A COMPLETE UNIFICATION ALGORITHM FOR ASSOCIATIVE-COMMUTATIVE FUNCTIONS[1,2]

by

Mark E. Stickel

Department of Computer Science

Carnegie-Mellon University

Pittsburgh, Pennsylvania

Abstract

An important component of mechanical theorem proving systems are unification algorithms which find most general substitutions which, when applied to two expresssions, make them equivalent. Functions which are associative and commutative (such as the arithmetic addition and multiplication functions) are often the subject of mechanical theorem proving. An algorithm which unifies terms whose function is associative and commutative is presented here. The algorithm eliminates the need for axiomatizing the associativity and commutativity properties and returns a complete set of unifiers without recourse to the indefinite generation of variants and instances of the terms being unified required by previous solutions to the problem.

[1] Research supported by the National Science Foundation (GJ-28457X2) and by the Advanced Research Projects Agency of the Secretary of the Office of Defense (F44620-73-C-0074).

[2] To be presented at the Fourth International Joint Conference on Artificial Intelligence (Tbilisi, Georgia, U.S.S.R.; Sept. 1975).

Introduction

At the core of many theorem proving systems is a unification algorithm which returns for a pair of input expressions a set of unifying substitutions, assignments to the variables of the expressions which make the two expressions equivalent. Typical is the unification algorithm of Robinson [6] for unifying atomic formulas of the first order predicate calculus in resolution theorem proving [1].

This work treats the case of unifying terms of the first order predicate calculus where the function is associative and commutative. Such functions are mathematically important and thus of interest to developers of theorem proving programs. Examples of such functions are the arithmetic addition and multiplication functions.

The case where the function is simply commutative is easily handled by a trivial extension to Robinson's unification algorithm which unifies the arguments of one term against permutations of the arguments of the other term.

The case where the function is simply associative is quite difficult and we know of no general solution. Suggestive of the difficulty of this problem is the fact that there may be an infinite number of unifiers for a pair of terms. For example, the terms f(xa) and f(ax) where f is associative, a is a constant, and x is a variable has unifiers with x=a, x=f(aa), x=f(aaa), ... (We represent the argument lists of associative functions with no extra parentheses, i.e., f(abc) rather than f(af(bc)) or f(f(ab)c).)

Two principal approaches to handling associativity or commutativity are available. The first, standard approach is to represent the terms conventionally, i.e., f(af(bc)) or f(f(ab)c) rather than f(abc), and axiomatize the associativity or commutativity property. The associativity axiom would be f(xf(yz))=f(f(xy)z) and the commutativity axiom would be f(xy)=f(yx). These axioms could be applied using some equality inference rules such as paramodulation [5].

The second approach represents associative functions as functions with an arbitrary number of arguments, i.e., uses f(abc) rather than f(af(bc)) or f(f(ab)c). Special purpose unification algorithms are provided for terms whose functions are associative, commutative, or both. Examples of this approach in first order predicate calculus theorem proving are the work of Nevins [2] and Slagle [8]. The algorithms for associativity, and for associativity and commutativity are incomplete, i.e., they fail to return all the unifiers in some cases. An example of this approach in the area of programming languages for problem solving is the use of the associative data type tuple or vector and associative and commutative data type bag in the QA4 and QLISP languages [7,4]. Again, in this case the algorithms for pattern matching (unifying) these expressions are incomplete. In both these cases, the incomplete algorithms can be augmented by a process which alters the input expressions to cause the unification algorithm applied to the altered expressions to return additional unifiers. The addition of this process (Slagle's widening operation for the first order predicate calculus [8] and Stickel's variable splitting operation for expressions of QA4 and QLISP [9]) results in completeness. Widening and variable splitting are both operations that must be performed on one or both input expressions an arbitrary number of times, replacing single variables of the expressions uniformly by two variables; it is essentially (repeated) paramodulation by the functionally reflexive axiom.

