it makes a difference whether single equations or systems of equations are considered. In addition, one has to be careful with regard to the signature over which the terms of the unification problems can be built. This leads to the distinction between elementary unification, unification with constants, and general unification (where arbitrary free function symbols may occur). Going from elementary unification to general unification is an instance of the so-called combination problem for equational theories which can be formulated as follows: Let E, F be equational theories over disjoint signatures. How can unification algorithms for E, F be combined to a unification algorithm for the theory $E \cup F$.

1 What is E-unification?

E-unification is concerned with solving term equations modulo an equational theory E. The theory is called "unitary" ("finitary") if the solutions of an equation can always be represented by one (finitely many) "most general" solutions. Otherwise the theory is of type "infinitary" or "zero." Equational theories which are of unification type unitary or finitary play an important rôle in automated theorem provers with "built in" theories (see e.g., [Pl72,Ne74,Sl74,St85]), in generalizations of the Knuth-Bendix algorithm (see e.g., [Hu80,PS81,JK86,Bc87]), and in logic programming with equality (see e.g., [JL84]).

The first two applications will be considered in subsequent sections. In the present section we shall introduce the basic notions of unification theory such as complete and minimal complete sets of unifiers and unification types of equational theories, and illustrate them by examples.

Let Ω be a signature, i.e., a set of function symbols with fixed arity, and let V be a countable set of variables. The set of Ω-terms with variables in V is denoted by $T(\Omega, V)$. A set of identities $E \subseteq T(\Omega, V) \times T(\Omega, V)$ defines an *equational theory* $=_E$, i.e., the equality of terms induced by E. The quotient algebra $T(\Omega, V)/{=_E}$ is the *E-free algebra* with generators V, i.e., the free algebra with countably many generators over the class of all models of E.

Example 1.1 Let Ω be the signature consisting of one binary function symbol f. The set of identities $A := \{f(x, f(y, z)) = f(f(x, y), z)\}$ defines the theory of semigroups. Obviously, the $=_A$-classes may be considered as words over the alphabet V, and the A-free algebra $T(\Omega, V)/{=_A}$ is isomorphic to the free semigroup V^+.

Informally, we can now say that *E-unification* is just solving equations in the E-free algebra $T(\Omega, V)/{=_E}$. To be more precise, we have to define what is meant by equation and by solution of the equation.

For this reason we consider *substitutions* which are mappings θ: $V \to T(\Omega, V)$ such that $\{x \in V \mid x\theta \neq x\}$ is finite. Since $T(\Omega, V)$ is the free Ω-algebra with generators V, this mapping can uniquely be extended to a homomorphism θ: $T(\Omega, V) \to T(\Omega, V)$. A *unification problem* (the equation) is a pair of terms s, t, and an *E-unifier* of the problem (the solution of the equation $s = t$ in $T(\Omega, V)/{=_E}$) is a substitution θ such that $s\theta =_E t\theta$. The set of all E-unifiers of s, t will be denoted by $U_E(s, t)$.

Example 1.2 Let Ω be the signature consisting of a binary function symbol f and a constant symbol a. We consider the terms $s = f(x, a)$ and $t = f(a, y)$.

$E = \emptyset$: In this case, the substitution $\theta = \{x \mapsto a, y \mapsto a\}$ is the only $\emptyset$-unifier of the terms s, t.

$E = C := \{f(x, y) = f(y, x)\}$: Obviously, θ is also a C-unifier of s, t. But since f is now commutative, there exists another C-unifier, namely $\sigma = \{x \mapsto y\}$. These two solutions of our equation $s = t$ are however not independent of each other. In fact, θ is an instance of σ because $\theta = \sigma \circ \{y \mapsto a\}$.

For most applications, one does not need the set of all E-unifiers. A complete set of E-unifiers, i.e., a set of E-unifiers from which all E-unifiers can be generated by *E-instantiation*, is usually sufficient. More precisely, we extend the relation $=_E$ to $U_E(s, t)$, and define the quasi-ordering $\leq_E$ on $U_E(s, t)$ by

$$\sigma =_E \theta \quad \text{iff} \quad x\sigma =_E x\theta \text{ for all variables } x \text{ occurring in } s \text{ or } t.$$
$$\sigma \leq_E \theta \quad \text{iff} \quad \text{there exists a substitution } \lambda \text{ such that } \theta =_E \sigma \circ \lambda.$$

If $\sigma \leq_E \theta$ then θ is called an *E-instance* of σ, and σ is said to be *more general* than θ.

A *complete set $cU_E(s,t)$ of E-unifiers of s,t* has to satisfy the conditions

- $cU_E(s,t) \subseteq U_E(s,t)$, and
- for all $\theta \in U_E(s,t)$ there exists $\sigma \in cU_E(s,t)$ such that $\sigma \leq_E \theta$.

For reasons of efficiency, such a set should be as small as possible. Thus one is interested in *minimal complete sets $\mu U_E(s,t)$ of E-unifiers of s,t*, that is, complete sets satisfying the additional condition

- For all $\sigma, \theta \in \mu U_E(s,t)$, $\sigma \leq_E \theta$ implies $\sigma = \theta$.

Example 1.3 As in Example 1.2 we consider the terms $s = f(x,a)$ and $t = f(a,y)$.

$E = A := \{f(x,f(y,z)) = f(f(x,y),z)\}$: The substitutions $\theta = \{x \mapsto a, y \mapsto a\}$ and $\tau = \{x \mapsto f(a,z), y \mapsto f(z,a)\}$ are A-unifiers of s,t, and it is easy to see that the set $\{\theta,\tau\}$ is complete. In addition, θ and τ are not comparable with respect to $\leq_A$, which shows that $\{\theta,\tau\}$ is a minimal complete set of E-unifiers of s,t.

A minimal complete set of E-unifiers may not always exist, but if it exists it is unique up to the equivalence defined by $\sigma \equiv_E \theta$ iff $\sigma \leq_E \theta$ and $\theta \leq_E \sigma$. For this reason, the *unification type* of an equational theory E can be defined with reference to the cardinality and existence of minimal complete sets.

