Appeared in CADE 9, 1988, LNCS 310

Adventures in Associative-Commutative Unification

Patrick Lincoln and Jim Christian
MCC, Systems Languages
3500 W. Balcones Cntr. Dr.
Austin, TX, 78759
(512) 338-3717

April 15, 1994

Abstract

We have discovered an efficient algorithm for matching and unification in associative-commutative (AC) and associative-commutative-idempotent (ACI) equational theories. In most cases of AC unification and in all cases of ACI unification our method obviates the need for solving diophantine equations, and thus avoids one of the bottlenecks of other associative-commutative unification techniques. The algorithm efficiently utilizes powerful constraints to eliminate much of the search involved in generating valid substitutions. Moreover, it is able to generate solutions lazily, enabling its use in an SLD-resolution-based environment like Prolog. We have found the method to run much faster and use less space than other associative-commutative unification procedures on many commonly encountered AC problems.

1 Introduction

A number of computer science applications, including term rewriting, automatic theorem proving, software verification, and database retrieval require AC unification (also called "bag unification") or ACI unification ("set unification"). A complete unification algorithm for AC theories was developed several years ago by Mark Stickel [16]. Independently, Livesey and Siekmann published a similar algorithm for AC and ACI unification. Their procedures center around generating solutions to a linear diophantine equation, each coefficient of which represents the multiplicity of some subterm in one of the unificands. There are two nagging properties of this method. First, it requires generating a basis for the solution space of the diophantine equation. Second, there can be a large amount of search involved in actually generating solutions once a basis is discovered.

We have found an algorithm for dealing with associative-commutative theories without resorting to the solution of diophantine equations. By weakening the variable abstraction introduced by Stickel, most cases of AC and all cases of ACI unification can be solved by working only with equations in which all coefficients are unity. The basis of solutions of such an equation possesses a highly regular structure; this allows us to optimize the representation of the problem and avoid spending time finding a basis. We are able instead

to begin generating unifiers almost immediately. In addition, our representation allows the incorporation of several simple but powerful constraints in a way that is much more natural and efficient than previous methods have allowed.

Our algorithm can solve AC matching problems (so-called "one-way unification"), most cases of AC unification, and all cases of ACI unification very efficiently – in most cases, several times faster than Stickel's algorithm. However, if repeated variables occur in one unificand, our algorithm may return redundant unifiers. If repeated variables occur in both unificands, our algorithm may not terminate. In these easily detected cases, it suffices to dispatch to some complete algorithm, like Stickel's; the overhead in making the decision to dispatch is negligible. Fortunately, these cases occur in only a few percent of many applications of AC unification like Knuth-Bendix completion [12]. In some applications like database query compilation, these cases never occur. Thus our procedure can achieve a significant improvement in average execution speed of AC unification. Furthermore, our procedure requires nominal space overhead for generating solutions, and is amenable to the lazy generation of solutions required in an SLD-resolution environment like that of Prolog. We give a summary of our full paper [2] in the text that follows.

2 History Of AC Unification

Mark Stickel was the first to develop a complete, terminating algorithm for AC unification; the algorithm was initially presented in 1975 [15]. Livesey and Siekmann published a similar algorithm in 1976 [13]. Most AC unification procedures in use today are essentially modifications of that of Stickel or of Livesey and Siekmann, but a few novel approaches have been proposed. Within the loose framework of Stickel's method there are two hard problems: generating a basis of solutions to linear homogeneous diophantine equations, and searching through all combinations of this basis for a solution to the given AC unification problem.

Since Gordan's study of diophantine equations in 1873, only in the last few years has there been any significant progress made regarding the generation of their bases. Fortenbacher, Huet, and Lankford have separately proposed a number of refinements to Gordan's basic method. Recently, Zhang has discovered a class of diophantine equations which can be quickly solved. However, no published algorithm has proven to be superior in all cases. [5, 4, 7, 12, 6, 18].