An example of the latter approach is the unification of f(abz) and f(xy) where f is associative and commutative. The special purpose unification algorithm would return the unifiers $\{x \leftarrow a, y \leftarrow f(bz)\}$, $\{x \leftarrow b, y \leftarrow f(az)\}$, $\{x \leftarrow z, y \leftarrow f(ab)\}$, $\{x \leftarrow f(bz), y \leftarrow a\}$, $\{x \leftarrow f(az), y \leftarrow b\}$, and $\{x \leftarrow f(ab), y \leftarrow z\}$. But this is an incomplete set of unifiers since the possibility that the value of z is not wholly contained in either the value of x or the value of y is not represented. After performing a widening operation on f(abz) resulting in $f(abz_1z_2)$ by instantiating z by $f(z_1z_2)$, additional new unifiers such as $\{x \leftarrow f(az_1), y \leftarrow f(bz_2), z \leftarrow f(z_1z_2)\}$ and $\{x \leftarrow f(abz_1), y \leftarrow z_2, z \leftarrow f(z_1z_2)\}$ are returned by the unification algorithm.

Related to this approach, though different in detail, is Plotkin's work on the theory of building in equational theories [3] of which associativity and commutativity are examples. In the case of associativity, Plotkin retains terms in a normal form: right associative form, although it could equivalently have been our unparenthesized form. His equivalent of the widening rule, the replacement of a variable by two new variables, is applied continually inside the unification algorithm rather than being used outside it. Thus his unification algorithm may generate an infinite number of unifiers as opposed to a unification algorithm guaranteed to produce a finite number of unifiers and a potentially infinite process (widening) for altering inputs to the unification algorithm to obtain additional unifiers. The difference in approaches seems to be principally one of organization of the search process.

In this paper, we present a new special purpose unification algorithm which we call the AC unification algorithm for terms whose functions are associative and commutative which returns a complete set of unifiers. This algorithm eliminates the need for axiomatizing associativity and commutativity and also eliminates the cost of continually applying these axioms which often results in much unnecessary or redundant computation. It also eliminates the need for using the process of widening or variable splitting whose necessity (for discovering a complete set of unifiers in the case of unifying any particular pair of expressions) is difficult to ascertain.

Terminology

Definition. A term is defined to be

(1) a constant,

(2) a variable, or

(3) a function symbol succeeded by a list of terms (the arguments of the function).

We shall use the symbols a, b, and c to represent constants, x, y, and z (possibly indexed) to represent variables, and f to represent a function which is associative and commutative.

Definition. A substitution component is an ordered pair of a variable v and a term t written as $v \leftarrow t$. A substitution component denotes the assignment of the term to the variable or the replacement of the variable by the term.

Definition. A substitution is a set of substitution components with distinct first elements, i.e., distinct variables being substituted for. Applying a substitution to an expression results in the replacement of those variables of the expression included among the first elements of the substitution components by the corresponding terms. The substitution components are applied to the expression in parallel and no variable occurrence in the second element of a substitution component will be replaced even if the variable occurs as the first element in another substitution component. Substitutions will be represented by the symbols σ and θ. The application of substitution θ to expression A is denoted by $A\theta$. The composition of substitutions $\theta\sigma$ denotes the substitution whose effect is the same as first applying substitution θ, then applying substitution σ, i.e., $A(\theta\sigma) = (A\theta)\sigma$ for every expression A.

Definition. A unifying substitution or unifier of two expressions is a substitution which when applied to the two expressions results in equivalent expressions. In ordinary unification, two expressions are equivalent if and only if they are identical. In unification of argument lists of commutative functions, two expressions are equivalent if they have the same function symbol and the same arguments in the same or different order.

Definition. Term s is an instance of term t, and t is a generalization of s, if there exists a substitution θ such that $t\theta=s$.

Similarly, substitution θ is an instance (generalization) of substitution σ if, for every term t, $t\theta$ is an instance (generalization) of $t\sigma$.

The AC Unification Algorithm

We present here an algorithm for unifying two terms whose function is associative and commutative. Terms will be represented as if the function had an arbitrary number of arguments with no superfluous parentheses.

We will assume that the argument lists of the two terms being unified have no common arguments. This presents no difficulty since no unifiers are lost and efficiency is gained if common arguments are eliminated immediately. This is done by removing common arguments a pair at a time, one from each of the argument lists. For example, before unifying f(xxyabc) and f(bbbcz), the b's common to the two terms are removed yielding f(xxyac) and f(bbcz), and the c's common to the two new terms are removed yielding f(xxya) and f(bbz). An example of the utility of immediately removing common arguments is the unification of f(g(x)y) and f(g(x)g(a)). If the g(x)'s common to the two terms are immediately removed, the unification algorithm will return the most general unifier $\{y \leftarrow g(a)\}$. If the common g(x)'s are retained, unification will likely result in the generation of the additional less general unifier $\{x \leftarrow a, y \leftarrow g(a)\}$.