Type 1 *(unitary):*	A set $\mu U_E(s,t)$ exists for all s,t and has cardinality ≤ 1.
Type ω *(finitary):*	A set $\mu U_E(s,t)$ exists for all s,t and is of finite cardinality.
Type ∞ *(infinitary):*	A set $\mu U_E(s,t)$ exists for all s,t, but may be infinite.
Type 0 *(zero):*	There are terms s,t such that a set $\mu U_E(s,t)$ does not exist.

For example, the empty theory $\emptyset$ is unitary (see [Ro65]), commutativity $C = \{f(x,y) = f(y,x)\}$ is finitary (see e.g., [Si76]), associativity $A = \{f(x,f(y,z)) = f(f(x,y),z)\}$ is infinitary (see [Pl72]), and the theory $B = A \cup \{f(x,x) = x\}$ of idempotent semigroups (bands) is of type zero (see [Ba86,Sc86]).

If a theory E is unitary, then a minimal complete set $\mu U_E(s,t)$ is either empty, if s,t are not unifiable, or it consists of a single E-unifier of s,t. This unifier is called *most general E-unifier* of s,t. It is unique up to $\equiv_E$-equivalence. For the empty theory, this means that most general unifiers are unique up to variable renaming, but in general the relation $\equiv_E$ may be more complex.

As already mentioned above, most applications of E-unification presuppose that the theory E is unitary or finitary. Of course, for these applications it is not enough to just know that a given theory is of type finitary. One also needs an *E-unification algorithm.*

Such an algorithm should be able to decide whether a given pair s, t of terms is unifiable; and if the answer is "yes" it should compute a finite complete set of E-unifiers of s, t. This notion of a "unification algorithm" should be distinguished from the notion "unification procedure" which is only required to enumerate a (possibly infinite) complete set of E-unifiers, without necessarily yielding a decision procedure for E-unifiability (see e.g., [Pl72] for an example of such a procedure for A-unification).

In order to get efficient applications, the complete set computed by the unification algorithm should be as small as possible; but for some theories, computing a minimal complete set as opposed to just computing a finite complete set may cause too much overhead compared to what is gained by having a smaller set. As an example of a theory for which this phenomenon occurs one can take commutativity $C = \{f(x,y) = f(y,x)\}$. It is very easy to devise an algorithm computing finite complete sets of C-unifiers, but it is much harder to get minimal complete sets (see e.g., [Si76,He87]).

2 Unification in the empty theory

The earliest references for unification of terms (which in the framework of the previous section is called $\emptyset$-unification) date back to E. Post in the 1920s and J. Herbrand in 1930 (see [Si89] for an account of the early history of unification theory). But its real importance became clear only when $\emptyset$-unification was independently rediscoverd in J.A. Robinson's paper on the resolution principle [Ro65] and in D. Knuth's paper on completion of term rewriting systems [KB70]. Both papers were seminal for their respective fields, namely automated theorem proving and term rewriting.

Robinson and Knuth show that two unifiable terms always have a most general $\emptyset$-unifier, i.e., that the empty theory is unitary, and they describe an algorithm which computes this most general $\emptyset$-unifier.

2.1 An informal description of Robinson's algorithm

We shall now explain Robinson's algorithm with the help of two examples. A formal description of a very similar algorithm can be found in the next section.

Example 2.1 Assume that we want to unify $s = f(x, g(a,z))$ and $t = f(g(a,y),x)$, where f, g are binary function symbols, a is a constant symbol, and x, y, z are variables.

In the first step, one reads the terms simultaneously from left to right until the first disagreement occurs. In our example, this disagreement occurs at the variable x in s and the function symbol g in t. These places of disagreement define the so-called disagreement terms, which are in our example x and $g(a,y)$. To unify s and t one has to unify these disagreement terms, and this can obviously be done with the help of the substitution $\sigma_1 := \{x \mapsto g(a,y)\}$.

Now one applies this substitution to s and t, and carries on with reading the obtained terms—which are $s\sigma_1 = f(g(a,y), g(a,z))$ and $t\sigma_1 = f(g(a,y), g(a,y))$ in our example—

from left to right until the first disagreement occurs. This process has to be iterated until the terms are unified. In the example, we get the terms z and y as the next pair of disagreement terms. After applying the substitution $\sigma_2 := \{y \mapsto z\}$ to $s\sigma_1$ and $t\sigma_1$, we have obtained the unified term $s\sigma_1\sigma_2 = f(g(a,z),g(a,z)) = t\sigma_1\sigma_2$. The composition $\sigma := \sigma_1 \circ \sigma_2$ is a most general $\emptyset$-unifier of s, t.

Obviously, we could also have used the substitution $\{z \mapsto y\}$ instead of $\sigma_2 = \{y \mapsto z\}$. This explains why most general $\emptyset$-unifiers are unique only up to variable renaming.

Until now we have only treated the case where the two terms are unifiable. The next example considers all the possible reasons for non-unifiability of terms.

Example 2.2 First, assume that we want to unify the terms $s = f(g(a,y),z)$ and $t = f(f(x,y),z)$. In this case, the disagreement occurs at the function symbol g in s and at the second symbol f in t. This means that the disagreement terms—namely $g(a,y)$ and $f(x,y)$—have different function symbols as top level symbol. Obviously, this means that the disagreement terms, and thus also the terms s,t, are not unifiable. This kind of reason for non-unifiability is called *clash failure.*

Second, assume that we want to unify the terms $s = f(g(a,x),z)$ and $t = f(x,z)$. Here we obtain disagreement terms $g(a,x)$ and x. These two terms cannot be unified because the variable x occurs in the term $g(a,x)$. In fact, for any substitution σ the size of the term $x\sigma$ is strictly smaller than the size of $g(a,x)\sigma = g(a,x\sigma)$. This kind of reason for non-unifiability is called *occur-check failure.*

2.2 Motivations for using $\emptyset$-unification

In the remainder of this section we shall shortly sketch the reason why unification is important for resolution-based theorem proving and completion of term rewriting systems.

The aim of resolution-based theorem proving is to refute a given set of clauses. In the propositional case, the *resolution principle* can be described roughly as follows. Suppose that one already has derived clauses $A \vee p$ and $B \vee \neg p$ where A, B are clauses and p is a propositional variable. Then one can also deduce $A \vee B$.