The extraction of solutions to AC problems given a basis of solutions to the diophantine equation is also an area of concern. In the past few years Fortenbacher [4] has proposed a method of reducing the search space by eliminating certain basis elements. Claude Kirchner has recently developed an AC unification algorithm within the framework of the Martelli-Montanari unification procedure, but, like Stickel's, his method requires solving diophantine equations [11, 14]. Also, Hullot invented an algorithm for AC unification which involves ordered partitions of multisets [9]. While his algorithm is faster than Stickel's, it does not seem to offer nearly the dramatic speed increases we have obtained with our procedure. We have not implemented Hullot's algorithm, but base our judgement on timing comparisons listed in his paper. In Germany, Büttner has developed a parallel algorithm for AC unification [1]. The method involves linear algebraic operations in multi-dimensional vector spaces, but he fails to provide the details necessary for a realistic comparison. Recently, Kapur [10]

has developed an algorithm based on Stickel's method that uses Zhang's equation solving technique. The survey by Huet and Oppen [8] summarizes results in related areas.

3 Our Method

There are two basic difficulties with previous algorithms. First, generation of a basis for a diophantine equation is an expensive operation. Second, given a basis, the search which must be performed to produce solutions can be very expensive. It is thus necessary to enforce several non-trivial constraints [4, 17]. Fortenbacher [4] has described many obvious optimizations of Stickel's method, and Stickel himself has implemented quite impressive constraints on the generation of solutions which tame this search problem. As we shall soon see, our algorithm is able to exploit similar constraints in a very natural and efficient way.

3.1 Slaying the Diophantine Dragon

The preparation phase of our algorithm is very similar to previous approaches. First, both terms are put through a "flattening" operation which removes nested AC function symbols. This operation can be viewed more precisely as term reduction by root application of the rewrite rule $f(t_1, \ldots, f(s_1, \ldots, s_m), \ldots, t_n) \longrightarrow f(t_1, \ldots, s_1, \ldots, s_m, \ldots, t_n)$. Hence, the term $f(f(a,a),a,f(g(u),y,x))$ will be flattened to $f(a,a,a,g(u),y,x)$, while $f(a,f(b,g(c)),f(y,y),z)$ will be changed to $f(a,b,g(c),y,y,z)$. The flattened term $f(s_1,s_2,\ldots,s_n)$ is merely syntactic sugar for the right-associative normal form $f(s_1,f(s_2,\ldots,f(s_{n-1},s_n)\ldots))$. The validity of the flattening operation is guaranteed by the associativity axiom, which implies that nesting of AC function symbols is largely irrelevant.

After flattening terms, the next step is to remove subterms which occur pairwise in both unificands. For instance, after deleting duplicate subterms from $f(a,a,a,g(u),y,x)$ and $f(a,b,g(c),y,y,z)$ we obtain the terms $f(a,a,g(u),x)$ and $f(b,g(c),y,z)$, specifically by removing one occurrence of a and one occurrence of y from each.

We suppose that both terms are sorted so that atomic constants are grouped together, followed by function terms, followed by variables. For instance, $f(a,g(u),a,y,a,x)$ would be sorted to produce $f(a,a,a,g(u),y,x)$. An important non-intuitive point is that compared to the time required to generate all unifiers, the time spent flattening and sorting terms is insignificant.

In the ACI case, repeated constants, terms, and variables are removed from each unificand. For example, $f(a,a,b,x,x)$ is simplified to $f(a,b,x)$. In the AC case, this step is omitted.

Now, our generalization step differs from others, in that we assign a distinct variable for *each* argument. Thus, while Stickel's algorithm would convert the $f(a,a,g,x)$ to $f(x_1,x_1,x_2,x_3)$, ours will produce $f(x_1,x_2,x_3,x_4)$. Effectively, we convert the problem of solving the unification problem $f(X_1,\ldots,X_m) = f(Y_1,\ldots,Y_n)$ into the equivalent conjunction of problems $f(x_1,\ldots,x_m) = f(y_1,\ldots,y_n) \wedge x_1 = X_1 \wedge \ldots \wedge x_m = X_m \wedge y_1 = Y_1 \wedge \ldots \wedge y_n = Y_n$, where the x_i and y_j are distinct variables.