The algorithm will be expressed partially in terms of an algorithm for the complete unification of terms with an associative and commutative function with only variables as arguments. The result of unifying such terms is an assignment to each variable of the terms some sequence of terms. Each variable is assigned a term t_i (whose function symbol is not f) or a term $f(t_1{}^{n_1} \ldots t_m{}^{n_m})$ (with n_i occurrences of term t_i as arguments of f). For such an assignment to be a unifier, the only requirement is that for each term t_i used in any assignment there are the same number of occurrences of that term occurring as arguments of f in each of the unified terms instantiated by the assignment. For example, in unifying $f(x_1 x_1 x_2 x_3)$ and $f(y_1 y_1 y_2)$, if term t is part of some assignment to one of the variables, then 2 times the number of occurrences of t in the assignment for x_1 plus the number of occurrences of t in the assignment for x_2 plus the number of occurrences of t in the assignment for x_3 must equal 2 times the number of occurrences of t in the assignment for y_1 plus the number of occurrences of t in the assignment for y_2. For example, $\{x_1 \leftarrow f(bb),$

$x_2 \leftarrow f(ab)$, $x_3 \leftarrow a$, $y_1 \leftarrow b$, $y_2 \leftarrow f(aabbb)\}$ is a unifier of $f(x_1x_1x_2x_3)$ and $f(y_1y_1y_2)$ since there are 2 a's and 5 b's in the instantiations of $f(x_1x_1x_2x_3)$ and $f(y_1y_1y_2)$ with the unified term being $f(aabbbbb)$.

With each pair of terms with an associative and commutative function with only variable arguments is associated a single equation representing the number and multiplicity of variables in each term. For example, the equation $2x_1+x_2+x_3 = 2y_1+y_2$ is associated with the pair of terms given above. This equation succintly represents the condition for a substitution to be a unifier: that the sum of the number of occurrences of any term in the value of each variable multiplied by the multiplicity of the variable in the term must be equal for the two terms.

Non-negative integral solutions to such equations can be used to represent unifiers. The solutions must be non-negative integral since each variable must be assigned a non-negative integral number of occurrences of each term.

In order to generate all the solutions to the problem of unifying the two terms, it is necessary to be able to represent all the solutions to the equation derived from the terms. Every non-negative integral solution to the equation is representable as a sum of elements of a particular finite set of non-negative integral solutions to the equation, i.e., every non-negative integral solution to the equation is a sum (equivalently, a sum with non-negative integral weights) of elements of a particular finite set of non-negative integral solutions. The finite set of non-negative integral solutions by whose addition the entire non-negative integral solution space is spanned is generable by generating in ascending order of value solutions to the equation, eliminating those solutions composable from those previously generated. This process can be made finite by placing a bound on the maximum solution value which will be used; such a maximum is proved in a later lemma to eliminate no needed solutions.

Consider the equation $2x_1+x_2+x_3 = 2y_1+y_2$. Solutions to the equation are:

	x_1	x_2	x_3	y_1	y_2	$2x_1+x_2+x_3$	$2y_1+y_2$	
1	0	0	1	0	1	1	1	z_1
2	0	1	0	0	1	1	1	z_2
3	0	0	2	1	0	2	2	z_3
4	0	1	1	1	0	2	2	z_4
5	0	2	0	1	0	2	2	z_5
6	1	0	0	0	2	2	2	z_6
7	1	0	0	1	0	2	2	z_7

Associated with each solution above is a new variable (in the rightmost column). The assignment of as many occurrences of that variable as specified in the solution to each of the variables of the original term results in a partial solution to the unification of the the original terms. In particular, the assignment of 2 occurrences of variable z_3 to x_3 and 1 occurrence to y_1 results in an equal number of occurrences of variable z_3 in each of $f(x_1x_1x_2x_3)$ and $f(y_1y_2)$.

Every non-negative integral solution to the equation is a (non-negative integer weighted) sum of the 7 solutions presented above, i.e., every solution is representable as $x_1=z_6+z_7$, $x_2=z_2+z_4+2z_5$, $x_3=z_1+2z_3+z_4$, $y_1=z_3+z_4+z_5+z_7$, $y_2=z_1+z_2+2z_6$ for some non-negative integral values of $z_1,\dots,z_6$. However, not every solution to the equation is a solution to the unification problem for which the equation was derived. There is an additional constraint that each variable of the original terms must be have at least one term in its value; it cannot have zero terms in its value.