In the first order case, the rôle of propositional variables is played by atomic formulae. For example, assume that we have clauses of the form $A \vee P(x,a)$ and $B \vee \neg P(a,y)$. Before the resolution rule can be applied one has to instantiate the variables x, y in a suitable way. The appropriate instantiations can be found via unification (where the predicate symbols are treated like function symbols). In the example, we can apply the $\emptyset$-unifier $\theta := \{x \mapsto a, y \mapsto a\}$, which yields the clauses $A\theta \vee P(a,a)$ and $B\theta \vee \neg P(a,a)$. After applying the resolution rule we thus get $A\theta \vee B\theta$.

In the present example, there was only one $\emptyset$-unifier of the given pair of atomic formulae, but in general there may exist infinitely many unifiers. However, it can be shown that one can restrict oneself to most general unifiers without losing refutation completeness.

The aim of a *completion procedure for term rewriting systems* is to transform a given system into an equivalent complete (i.e., confluent and terminating) system, which then can be used to decide the word problem for the corresponding equational theory.

If a rewrite system is terminating, then confluence is equivalent to local confluence, and this property can be decided by considering finitely many critical pairs (see e.g., [KB70,Hu80]). For local confluence, one has to consider triples s, t_1, t_2 of terms where t_1 is obtained from s by applying some rule $g \rightarrow d$ of the system, and t_2 is obtained from s by applying some rule $l \rightarrow r$. The system is *locally confluent* iff for all such triples there exists a common descendant t of t_1 and t_2 (see the picture below). The picture also shows

$$\begin{tikzcd} & s \arrow[dl, "g \rightarrow d"'] \arrow[dr, "l \rightarrow r"] & \\ t_1 \arrow[dr, "*"'] & & t_2 \arrow[dl, "*"] \\ & t & \end{tikzcd}$$

Local confluence

$$\begin{tikzcd} & f(f(x,e),z) \arrow[dl, "{f(x',e) \rightarrow x'}"'] \arrow[dr, "{f(f(x,y),z) \rightarrow f(x,f(y,z))}"] & \\ f(x,z) & & f(x,f(e,z)) \end{tikzcd}$$

A critical pair

an example for such a triple where the rule for a right neutral element e applied to the subterm $f(x,e)$ of $s = f(f(x,e),z)$ yields $t_1 = f(x,z)$, whereas the associativity rule for f applied to s yields $t_2 = f(x, f(e,z))$. The term s of this example was generated from the two rules $f(x',e) \rightarrow x'$ and $f(f(x,y),z) \rightarrow f(x,f(y,z))$ as follows:[1] We have applied the unifier $\theta := \{x' \mapsto x, y \mapsto e\}$ of the left hand side $f(x',e)$ of the first rule and the subterm $f(x,y)$ of the other left hand side to this other left hand side. The *critical pair* t_1, t_2 was then obtained from s by applying the two rules at the appropriate positions. As for resolution it can be shown that it suffices to use most general $\emptyset$-unifiers in the computation of critical pairs.

3 Efficient algorithms for $\emptyset$-unification

The naive unification algorithm described in the previous section is of exponential time and space complexity. This is demonstrated by the following example.

Example 3.1 We consider the terms

$$\begin{aligned} s_n &= f(f(x_0,x_0), f(f(x_1,x_1), f(f(x_2,x_2), f(\ldots, f(x_{n-1},x_{n-1}))\ldots))) \text{ and} \\ t_n &= f(x_1, f(x_2, f(x_3, f(\ldots, x_n)\ldots))) \end{aligned}$$

where f is a binary function symbol and $x_0, \ldots, x_n$ are variables. The most general $\emptyset$-unifier of s_n, t_n computed by the naive algorithm is of the form

$$\sigma_n = \{x_1 \mapsto f(x_0,x_0),$$

[1]Please note that the variables in the two rules have been made disjoint.

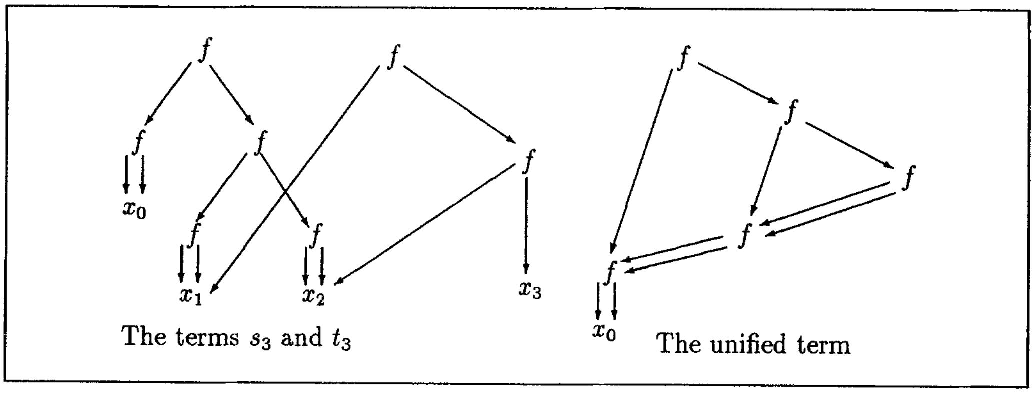
$$\begin{array}{rcl} x_2 & \mapsto & f(f(x_0,x_0),f(x_0,x_0)), \\ x_3 & \mapsto & f(f(f(x_0,x_0),f(x_0,x_0)),f(f(x_0,x_0),f(x_0,x_0))), \\ & \vdots & \\ & & \}. \end{array}$$

This means that $x_i\sigma_n$ contains the variable x_0 2^i times, and hence x_0 is contained in the unified term $\sum_{i=1}^{n} 2^i = 2^{n+1} - 1$ times. Since the size of s_n, t_n is linear in n, this shows that we need space—and thus also time—which is exponential in the size of the input terms.