Notice that the diophantine equation corresponding to any pair of such generalized terms

		a	a	$g(u)$	x
		x_1	x_2	x_3	x_4
b	y_1	$z_{1,1}$	$z_{1,2}$	$z_{1,3}$	$z_{1,4}$
$g(c)$	y_2	$z_{2,1}$	$z_{2,2}$	$z_{2,3}$	$z_{2,4}$
y	y_3	$z_{3,1}$	$z_{3,2}$	$z_{3,3}$	$z_{3,4}$
z	y_4	$z_{4,1}$	$z_{4,2}$	$z_{4,3}$	$z_{4,4}$

	C	T	V
C	0	0	$\leftrightarrow$
T	0	$\updownarrow$	$\leftrightarrow$
V	$\updownarrow$	$\updownarrow$	any

Table 1: Matrix for a simple problem and some constraints

will have only unit coefficients. Such an equation has a few nice properties, namely given a diophantine equation of the form $x_1 + \ldots + x_m = y_1 + \ldots + y_n$, the minimal solution basis is that set of solutions such that, for each solution, exactly one x_i has value one, exactly one y_j has value one, and all other variables have value zero. Also, the number of basis solutions is nm.

Knowing that the basis has such a nice, regular structure, we need not explicitly generate it; for, given only the respective arities of the generalized unificands, we can immediately construct a two dimensional matrix, where each column is labeled with an x_i, and each row is labeled with one of the y_j. Each entry i, j in the matrix is a boolean value, that corresponds to a new variable, $z_{i,j}$, which represents the solution vector which assigns a one to x_j and y_i. Thus every true boolean value i, j in a solution matrix corresponds to one basis element of the solution of the diophantine equation. Any assignment of true and false to all the elements of a matrix represents a potential solution to the AC unification problem in the same way that any subset of the basis elements of the diophantine equation represents a potential solution to the same AC problem.

For instance, suppose we are given the (already flattened) unificands $f(a, a, g(u), x)$ and $f(b, g(c), y, z)$. Substituting new variables for each argument, we obtain $f(x_1, x_2, x_3, x_4)$ and $f(y_1, y_2, y_3, y_4)$. The associated solution matrix is displayed in Table 1

In our implementation, we do not create the entire n by m matrix; rather, we will utilize a more convenient and compact data structure. But for now, let us pretend that the matrix is represented as a simple 2-dimensional array. And as we will demonstrate below, the matrix representation is inherently amenable to constraining the search for unifiers.

3.2 Constraining Search

Remember that unificands are sorted in the preparation step of our algorithm. Hence, a given solution matrix comprises nine regions, illustrated in Table 1. In the table, C, T, and V stand, respectively, for atomic constants, functional terms, and variables. An entry in the lower left region of the matrix, for instance, corresponds to an assignment in the (unprepared) unificands of a constant in one and a variable in the other.

As Table 1 indicates, there are several constraints on the distribution of ones and zeros within a solution matrix. First, notice that there must be at least one non-zero entry in each row and column of a solution matrix, so that all variables in the generalized terms receive an assignment. The upper left corner involves assignments to incompatible constants (since we have removed duplicate arguments from the unificands, no constants from one term can

		a	a	$g(u)$	x
		x_1	x_2	x_3	x_4
b	y_1	0	0	0	1
$g(c)$	y_2	0	0	1	0
y	y_3	0	0	0	1
z	y_4	1	1	0	0

Unifying substitution:

$$x \leftarrow f(b, y)$$
$$z \leftarrow f(a, a)$$
$$u \leftarrow c$$

Table 2: A solution to the matrix

possibly unify with any constant from the other term). This part of any solution matrix, then, must consist only of zeros. Similarly, the C/T and T/C regions of a solution matrix must contain all zeros. The C/V region is constrained to have exactly a single one in each column, since any additional ones would cause the attempted unification of a functional term, say $f(z_{1,1}, z_{1,2})$, with a constant. Similarly, any T row or T column must contain exactly one one. Finally, the V/V region of a matrix can have any combination of ones and zeros which does not leave a whole row or column filled only with zeros.

In reality, the nine regions depicted in Table 1 are further partitioned to handle repeated variables and constants; for now, we will ignore this detail, and assume that all arguments within a unificand are distinct.

3.3 Generating Solutions

Once a unification problem has been cast into our matrix representation, it is not a difficult matter to find unifying substitutions. The approach is to determine a valid configuration of ones and zeros within the matrix, perform the indicated assignments to the variables in the generalized terms, and finally unify the arguments of the original unificands with their variable generalizations.