Hence, we must form that subset of the $2^7=128$ sums for which each element of the 5-tuple is non-zero. (It is not necessary to consider sums in which any solution has a coefficient other than 0 or 1 since such solutions (in the unification problem) are already representable since the solution's inclusion with coefficient 1 introduces a variable which can have as its value an arbitrary number of terms as arguments of f thus simulating the case of the coefficient being greater than 1.) There are 69 such sums including for example (representing the sum by the set of its indices) {2,3,6}, {1,2,3,6}, and {4,6} with associated unifiers

$\{x_1 \leftarrow z_6, x_2 \leftarrow z_2, x_3 \leftarrow f(z_3 z_3), y_1 \leftarrow z_3, y_2 \leftarrow f(z_2 z_6 z_6)\}$,

$\{x_1 \leftarrow z_6, x_2 \leftarrow z_2, x_3 \leftarrow f(z_1 z_3 z_3), y_1 \leftarrow z_3, y_2 \leftarrow f(z_1 z_2 z_6 z_6)\}$, and

$\{x_1 \leftarrow z_6, x_2 \leftarrow z_4, x_3 \leftarrow z_4, y_1 \leftarrow z_4, y_2 \leftarrow f(z_6 z_6)\}$.

Note that if a variable could have as its value zero terms rather than one or more terms as in the first order predicate calculus, it would be unnecessary to form this subset of 2^n (where n is the number of solutions) sums. Only the sum of all the solutions would be required since any variable present in this sum could have value zero, and the variables in the corresponding unifier could be matched against zero terms. This is the situation with fragment variables in the bag data type in QA4 and QLISP [7,4] (see [9]).

To be more precise in the definition of the algorithm, the algorithm consists of the following steps:

1. Form an equation from the two terms where the coefficient of each variable in the equation is equal to the multiplicity of the corresponding variable in the term.

2. Generate all non-negative integral solutions to the equation eliminating all those solutions composable from other solutions.

3. Associate with each solution a new variable.

4. For each sum of the solutions (no solution occurring in the sum more than once) with no zero components assemble a unifier composed of assignments to the original variables with as many of each new variable as specified by the solution element in the sum associated with the new variable and the original variable.

Now we present the complete algorithm for unifying general terms with associative and commutative functions using the algorithm for the variable only case above. We are here concerned with terms whose function is associative and commutative with arbitrary arguments, i.e., arguments that may contain ordinary (non-associative, non-commutative) functions or f or other functions which are associative and commutative. We assume the presence of ordinary unification to deal with those aspects of the unification problem not dealt with explicitly here.

First, when unifying two terms, two new terms with only variable arguments are formed by uniformly replacing distinct arguments by new variables. These new terms have only variable arguments and are generalizations of the original two terms. For example, in unifying f(xxya) and f(bbz), we form generalizations $f(x_1x_1x_2x_3)$ and $f(y_1y_1y_2)$ with substitution $\{x_1 \leftarrow x, x_2 \leftarrow y, x_3 \leftarrow a, y_1 \leftarrow b, y_2 \leftarrow z\}$ instantiating the new terms to the original terms.

Next, using the previous algorithm for the variable only case, we unify the generalizations of the original terms. This has already been done for the example above resulting in 69 unifiers as stated previously.

Now we have the generalizations of the two original terms, a substitution to instantiate them to the original terms, and a complete set of their unifiers. Every unifier of the original terms is a simultaneous instance of the substitution to instantiate the generalizations to the original terms and a unifier of the generalizations. So all that is necessary to get all the unifiers of the original terms is to unify (for each variable being substituted for) the value in the substitution and the value in the unifiers.