Until now we have represented terms as strings of symbols. The example shows that more efficient unification algorithms depend on a better representation of terms. Robinson himself [Ro71] proposed a more succinct representation of terms by tables which improves the space complexity, but his algorithm is still exponential with respect to time complexity. Algorithms having almost linear time complexity were e.g. discovered by Huet [Hu76] and by Baxter [Bx76]; and finally Paterson and Wegman [PW78], and Martelli and Montanari [MM77] developed algorithms which are of linear time complexity. Later on, an algorithm which is of quadratic time complexity, but shows a better behaviour than the linear ones for most applications, was proposed by Bidoit and Corbin [BC83] (for a more complete survey of the history of efficient algorithms for $\emptyset$-unification see e.g. [Kn89,Si89]).

3.1 A recursive version of Robinson's algorithm working on dags

The algorithms of Paterson and Wegman and of Bidoit and Corbin use *directed acyclic graphs (dags)* for the representation of terms. This representation differs from the usual tree representation in that variables have to be shared and other subterms may be shared. The following picture shows the terms s_3, t_3 and the unified term $s_3\sigma_3 = t_3\sigma_3$ of Example 3.1 in dag-representation. This example shows that the unified term—which in string or



tree representation would have been exponential in the size of the input terms—can be represented by a dag which is not larger than the input terms.

Now we shall give a recursive version of the naive algorithm which works on dags. This algorithm will be linear with respect to space complexity, but still exponential with respect to time complexity. Then it will be shown how this algorithm can be modified, first to a quadratic algorithm, and then to an almost linear algorithm.

We assume that dags consist of nodes. Any node in a given dag defines a unique (sub)dag (consisting of the nodes which can be reached from this node), and thus a unique (sub)term. There are two different types of nodes, namely variable nodes and function nodes. Function nodes carry the following information: the name of the function symbol, the arity n of this symbol, and a list (of length n) of the nodes corresponding to the arguments of the function, the so-called successor list. Both function and variable nodes may be equipped with one additional pointer to another node.

The *input* of the unification procedure (see Figure 1) is a pair of nodes in a dag. The *output* is "true" or "false," depending on whether the corresponding terms are unifiable or not. As a *side effect* the procedure creates an additional pointer structure which allows us to read off the unified term and the most general $\emptyset$-unifier.

These additional pointers are manipulated or used in the following three auxiliary procedures:

find: This procedure gets a node of the dag as input, and follows the additional pointers until it reaches a node without such a pointer. This node is the output of *find*.

union: This procedure gets a pair u, v of nodes (which do not have additional pointers) as input, and it creates an additional pointer from u to v.

occur: This procedure gets a variable node u and another node v (both of which do not have additional pointers) as input, and it performs the *occur check*, i.e., it tests whether the variable is contained in the term corresponding to v. This test is performed on the virtual term expressed by the additional pointer structure, i.e., one first applies *find* to all nodes reached during this test.

The unification algorithm described in Figure 1 requires only linear space since it does not create new nodes, and it creates at most one additional pointer for each variable node. However, the time complexity is still exponential. To see this one can consider the behaviour of the procedure *unify1* for the input terms $f(s_n, f(t'_n, x_n))$ and $f(t_n, f(s'_n, y_n))$ where s_n, t_n are defined as in Example 3.1 and s'_n, t'_n are obtained from s_n, t_n by replacing the x_i's by y_i's. The procedure needs exponentially many calls of *unify1* to finally unify the node corresponding to x_n with the node corresponding to y_n. To be more precise, these nodes are already unified after n calls of *unify1* (when x_1 and y_1 are unified), but the procedure needs exponentially many additional calls of *unify1* to recognize this fact.

3.2 A quadratic algorithm

As a solution to this problem, Bidoit and Corbin propose to not only replace variable nodes during the unification process, but also function nodes, provided that one unifies

```
procedure unify1($k_1$,$k_2$)
if    $k_1 = k_2$ then return true
else  %$k_1$ and $k_2$ are physically different nodes
      if    function-node($k_2$)       %if one of the nodes is a
      then  $u := k_1; v := k_2$       %variable node then u
      else  $u := k_2; v := k_1$       %is now a variable node
      fi
      if    variable-node($u$)
      then  if    occur($u$,$v$)
            then  return false         %occur-check failure
            else  union($u$,$v$);      %replace variable u by the
                  return true          %term corresponding to v
            fi
      else  % u and v are function nodes
            if    function-symbol($u$) $\neq$ function-symbol($v$)
            then  return false         %clash failure
            else  $n$ := arity(function-symbol($u$));
                  $(u_1, ..., u_n)$ := successor-list($u$);
                  $(v_1, ..., v_n)$ := successor-list($v$);
                  $i := 0$; bool := true
                  while $i < n$ and bool do
                  $i := i + 1$; bool := unify1(find($u_i$),find($v_i$))
                  od
                  return bool
            fi
      fi
fi
end procedure unify1
```

Figure 1: A recursive version of Robinson's algorithm working on dags

the corresponding arguments. This can be achieved by a very simple modification of our procedure *unify1*. One simply has to insert the statement "union(u,v)" immediately in front of the while-loop. Thereby, one obtains a *procedure unify2* which is of quadratic time complexity. Since each call of *unify2* either returns "true" immediately (if the nodes were physically identical) or makes one more node virtually unreachable (i.e., it can no longer be the result of a find operation), there can only be linearly many recursive calls of *unify2*. This also shows that there are only linearly many calls of *find*, *union*, and *occur*.

The quadratic time complexity comes from the fact that the complexity of both *find* and *occur* is not constant, but may be linear. This should be obvious for *occur*. As an example for the linearity of *find*, consider the unification problem for the terms $s_1 := f(x_2, f(x_3, \ldots, f(x_n, y)\ldots))$ and $s_2 := f(x_1, f(x_1, \ldots, f(x_1, x_1)\ldots))$. Let k_1, k_2 be the nodes corresponding to s_1, s_2 in a dag-representation of this problem. During the execution of unify2(k_1,k_2), *find* is called n times with the node corresponding to x_1, and for $i = 1, \ldots n$, the i^{th} call has to follow a pointer chain of length $i - 1$.