Consider the matrix in Table 1. We know that $z_{1,1}$ must be zero, since it falls within the C/C region of the matrix. Likewise, $z_{1,2}$, $z_{1,3}$, $z_{2,1}$, and $z_{2,2}$ must always be zero. In fact, the only possible position for the required one in the y_1 column is at $z_{1,4}$. Filling out the rest of the matrix, we arrive at the solution shown in Table 2; after assigning the nonzero $z_{i,j}$'s to the x and y variables, and then unifying the variables with the original unificand arguments, we obtain the substitution shown at the side of Table 2. Note that the step at which $g(u) \leftarrow g(c)$ was derived, and recursively solved to produce $u \leftarrow c$ has been omitted.

3.4 Lazy generation of solutions

Enumerating solutions essentially amounts to performing simple binary operations on regions of the matrix. For instance, in the variable-constant region of the martix, a binary rotate instruction and a few numeric comparisons usually suffice to generate the next solution from the current state.

	a	a
x	1	0
y	0	1

	a	a
x	0	1
y	1	0

Table 3: Redundant matrix configurations for $f(a, a) = f(x, y)$

3.5 Repeated terms

Until now, we have assumed that all arguments within a unificand are distinct. This will always be true for ACI unification, since the repeated terms are removed during preparation. However, this is not necessarily the case for AC unification. In practice, repeated terms occur infrequently; Lankford, for instance, has found that more than 90 percent of the unification problems encountered in some completion applications involve unificands with distinct arguments. Nevertheless, the ability to handle repeated arguments is certainly desirable.

Our algorithm can easily be adapted to handle repetitions in constants and functional terms in either or both unificands. Repeated variables are more difficult to handle. If they occur in a single unificand, our algorithm is complete and terminating, but may return redundant unifiers. If they occur in both unificands, the algorithm might generate subproblems at least as hard as the original, and thus not terminate. Stickel's algorithm can be employed whenever repeated variables are detected; the overhead involved in making this decision is negligible. Thus in the worst cases we do simple argument checking, and dispatch to Stickel's algorithm. We have several methods of minimizing the use of Stickel's algorithm but we have not yet discovered a straightforward, general method. In [2] we prove that our procedure does indeed terminate whenever variables are repeated in at most one of the unificands.

The set of unifiers returned by our algorithm is guaranteed to be complete. However, the set of unifiers may not be minimal. The set of unifiers returned by Stickel's algorithm is similarly not guaranteed to be minimal. If a minimal set of unifiers is required, it suffices to simply remove elements of the non-minimal set which are subsumed by other unifiers in the set.

Assuming no repeated variables in one term, our algorithm can handle arbitrary repetitions of constants and functional terms. But before disclosing the modification to our algorithm which facilitates handling of repeated arguments, we show with a simple example why the modification is needed. Suppose we wish to unify $f(a, a)$ with $f(x, y)$, which is a subproblem of the earlier example. Without alteration, our algorithm as so far stated will generate the two configurations shown in Table 3. While the matrix configurations are distinct, they represent identical unifying substitutions – namely $\{x \leftarrow a, y \leftarrow a\}$.

The solution to this problem is surprisingly simple. In short, whenever adjacent rows represent the same term, we require that the contents of the upper row, interpreted as a binary number, be greater than or equal to the contents of the lower row. A symmetric restriction is imposed on columns. See the full paper for a detailed explanation of these restrictions.

3.6 An Algorithm for Associative-Commutative Unification

Until now, we have concentrated almost exclusively on the matrix solution technique which lies at the heart of our AC unification algorithm. Following is a statement of the unification algorithm proper. This will serve, in the next section, as a basis for results involving the completeness and termination of our method. The algorithm is presented as four procedures: `AC-Unify, Unify-With-Set, Unify-Conjunction, and Matrix-Solve`.

`Procedure AC-Unify`: Given two terms x and y, return a complete set of unifiers for the equation $x =_{AC} y$.

Step 1 If x is a variable, then see if y is a functional term and x occurs in y. If both are true, return `fail`. Otherwise, return $\{\{x \leftarrow y\}\}$, unless $x = y$ — in that case, return the null substitution set $\{\{\}\}$.