In the example, x_3 must have value a and y_1 must have value b. Thus, any unifier of $f(x_1x_1x_2x_3)$ and $f(y_1y_1y_2)$ which assigns to x_3 or y_2 a non-variable, i.e., a term of the form f(...) may be immediately excluded from consideration since the unification of it with the assignment including $x_3 \leftarrow a$ and $y_1 \leftarrow b$ will fail. (This constraint could be applied during the generation of sums of solutions to the equation rather than afterwards.) This constraint eliminates 63 of the 69 unifiers, leaving sums (1) {4,6}, (2) {2,4,6}, (3) {1,5,6}, (4) {1,2,5,6}, (5) {1,2,7}, and (6) {1,2,6,7} with associated unifiers

(1) $\{x_1 \leftarrow z_6, x_2 \leftarrow z_4, x_3 \leftarrow z_4, y_1 \leftarrow z_4, y_2 \leftarrow f(z_6z_6)\}$,

(2) $\{x_1 \leftarrow z_6, x_2 \leftarrow f(z_2z_4), x_3 \leftarrow z_4, y_1 \leftarrow z_4, y_2 \leftarrow f(z_2z_6z_6)\}$,

(3) $\{x_1 \leftarrow z_6, x_2 \leftarrow f(z_5z_5), x_3 \leftarrow z_1, y_1 \leftarrow z_5, y_2 \leftarrow f(z_1z_6z_6)\}$,

(4) $\{x_1 \leftarrow z_6, x_2 \leftarrow f(z_2z_5z_5), x_3 \leftarrow z_1, y_1 \leftarrow z_5, y_2 \leftarrow f(z_1z_2z_6z_6)\}$,

(5) $\{x_1 \leftarrow z_7, x_2 \leftarrow z_2, x_3 \leftarrow z_1, y_1 \leftarrow z_7, y_2 \leftarrow f(z_1z_2)\}$, and

(6) $\{x_1 \leftarrow f(z_6z_7), x_2 \leftarrow z_2, x_3 \leftarrow z_1, y_1 \leftarrow z_7, y_2 \leftarrow f(z_1z_2z_6z_6)\}$.

Unifying each of these with $\{x_1 \leftarrow x, x_2 \leftarrow y, x_3 \leftarrow a, y_1 \leftarrow b, y_2 \leftarrow z\}$, we obtain

(1) no unifier since $z_4 \leftarrow a$ and $z_4 \leftarrow b$ are not unifiable,

(2) no unifier since $z_4 \leftarrow a$ and $z_4 \leftarrow b$ are not unifiable,

(3) $\{x \leftarrow z_6, y \leftarrow f(bb), z \leftarrow f(az_6z_6)\}$ (= $\{y \leftarrow f(bb), z \leftarrow f(axx)\}$),

(4) $\{x \leftarrow z_6, y \leftarrow f(bbz_2), z \leftarrow f(az_2z_6z_6)\}$ (= $\{y \leftarrow f(bbz_2), z \leftarrow f(az_2xx)\}$),

(5) $\{x \leftarrow b, y \leftarrow z_2, z \leftarrow f(az_2)\}$ (= $\{x \leftarrow b, z \leftarrow f(ay))\}$), and

(6) $\{x \leftarrow f(bz_6), y \leftarrow z_2, z \leftarrow f(az_2z_6z_6)\}$ (= $\{x \leftarrow f(bz_6), z \leftarrow f(ayz_6z_6)\}$).

This is a complete set of unifiers of f(xxya) and f(bbz).

Since x_3 and y_1 of the variable only case correspond to a and b respectively, and a and b are not unifiable, any sum including solution 4 to the equation $2x_1+x_2+x_3 = 2y_1+y_2$ can be excluded from consideration since it would require (as in (1) and (2) above) the unification of a and b. As with the constraint on variables corresponding to non-variable terms not being assigned more than one variable (terms of the form f(...)) in the variable only case, this latter constraint on solutions can be applied during the generation of unifiers in the variable only case rather than afterwards. Elimination of solution 4 before generation of the 2^n sums, and elimination of sums which do not meet the first constraint would result in the formation only of unifiers (3), (4), (5), and (6) of the variable only case, each of which has a corresponding unifier in the general case.

More precisely, the algorithm consists of the following steps:

1. Form generalizations of the two terms replacing each distinct argument by a new variable.

2. Use the algorithm for the variable only case to generate unifiers for the generalizations of the two terms. The variable only case algorithm may be constrained to eliminate the generation of unifiers assigning more than one term to variables whose value must be a single term, and the generation of unifiers which will require the later unification of terms which are obviously not unifiable.

3. Unify for each variable in the substitution from step 1 and the unifiers from step 2 the

variable values and return the resulting assignments for variables of the original terms. This is a complete set of unifiers of the original terms.