3.3 An almost linear algorithm

Thus we have detected two sources of non-linearity of *unify2*, namely *occur* and *find*. The first source can easily be circumvented by just omitting the occur check during the execution of the unification procedure. Since occur-check failures are not detected immediately, the procedure may return "true" even if the terms are not unifiable. But in this case a cyclic structure has been generated, and this can be recognized by a linear search. The complexity of *find* can be reduced by employing a more efficient union-find algorithm as e.g. described in [Tr75]. In this way one gets an almost linear unification algorithm (see Figure 2) which is very similar to Huet's algorithm. To be more precise, the algorithm is of time complexity $O(n \cdot \alpha(n))$ where the function α is an extremely slow-growing function, which for practical purposes (i.e., for all terms representable at all on a computer) never exceeds the value 5.

The algorithm uses three additional auxiliary procedures, namely:

collapsing-find: Like *find*, this procedure gets a node k of the dag as input, and follows the additional pointers until the node find(k) is reached. In addition, *collapsing-find* relocates the pointer of all the nodes reached during this process to find(k).

union-with-weight: This procedure gets a pair u, v of nodes (which do not have additional pointers) as input. If the set $\{k \mid k$ is a node with find$(k) = u\}$ is larger than the set $\{k \mid k$ is a node with find$(k) = v\}$, then it creates an additional pointer from v to u, otherwise it creates an additional pointer from u to v.

not-cyclic: This procedure gets a node k as input, and it tests the graph which can be reached from k for cycles. The test is performed on the virtual graph expressed by the additional pointer structure, i.e., one first applies *collapsing-find* to all nodes reached during this test.

Please note that we cannot apply the weighted union procedure in the case where we have a variable node and a function node. In this case the pointer has to go from the variable to the function node. Otherwise we should lose important information such as the name of the function symbol and the argument list. However, it is easy to see that the use of this non-optimal *union* can increase the time complexity at most by a summand $O(m)$ where m is the number of different variable nodes occurring in the dag.

```
procedure unify3($k_1$,$k_2$)
if      cyclic-unify($k_1$,$k_2$) and
        not-cyclic($k_1$)
then    return true
else    return false
fi end procedure unify3

procedure cyclic-unify($k_1$,$k_2$)
if      $k_1 = k_2$ then return true
else    %$k_1$ and $k_2$ are physically different nodes
        if      function-node($k_2$)    %if one of the nodes is a
        then    $u := k_1; v := k_2$    %variable node then u
        else    $u := k_2; v := k_1$    %is now a variable node
        fi
        if      variable-node($u$)
        then    if      variable-node($v$)
                then    union-with-weight($u$,$v$)
                else    union($u$,$v$)                  %no weighted union
                fi
                return true                             %no occur-check
        else    % u and v are function nodes
                if      function-symbol($u$) $\neq$ function-symbol($v$)
                then    return false  %clash failure
                else    $n$ := arity(function-symbol($u$));
                        $(u_1, ..., u_n)$ := successor-list($u$);
                        $(v_1, ..., v_n)$ := successor-list($v$);
                        $i := 0$; bool := true;
                        union-with-weight($u$,$v$)
                        while $i < n$ and bool do
                        $i := i + 1$;
                        bool := cyclic-unify(collapsing-find($u_i$),collapsing-find($v_i$))
                        od
                        return bool
                fi
        fi
fi end procedure unify3
```

Figure 2: An almost linear unification algorithm

4 Unification in non-empty theories

In this section we shall first sketch by two examples why unification in equational theories was introduced into the fields automated theorem proving and term rewriting. Then we shall give some examples for new problems—i.e., problems not occurring for the empty theory—which arise in theory unification. These examples will show that one has to be very careful when trying to generalize definitions and results from $\emptyset$-unification to unification in non-empty theories.

4.1 Motivations for using E-unification

Plotkin [P172] observed that resolution theorem provers may waste a lot of time by applying axioms like associativity and commutativity. As a solution to this problem he proposed to build such equational axioms into the theorem proving mechanism. As a consequence one has to use unification modulo theses axioms in place of unification in the empty theory. Plotkin's paper was seminal for unification theory since, for example, the important notion of minimal complete set of unifiers (which Plotkin called a maximally general set of unifiers) was formally introduced for the first time.

Example 4.1 Assume that we have the axioms $f(f(x,y),z) = f(x,f(y,z))$ for associativity and $f(x,x) = x$ for idempotence, and that we should like to apply idempotency to the term

$$f(x_0, f(x_1, \ldots, f(x_{n-1}, f(x_n, f(x_0, \ldots, f(x_{n-1}, x_n) \ldots))) \ldots))$$

There are exponentially many ways of rearranging the parentheses with the help of associativity, and it takes a lot of time if the theorem prover has to search for the right one.

To solve this problem one can consider what a human mathematician would do in this case. (S)he would of course use words instead of terms, i.e., (s)he would work modulo associativity. In this framework one could at once apply idempotency $xx = x$ to the word

$$\underbrace{x_0 \cdots x_n}_{x} \underbrace{x_0 \cdots x_n}_{x}.$$

If we want to adopt this proceeding in a resolution theorem prover, then we have to replace $\emptyset$-unification in the resolution step by A-unification.

In term rewriting one comes very soon to the point where one would like to work modulo an equational theory. This is a consequence of the fact that certain identities cannot be oriented into terminating rewrite rules. As a solution to this problem one can leave some identities unoriented, and then pursue rewriting modulo these equational axioms. But then one must also use unification modulo these axioms when computing critical pairs (see e.g., [PS81,JK86] for details).

Example 4.2 In [KB70], the equational theory of groups was used as the motivating example. If one tries to apply the completion method of [KB70] to the theory of abelian groups, then one has to face the following problem: obviously, the commutativity axiom $f(x,y) = f(y,x)$ cannot be oriented into a terminating rule. A solution to this problem is to use rewriting modulo

$$AC := \{f(f(x,y),z) = f(x,f(y,z)), f(x,y) = f(y,x)\}.$$

Rewriting modulo $C := \{f(x,y) = f(y,x)\}$ is not possible because modulo C associativity cannot be oriented into a C-terminating rule. Modulo AC one can for example make the following rewrite step:

$$f(x,f(y,i(x))) \xrightarrow[f(x,i(x)) \to e]{\text{modulo } AC} f(e,y)$$

When computing critical pairs for the resulting system one has to use an AC-unification algorithm.

In the following subsections we shall consider some of the particular problems which arise when one goes from unification in the empty theory to unification in non-empty theories.