Step 2 If y is a variable, then see if y occurs in x. If it does, return `fail`. Otherwise, return $\{\{y \leftarrow x\}\}$.

Step 3 If x and y are distinct constants, return `fail`.

Step 4 If x and y are the same constant, return $\{\{\}\}$.

Step 5 At this point, x and y are terms of the form $f(x_1, \ldots, x_m)$ and $g(y_1, \ldots, y_n)$. If $f \neq g$, return `fail`.

Step 6 If f is not an AC function symbol, and $m = n$, then call procedure `Unify-With-Set` with the substitution set $\{\{\}\}$ and the conjunction of equations $x_1 =_{AC} y_1 \wedge \ldots \wedge x_n =_{AC} y_n$, and return the result. If $m \neq n$, return `fail`.

Step 7 Flatten and sort x and y, and remove arguments common to both terms. Call the resulting terms $\hat{x}$ and $\hat{y}$, respectively. Assume $\hat{x} = f(x_1, \ldots, x_j)$ and $\hat{y} = f(y_1, \ldots, y_k)$. Set up the conjunction of equations $f(X_1, \ldots, X_j) =_{AC} f(Y_1, \ldots, Y_k) \wedge X_1 =_{AC} x_1 \wedge \ldots \wedge X_j =_{AC} x_j \wedge Y_1 =_{AC} y_1 \wedge \ldots \wedge Y_k =_{AC} y_k$, where the X_i and Y_i are new, distinct variables. Call this conjunction E.

Step 8 Let T be the result of applying `Matrix-Solve` to the conjunction E. If $T =$ `fail`, return `fail`.

Step 9 Call procedure `Unify-With-Set` with the set of substitutions T and the conjunction of equations $X_1 =_{AC} x_1 \wedge \ldots \wedge X_j =_{AC} x_j \wedge Y_1 =_{AC} y_1 \wedge \ldots \wedge Y_k =_{AC} y_k$, and return the result.

`Procedure Unify-With-Set`: Given a set of substitutions T and a conjunction of equations E, return $\bigcup_{\theta \in T} CSU(\theta E)$, where $CSU(X)$ is a complete set of unifiers for X.

Step 1 Let $S = \{\}$.

Step 2 For each $\theta \in T$, set S to $S \cup \{\bigcup_{\sigma_j \in Z} \{\theta \cup \sigma_j\}\}$, where Z is the result of applying procedure `Unify-Conjunction` to θE.

Step 3 Return S.

`Procedure Unify-Conjunction` Given a conjunction of equations $E = e_1 \wedge \ldots \wedge e_n$, return a complete set of unifiers for E.

Step 1 . Let V be the result of calling procedure `AC-Unify` with e_1. If $n = 1$, return V. If $V =$ `fail`, return `fail`.

Step 2 . Call procedure `Unify-With-Set` with the set of substitutions V and the conjunction $e_2 \wedge \ldots \wedge e_n$, and return the result.

`Procedure Matrix-Solve` Given a conjunction of equations $f(X_1, \ldots, X_m) =_{AC} f(Y_1, \ldots, Y_n) \wedge X_1 =_{AC} x_1 \wedge \ldots \wedge X_m =_{AC} x_m \wedge Y_1 =_{AC} y_1 \wedge \ldots \wedge Y_n =_{AC} y_n$, where the X_i and Y_i are distinct variables, determine a set of substitutions which will unify $f(X_1, \ldots, X_m)$ with $f(Y_1, \ldots, Y_n)$.

Step 1 Establish an m-by-n matrix M where row i (respectively column j) is headed by X_i (Y_j).

Step 2 Generate an assignment of 1s and 0s to the matrix, subject to the following constraints. If x_i (y_j) is a constant or functional term, then exactly a single 1 must occur in row i (column j). If x_i and y_j are both constants, or if one is a constant and the other is a functional term, then $M[i, j] = 0$. Also, there must be at least a single 1 in each row and column. Finally, if $x_i = x_{i+1}$ for some i, then row i interpreted as a binary number must be less than or equal to row $i + 1$ viewed as a binary number. (Symmetrically for y_j and y_{j+1}.)