Proof of Termination, Soundness, and Completeness of the AC Unification Algorithm

We will first establish the validity of eliminating arguments common to the two terms. This will be done by proving that any unifier of the terms is a unifier of the terms with a pair of common arguments removed and vice versa.

Theorem. Let $s_1,\dots,s_m,t_1,\dots,t_n$ be terms with $s_i=t_j$ for some i,j. Let θ be a unifier of $f(s_1\dots s_m)$ and $f(t_1\dots t_n)$, and let σ be a unifier of $f(s_1\dots s_{i-1}s_{i+1}\dots s_m)$ and $f(t_1\dots t_{j-1}t_{j+1}\dots t_n)$. Then (1) θ is a unifier of $f(s_1\dots s_{i-1}s_{i+1}\dots s_m)$ and $f(t_1\dots t_{j-1}t_{j+1}\dots t_n)$, and (2) σ is a unifier of $f(s_1\dots s_m)$ and $f(t_1\dots t_n)$.

Proof.

1. $f(s_i\theta f(s_1\dots s_{i-1}s_{i+1}\dots s_m)\theta) = f(s_1\dots s_m)\theta = f(t_1\dots t_n)\theta = f(t_j\theta f(t_1\dots t_{j-1}t_{j+1}\dots t_n)\theta)$, and $s_i\theta=t_j\theta$. Therefore $f(s_1\dots s_{i-1}s_{i+1}\dots s_m)\theta = f(t_1\dots t_{j-1}t_{j+1}\dots t_n)\theta$ and θ is a unifier of $f(s_1\dots s_{i-1}s_{i+1}\dots s_m)$ and $f(t_1\dots t_{j-1}t_{j+1}\dots t_n)$.

2. $f(s_1\dots s_{i-1}s_{i+1}\dots s_m)\sigma = f(t_1\dots t_{j-1}t_{j+1}\dots t_n)\sigma$ and $s_i\sigma=t_j\sigma$. Therefore $f(s_i\sigma f(s_1\dots s_{i-1}s_{i+1}\dots s_m)\sigma) = f(s_1\dots s_m)\sigma = f(t_1\dots t_n)\sigma = f(t_j\sigma f(t_1\dots t_{j-1}t_{j+1}\dots t_n)\sigma)$ and σ is a unifier of $f(s_1\dots s_m)$ and $f(t_1\dots t_n)$. QED.

The lemma below establishes that every non-negative integral solution to an equation of the form $a_1x_1+\dots a_mx_m = b_1y_1+\dots+b_ny_n$ is composable as a (non-negative integral weighted) sum of a fixed finite set of non-negative integral solutions. It also establishes a solution value within which all the non-negative integral solutions in the set may be found.

Lemma. Every non-negative integral solution $(x_1,\dots,x_m,y_1,\dots,y_n)$ to the equation $a_1x_1+\dots+a_mx_m = b_1y_1+\dots+b_ny_n$ with positive integral coefficients $a_1,\dots,a_m,b_1,\dots,b_n$ is an additive linear combination of non-negative integral solutions with value $a_1x_1+\ldots+a_mx_m$ $(=b_1y_1+\ldots+b_ny_n)$ less than or equal to the maximum of m and n times the maximum of the least common multiples of pairs of numbers one from $a_1,\ldots,a_m$ and one from $b_1,\ldots,b_n$.

Proof. Assume with no loss of generality that the least common multiple (lcm) of a_1 and b_1 is the maximum of the least common multiples and that $m \geq n$.

Proof by induction on the value of a solution k.

k=0. The solution with k=0 with $x_1=0$, ..., $x_m=0$, $y_1=0$, ..., $y_n=0$ is generable as the additive linear combination of non-negative integral solutions with value less than or equal to $m*\text{lcm}(a_1,b_1)$ with zero coefficients.

Assume the lemma is true for every non-negative integral solution with value less than or equal to k. Prove it is true for k.

Case 1. $k \leq m*\text{lcm}(a_1,b_1)$. In this case, the solution is included among the non-negative integral solutions with value less than or equal to $m*\text{lcm}(a_1,b_1)$ and the lemma is true.