4.2 Single equations versus systems of equations

Until now we have considered E-unification problems for pairs s,t of terms, i.e., we have considered a single equation $s = t$ which has to be solved in the E-free algebra $T(\Omega,V)/{=_E}$. For many applications, one has to solve systems $\Gamma = \{s_1 = t_1, \ldots, s_n = t_n\}$ of equations, i.e., one wants to find a substitution σ satisfying $s_1\sigma =_E t_1\sigma$, ..., $s_n\sigma =_E t_n\sigma$. The substitution σ is then called E-unifier of the system Γ. Now the notions complete and minimal complete set of unifiers and unification type of a theory can also be defined with respect to solving systems of equations.

For the empty theory, solving systems of equations can trivially be reduced to solving single equations. In fact, it is easy to see that σ is an $\emptyset$-unifier of the system $\Gamma = \{s_1 = t_1, \ldots, s_n = t_n\}$ iff it is an $\emptyset$-unifier of the pair $f(s_1, f(s_2, \cdots f(s_{n-1}, s_n)\cdots))$, $f(t_1, f(t_2, \cdots f(t_{n-1}, t_n)\cdots))$ of terms.

More general, for unitary and finitary theories E, a finite E-unification algorithm for single equations can always be used to get a finite E-unification algorithm for finite systems of equations. This is an immediate consequence of the following fact: Let Γ be a finite unification problem, and let s,t be terms. If $cU_E(\Gamma)$ is a finite complete set of E-unifiers of Γ, and if for all substitutions $\sigma \in cU_E(\Gamma)$, the sets $cU_E(s\sigma, t\sigma)$ are finite complete sets of E-unifiers of $s\sigma, t\sigma$ then

$$\bigcup_{\sigma \in cU_E(\Gamma)} \sigma \circ cU_E(s\sigma, t\sigma) := \{\sigma \circ \tau \mid \sigma \in cU_E(\Gamma) \text{ and } \tau \in cU_E(s\sigma, t\sigma)\}$$

is a finite complete set of E-unifiers of $\Gamma \cup \{s = t\}$ (see e.g., [He87]).

There are also non-finitary equational theories where solving finite systems of equations can be reduced to solving single equations; but the reduction may often be more complex. As an example for this case one can take associativity (see e.g., [Pé81]). However, a reduction cannot be achieved for all equational theories. This is demonstrated by the following two results:

- Schmidt-Schauß has shown that there exists an equational theory E (see [BH89]) such that
  - E-unification for single equations is infinitary, but
  - E-unification for systems of equations is of type zero.
- Narendran and Otto have shown that there exists an equational theory F (see [NO90]) such that
  - testing for unifiability is decidable for single equations, but
  - it is undecidable for systems of equations.

4.3 A closer look at the signature

It is important to note that the signature over which the terms of the unification problems may be built has considerable influence on the unification type and on the existence of unification algorithms.

To make this remark more precise, we define the *signature of an equational theory E* (for short $sig(E)$) as the set of function symbols occurring in the identities of E. When talking about unification in the theory E, often one only thinks of E-unification problems where the terms to be unified are built over $sig(E)$, i.e., are elements of $T(sig(E), V)$. However, the applications of E-unification in theorem proving and term rewriting usually require that these terms may contain additional constant symbols, or even function symbols of arbitrary arity. Because the interpretation of these symbols is not constrained by the equation theory, they are called *free symbols.*

Example 4.3 The theory $A = \{f(f(x, y), z) = f(x, f(y, z))\}$ has signature $sig(A) = \{f\}$. When talking about A-unification, one first thinks of unifying modulo A terms built by using just the symbol f and variables, or equivalently, of unifying words over the alphabet V.

However, suppose that our resolution theorem prover—which has built in the theory A—gets the formula

$$\exists x : (\forall y : f(x, y) = y \ \land \ \forall y \exists z : f(z, y) = x)$$

as axiom. In a first step, this formula has to be Skolemized, i.e., the existential quantifiers have to be replaced by new function symbols. In our example, we need a nullary symbol

e and a unary symbol i in the Skolemized form

$$\forall y : f(e,y) = y \;\wedge\; \forall y : f(i(y),y) = e$$

of the axiom. This shows that, even if we start with formulae containing only terms built over f, our theorem prover has to handle terms containing additional free symbols.

The same situation occurs in term rewriting modulo equation theories. In Example 4.2 we have proposed to use rewriting modulo AC for the theory of abelian groups. Obviously, the remaining rules (for the neutral element and the inverse) also contain symbols not contained in $sig(AC) = \{f\}$.

To sum up, one should distinguish between three types of E-unification, namely

Elementary E-unification: the terms of the problem may contain only symbols of $sig(E)$;

E-unification with constants: the terms of the problem may contain additional free constant symbols;

General E-unification: the terms of the problem may contain additional free function symbols of arbitrary arity.

For the empty theory, we have of course considered general $\emptyset$-unification because elementary unification and unification with constants are trivial in this case. The following facts show that there really is a difference between the three types of E-unification.

- There exist theories which are unitary with respect to elementary unification, but finitary with respect to unification with constants. An example for such a theory is the theory of abelian monoids, i.e., $AC \cup \{f(x,1) = x\}$ (see e.g., [He87,Ba89]), or the theory of idempotent abelian monoids, i.e., $AC \cup \{f(x,1) = x, f(x,x) = x\}$ (see e.g., [BB86]).
- There exists an equational theory for which elementary unification is decidable, but unification with constants is undecidable (see [Bü86]).
- From the developement of the first algorithm for AC-unification with constants [St75,LS75] it took almost a decade until the termination of an algorithm for general AC-unification was shown by Fages [Fa84].

4.4 The combination problem for unification algorithms

Motivated by the previous section, one can now ask: How can algorithms for elementary unification (or for unification with constants) be used to get algorithms for general unification? This leads to the more general question of how to combine unification algorithms for equational theories with disjoint signatures.

More formally, this *combination problem* can be described as follows. Assume that two finitary equational theories E, F with $sig(E) \cap sig(F) = \emptyset$ are given. How can unification algorithms for E and F be combined to a unification algorithm for $E \cup F$? Recall that by a unification algorithm we mean an algorithm which computes a finite complete set of unifiers.