Case 2. $k > m*\text{lcm}(a_1,b_1)$. Since $a_1x_1+\ldots+a_mx_m = k > m*\text{lcm}(a_1,b_1)$, and each $a_ix_i \geq 0$, at least one a_ix_i must be greater than $\text{lcm}(a_1,b_1)$, and x_i must be greater than $\text{lcm}(a_1,b_1)/a_i$. Similarly, since $b_1y_1+\ldots+b_ny_n = k > m*\text{lcm}(a_1,b_1)$, and each $b_jy_j \geq 0$, and $n \leq m$, at least one b_jy_j must be greater than $\text{lcm}(a_1,b_1)$, and y_j must be greater than $\text{lcm}(a_1,b_1)/b_j$. Consider the solution with $x_i=\text{lcm}(a_i,b_j)/a_i$, $y_j=\text{lcm}(a_i,b_j)/b_j$, and all other variables zero. This is just the solution in lowest terms involving only x_i and y_j and has value $\text{lcm}(a_i,b_j) \leq \text{lcm}(a_1,b_1)$. Since $\text{lcm}(a_1,b_1)/a_i \geq \text{lcm}(a_i,b_j)/a_i$ and $\text{lcm}(a_1,b_1)/b_j \geq \text{lcm}(a_i,b_j)/b_j$ by the maximality of $\text{lcm}(a_1,b_1)$, the solution involving only x_i and y_j can be subtracted from the solution with value k leaving a non-negative integral solution as result. But this difference solution has value $k-\text{lcm}(a_i,b_j) < k$ and is thus composable from solutions with value less than or equal to $m*\text{lcm}(a_1,b_1)$. Therefore, the solution with value $k > m*\text{lcm}(a_1,b_1)$ is the sum of some solution involving only x_i and y_j with value less than or equal to $\text{lcm}(a_1,b_1)$ and some other set of solutions with value less than or equal to $m*\text{lcm}(a_1,b_1)$ and the lemma is true for this case. QED.

The lemma proves an upper bound on solution values that must be examined in the determination of a complete set of non-negative integral solutions which span the non-negative integral solution space by addition. We believe that tighter bounds can be proved. Although a proof for a tighter bound would be desirable, it should be noted that a lower proven bound would not reduce the number of found solutions theoretically necessary, but only decreases the cost of computing them, and would have no effect on the form or number of unifiers returned by the algorithm. This is true since any additional solutions discovered using a higher bound than necessary must be composable from solutions bounded by any proven lower bound and would therefore be recognized as redundant and be omitted.

The maximum of the least common multiples of the coefficients one from the left side and one from the right side of the equation is a lower bound on solution values which must be examined, i.e., solutions with at least this value must be examined. This is because one of the needed solutions not otherwise generable is the solution involving only the variables with those two coefficients with maximum least common multiple and having value equal to the maximum least common multiple.

Theorem. The AC unification algorithm for terms with associative and commutative function with only variables as arguments always terminates, is sound (returns no substitutions which are not unifiers), and is complete (every unifier is an instance of a returned unifier).

Proof. The algorithm is guaranteed to terminate since it performs a finite number of operations on the finite number of non-negative integral solutions generated from the equation corresponding to the two terms. The generation of these solutions is finite due to the trial solution values being bounded.

The algorithm is sound since each solution of the derived equation causes the introduction into each of the instantiated terms of an equal number of new variable occurrences. Thus, the two instantiated terms have the same number of occurrences of each new variable and are therefore unified.

Any unifier must assign to each variable a term of the form t_i (whose function symbol is not f) or a term $f(t_1{}^{n_1}...t_m{}^{n_m})$ (with n_i occurrences of term t_i as arguments of f). Let k be the cardinality of the set of such terms t_i in any solution to the unification of a pair of terms with only variables as arguments. The two instantiated terms must have an equal number of occurrences of each of these k terms as arguments of f. That is, $a_1c_{i1}+...+a_mc_{im} = b_1d_{i1}+...+b_nd_{in}$ $(1 \leq i \leq k)$ where m is the number of distinct variables in the first term being unified, n is the number of distinct variables in the second term, a_j is the multiplicity of the jth variable in the first term, b_j is the multiplicity of the jth variable in the second term, c_{ij} is the number of occurrences of term i in variable j in the first term, and d_{ij} is the number of occurrences of term i in variable j in the second term.

Each tuple $(c_{i1},...,c_{im},d_{i1},...,d_{in})$ is a solution to the equation $a_1x_1+...+a_mx_m = b_1y_1+...+b_ny_n$ corresponding to the terms being unified. It can thus (according to the lemma) be formed as the sum of certain non-negative integral solutions to the equation weighted by positive integers.