This combination problem was first considered in [St75,St81,Fa84,HS87] for the case where several AC-symbols and free symbols may occur in the terms to unified. More general combination problems were for example treated in [Ki85,Ti86,He86,Ye87], but the theories considered in these papers always had to satisfy certain restrictions (such as collapse-freeness or regularity[2]) on the syntactic form of their defining identities.

The problem was finally solved in its most general form by Schmidt-Schauß [Sc89]. His combination algorithm imposes no restriction on the syntactic form of the defining identities. The only requirements are:

- There exist unification algorithms for unification with constants for E and F.
- All constant elimination problems must be finitary solvable in E and F.

A *constant elimination problem* in a theory E is a finite set $\{(c_1, t_1), \ldots, (c_n, t_n)\}$ where the c_i's are free constants (i.e., constant symbols not occurring in $sig(E)$) and the t_i's are terms (built over $sig(E)$, variables, and some free constants). A solution to such a problem is called a *constant eliminator.* It is a substitution σ such that for all $i, 1 \leq i \leq n$, there exists a term t_i' not containing the free constant c_i with $t_i' =_E t_i\sigma$. The notion *complete set of constant eliminators* is defined analogously to the notion complete set of unifiers. The requirement that all constant elimination problems must be finitary solvable in E means that one can always compute a finite complete set of constant eliminators for E.

A more efficient version of this combination method has been described in [Bo90]. It should be noted that the method of Schmidt-Schauß can also handle theories which are not finitary. In this case, procedures which enumerate complete sets of unifiers for E and F can be combined to a procedure enumerating a complete set of unifiers for $E \cup F$.

5 Some topics in unification theory

The purpose of this paper was not to give an overview of the state of art in unification theory, and for this reason there are many important topics we have not touched. Now we shall give a (certainly not complete) list of some of the research problems of current interest in unification theory.

- Determine unification types of equational theories (see e.g., [Sc86,Ba87,Fr89]).

[2]A theory E is called collapse-free if it does not contain an identity of the form $x = t$ where x is a variable and t is a non-variable term, and it is called regular if the left and right hand sides of the identities contain the same variables.

- Investigate the decidability of the unification problem, and the complexity of this decision problem (see e.g., [SS86,KN89]; in [KN89] there is a table summarizing many of the known results in this direction).
- Devise unification algorithms for specific unitary and finitary theories. For example, a lot of work was—and still is—devoted to finding efficient algorithms for *AC*-unification (see e.g., [St81,Fa84,Ki85,Bt86,He87,HS87,Fo85,CF89,BD90]).
- Devise universal unification algorithms, i.e., algorithms where the equational theory also belongs to the input of the algorithm. Examples are methods based on narrowing (see e.g., [Fy79,Hl80,Fi84,NR89]), or on Martelli and Montanari's decomposition technique (see e.g., [GS87,KK90]).

For more information on these and other topics in unification theory one can consult Siekmann's overview of the state of art in unification theory [Si89], or Jouannaud and Kirchner's survey of unification [JK90]. In particular, these papers contain an almost complete list of references on unification theory.

References

[Ba86] F. Baader, "The Theory of Idempotent Semigroups is of Unification Type Zero," *J. Automated Reasoning* **2**, 1986.

[Ba87] F. Baader, "Unification in Varieties of Idempotent Semigroups," *Semigroup Forum* **36**, 1987.

[Ba89] F. Baader, "Unification in Commutative Theories," in C. Kirchner (ed.), *Special Issue on Unification, J. Symbolic Computation* **8**, 1989.

[BB86] F. Baader, W. Büttner, "Unification in Commutative Idempotent Monoids," *Theoretical Computer Science* **56**, 1986.

[Bc87] L. Bachmair, *Proof Methods for Equational Theories*, Ph.D. Thesis, Dep. of Comp. Sci., University of Illinois at Urbana-Champaign, 1987.

[Bx76] L. Baxter, *The Complexity of Unification*, Ph.D. Thesis, University of Waterloo, Waterloo, Ontario, Canada, 1976.

[BC83] M. Bidoit, J. Corbin, "A Rehabilitation of Robinson's Unification Algorithm," In R.E.A. Pavon, editor, *Information Processing 83*, North Holland, 1983.

[Bo90] A. Boudet, "Unification in a Combination of Equational Theories : An Efficient Algorithm," *Proceedings of the 10th Conference on Automated Deduction, LNCS* **449**, 1990.

[BD90] A. Boudet, E. Contejean, H. Devie, "A New AC-unification Algorithm with a New Algorithm for Solving Diophantine Equations," *Proceedings of the 5th IEEE Symposium on Logic in Computer Science, Philadelphia*, 1990.

[Bü86] H.-J. Bürckert, "Some Relationships Between Unification, Restricted Unification, and Matching," *Proceedings of the 8th Conference on Automated Deduction, LNCS* **230**, 1986.

[BH89] H.-J. Bürckert, A. Herold, M. Schmidt-Schauß, "On Equational Theories, Unification, and Decidability," in C. Kirchner (ed.), *Special Issue on Unification, J. Symbolic Computation* **8**, 1989.

[Bt86] W. Büttner, "Unification in the Data Structure Multiset," *J. Automated Reasoning* **2**, 1986.

[CF89] M. Clausen, A. Fortenbacher, "Efficient Solution of Linear Diophantine Equations," in C. Kirchner (ed.), *Special Issue on Unification, J. Symbolic Computation* **8**, 1989.

[Fa84] F. Fages, "Associative-Commutative Unification," *Proceedings of the 7th Conference on Automated Deduction, LNCS* **170**, 1984.

[Fy79] M. Fay, "First Order Unification in an Equational Theory," *Proceedings of the 4th Workshop on Automated Deduction*, Austin, Texas, 1979.

[Fo85] A. Fortenbacher, "An Algebraic Approach to Unification under Associativity and Commutativity," *Proceedings of the 1st Conference on Rewriting Techniques and Applications*, Dijon, France, *LNCS* **202**, 1985.

[Fr89] M. Franzen, "Hilbert's Tenth Problem Has Unification Type Zero," Preprint, 1989. To appear in *J. Automated Reasoning*.