Consider the unifier corresponding to the sum of all those solutions to the equation which are required in the formation of any of the tuples $(c_{i1},...,c_{im},d_{i1},...,d_{in})$. We will show that the hypothesized unifier is an instance of this unifier returned by the algorithm.

Include in the value of the new variable associated with each of these solutions a number of occurrences of term i equal to the coefficient of the solution in the weighted sum. This will result in the proper assignment of c_{ij} occurrences of term i to each variable j of the first term and d_{ij} occurrences of term i to each variable j of the second term.

Do this for each of the k terms in the solution. Let no other or additional terms be included in the values of the new variables.

This assignment of terms in the solution to new variables associated with equation solutions generated in the unification process results in the correct number c_{ij} or d_{ij} of each term being assigned to each variable of the original two terms.

Thus, any solution to the unification of two terms with only variables as arguments is an instance of a returned unifier and the algorithm is complete. QED.

Theorem. The AC unification algorithm for general terms with associative and commutative function always terminates, is sound, and is complete.

Proof. Let s and t be any two terms being unified. Let s^* and t^* be the terms resulting from replacing each distinct term by a new variable. s^* and t^* are generalizations of s and t respectively, i.e., $s^*\theta=s$ and $t^*\theta=t$ for some θ of the form $\{\ldots,x_i\leftarrow c_i,\ldots\}$ where each x_i is a new variable and each c_i is the term in s or t it replaces in s^* or t^*.

Let $\{\sigma_j\}$ denote the unifiers of s^* and t^* returned by the unification algorithm for terms with associative and commutative function with only variables as arguments. Each σ_j is of the form $\{\ldots,x_i\leftarrow d_i,\ldots\}$ where each x_i is a variable of s^* or t^* and d_i is the term assigned to it by the unification algorithm. According to the previous theorem, unification terminates, is sound, and is complete for this case.

Simultaneous instances of θ and σ_j represent unifiers of s and t since $s^*\theta=s$, $t^*\theta=t$, and $s^*\sigma_j=t^*\sigma_j$.

Unifying each c_i with each d_i of a returned unifier σ_j of s^* and t^* results in (by the assumption of termination, soundness, and completeness of the recursive call on the unification algorithm for terms of lesser complexity) a complete set of unifiers for the original terms s and t. QED.

Conclusion

We have presented an algorithm for unifying general terms with associative and commutative function. We have proven that the algorithm is guaranteed to terminate, is sound, and is complete.

The advantages of this algorithm as compared to other approaches to unifying such terms are that the associativity and commutativity properties need not be axiomatized and that all the unifiers of a pair of such terms are immediately returned eliminating the unnecessary and redundant computation often occurring in other approaches which generate only some of the unifiers at each step with no indication of when all the unifiers have been generated.

Bibliography

1 Chang, C. L. and Lee, R. C. T. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

2 Nevins, A. J. A human oriented logic for automatic theorem proving. *J. ACM 21*, 4 (Oct. 1974), 606-621.

3 Plotkin, G. D. Building-in equational theories. In Meltzer, B. and Michie, D. (Eds.). *Machine Intelligence 7*, Edinburgh University Press, Edinburgh, 1972, pp. 73-90.

4 Reboh, R. and Sacerdoti, E. A preliminary QLISP manual. Technical Note 81, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, Calif., Aug. 1973.

5 Robinson, G. and Wos, L. Paramodulation and theorem-proving in first-order theories with equality. In Meltzer, B. and Michie, D. (Eds.). *Machine Intelligence 4*, Edinburgh University Press, Edinburgh, 1969, pp. 135-150.

6 Robinson, J. A. A machine-oriented logic based on the resolution principle. *J. ACM 12*, 1 (Jan. 1965), 23-41.

7 Rulifson, J. F., Derksen, J. A. and Waldinger, R. J. QA4: a procedural calculus for intuitive reasoning. Technical Note 73, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, Calif., Nov. 1972.

8 Slagle, J. R. Automated theorem-proving for theories with simplifiers, commutativity, and associativity. *J. ACM 21*, 4 (Oct. 1974), 622-642.

9 Stickel, M. E. Unification algorithms for artificial intelligence languages. Chapter of incomplete Ph.D. Dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Penn.