[Fi84] L. Fribourg, "A Narrowing Procedure with Constructors," *Proceedings of the 7th Conference on Automated Deduction, LNCS* **170**, 1984.

[GS87] J.H. Gallier, W. Snyder, "A General Complete E-Unification Procedure," *Proceedings of the Second Conference on Rewriting Techniques and Applications*, Bordeaux, France, *LNCS* **256**, 1987.

[He86] A. Herold, "Combination of Unification Algorithms," *Proceedings of the 8th Conference on Automated Deduction, LNCS* **230**, 1986.

[He87] A. Herold, *Combination of Unification Algorithms in Equational Theories*, Dissertation, Fachbereich Informatik, Universität Kaiserslautern , 1987.

[HS87] A. Herold, J.H. Siekmann, "Unification in Abelian Semigroups," *J. Automated Reasoning* **3**, 1987.

[Hu76] G.P. Huet, *Résolution d'équations dans des langages d'ordre* $1, 2, ..., \omega$, Thèse d'État, Université de Paris VII, 1976.

[Hu80] G.P. Huet, "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems," *J. ACM* **27**, 1980.

[Hl80] F.M. Hullot, "Canonical Forms and Unification," *Proceedings of the 5th Conference on Automated Deduction, LNCS* **87**, 1980.

[JL84] J. Jaffar, J.L. Lassez, M. Maher, "A Theory of Complete Logic Programs with Equality," *J. Logic Programming* **1**, 1984.

[JK86] J.P. Jouannaud, H. Kirchner, "Completion of a Set of Rules Modulo a Set of Equations," *SIAM J. Computing* **15**, 1986.

[JK90] J.P. Jouannaud, C. Kirchner, "Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification," Preprint, 1990. To appear in the Festschrift to Alan Robinson's birthday.

[KN89] D. Kapur, P. Narendran, "Complexity of Unification Problems with Associative-Commutative Operators," Preprint, 1989. To appear in *J. Automated Reasoning*.

[Ki85] C. Kirchner, *Méthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans les Théories equationnelles*, Thèse d'Etat, Univ. Nancy, France, 1985.

[KK90] C. Kirchner, F. Klay, "Syntactic Theories and Unification," *Proceedings of the 5th IEEE Symposium on Logic in Computer Science, Philadelphia*, 1990.

[Kn89] K. Knight, "Unification: A Multidisciplinary Survey," *ACM Computing Surveys* **21**, 1989.

[KB70] D.E. Knuth, P.B. Bendix, "Simple Word Problems in Universal Algebras," In J. Leech, editor, *Computational Problems in Abstract Algebra*, Pergamon Press, Oxford, 1970.

[LS75] M. Livesey, J.H. Siekmann, "Unification of AC-Terms (bags) and ACI-Terms (sets)," Internal Report, University of Essex, 1975, and Technical Report 3-76, Universität Karlsruhe, 1976.

[MM77] A. Martelli, U. Montanari, "Theorem Proving with Structure Sharing and Efficient Unification," *Proceedings of International Joint Conference on Artificial Intelligence*, 1977.

[NO90] P. Narendran, F. Otto, "Some Results on Equational Unification," *Proceedings of the 10th Conference on Automated Deduction, LNCS* **449**, 1990.

[Ne74] A.J. Nevins, "A Human Oriented Logic for Automated Theorem Proving," *J. ACM* **21**, 1974.

[NR89] W. Nutt, P. Réty, G. Smolka, "Basic Narrowing Revisited," *J. Symbolic Computation* **7**, 1989.

[PW78] M.S. Paterson, M.N. Wegman, "Linear Unification," *J. Comput. Syst. Sci.* **16**, 1978.

[Pé81] J.P. Pécuchet, *Équation avec constantes et algorithme de Makanin*, Thèse de Doctorat, Laboratoire d'informatique, Rouen, 1981.

[PS81] G. Peterson, M. Stickel, "Complete Sets of Reductions for Some Equational Theories," *J. ACM* **28**, 1981.

[Pl72] G. Plotkin, "Building in Equational Theories," *Machine Intelligence* **7**, 1972.

[Ro65] J.A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *J. ACM* **12**, 1965.

[Ro71] J.A. Robinson, "The Unification Computation," *Machine Intelligence* **6**, 1971.

[Sc86] M. Schmidt-Schauß, "Unification under Associativity and Idempotence is of Type Nullary," *J. Automated Reasoning* **2**, 1986.

[Sc89] M. Schmidt-Schauß, "Combination of Unification Algorithms," *J. Symbolic Computation* **8**, 1989.

[Si76] J.H. Siekmann, "Unification of Commutative Terms," SEKI-Report, Universität Karlsruhe 1976.

[Si89] J.H. Siekmann, "Unification Theory: A Survey," in C. Kirchner (ed.), *Special Issue on Unification, Journal of Symbolic Computation* **7**, 1989.

[SS86] J.H. Siekmann, P. Szabo, "The Undecidability of the D_A-Unification Problem," SEKI-Report SR-86-19, Universität Kaiserslautern, 1986, and *J. Symbolic Logic* **54**, 1989.

[Sl74] J.R. Slagle, "Automated Theorem Proving for Theories with Simplifiers, Commutativity and Associativity," *J. ACM* **21**, 1974.

[St75] M. Stickel, "A Complete Unification Algorithm for Associative-Commutative Functions," *Proceedings of the International Joint Conference on Artificial Intelligence*, 1975.

[St81] M.E. Stickel, "A Unification Algorithm for Associative-Commutative Functions," *J. ACM* **28**, 1981.

[St85] M.E. Stickel, "Automated Deduction by Theory Resolution," *J. Automated Reasoning* **1**, 1985.

[Ti86] E. Tiden, "Unification in Combinations of Collapse Free Theories with Disjoint Sets of Function Symbols," *Proceedings of the 8th Conference on Automated Deduction, LNCS* **230**, 1986.

[Tr75] E.T. Trajan, "Efficiency of a Good But Not Linear Set Union Algorithm," *J. ACM* **22**, 1975.

[Ye87] K. Yelick, "Unification in Combinations of Collapse Free Regular Theories," *J. Symbolic Computation* **3**, 1987.