Step 3 With each entry $M[i, j]$, associate a new variable $z_{i,j}$. For each row i (column j) construct the substitution $X_i \leftarrow f(z_{i,j_1}, \ldots, z_{i,j_k})$ where $M[i, j_l] = 1$, or $X_i \leftarrow z_{i,j_k}$ if $k = 1$. (symmetrically for Y_j).

Step 4 Repeat Step 2 and Step 3 until all possible assignments have been generated, recording each new substitution. If there is no valid assignment, return `fail`.

Step 5 Return the accumulated set of substitutions.

When there are repeated variables in both unificands, it is possible that our algorithm will not terminate. For example, in the unification of $f(x, x)$ with $f(y, y)$ one of the recursive subproblems generated is identical (up to variable renaming) to the original problem. However, as we prove in the full version of this paper [2] our algorithm is totally correct in other cases.

4 Benchmarks

Although we have not presented the details of our actual implementation, it should be obvious that more efficient data structures exist than an n by m boolean matrix. In particular, finding the next matrix configuration often reduces to one lisp *incf* and a few comparisons in our most optimized code. In order to quickly generate a unifier from a matrix configuration we utilize some auxiliary data structures.

The table on the next page reflects the time in seconds necessary to prepare unificands and to find and construct all AC unifiers. For each problem, timings were supplied by Kapur and Zhang (RRL), Stickel (SRI), and by ourselves (MCC). All data were collected on a Symbolics 3600 with IFU. As shown in the table, our algorithm is consistently three to five times faster than Stickel's and Kapur's.

These benchmarks do not include any problems with repeated variables, since in such cases, our algorithm would either return non-minimal sets of unifiers, or it would dispatch to Stickel's procedure. This is not as serious a concession as it might appear, since the most common cases of AC Unification are the ones without repeated variables. In fact, Lankford has found that less than 8 percent of uses of AC unification in applications like Knuth-Bendix completion have repetitions of anything, and less than three percent have repetitions on both sides [12]. Also, in the case of ACI, variables are never repeated.

Problem	♯	RRL	SRI	MCC
$xab = ucde$	2	0.020	0.018	0.005
$xab = uccd$	2	0.023	0.011	0.005
$xab = uccc$	2	0.018	0.008	0.004
$xab = uvcd$	12	0.045	0.047	0.013
$xab = uvcc$	12	0.055	0.032	0.014
$xab = uvwc$	30	0.113	0.096	0.034
$xab = uvwt$	56	0.202	0.171	0.079
$xaa = ucde$	2	0.028	0.013	0.005
$xaa = uccd$	2	0.023	0.009	0.004
$xaa = uccc$	2	0.021	0.006	0.005
$xaa = uvcd$	8	0.043	0.032	0.010
$xaa = uvcc$	8	0.035	0.020	0.011
$xaa = uvwc$	18	0.087	0.062	0.023
$xaa = uvwt$	32	0.192	0.114	0.051

Problem	♯ solns	RRL	SRI	MCC
$xya = ucde$	28	0.093	0.094	0.024
$xya = uccd$	20	0.068	0.050	0.018
$xya = uccc$	12	0.045	0.026	0.013
$xya = uvcd$	88	0.238	0.247	0.064
$xya = uvcc$	64	0.211	0.133	0.048
$xya = uvwc$	204	0.535	0.538	0.160
$xya = uvwt$	416	0.918	1.046	0.402
$xyz = ucde$	120	0.375	0.320	0.118
$xyz = uccd$	75	0.185	0.168	0.072
$xyz = uccc$	37	0.093	0.073	0.038
$xyz = uvcd$	336	0.832	0.840	0.269
$xyz = uvcc$	216	0.498	0.431	0.171
$xyz = uvwc$	870	2.050	2.102	0.729
$xyz = uvwt$	2161	5.183	5.030	1.994

5 Future Extensions

With simple modifications, our algorithm can apparently handle arbitrary combinations of associativity, commutativity, identity, and idempotence. We say "apparently" because we have not yet proven completeness or termination in all these cases, but preliminary findings have been encouraging. Of particular interest is unification with the single axiom of associativity, which corresponds to the word problem in free semigroups. The matrix representation of unifiers seems well suited to capturing this limited form of equational theory, but the details of this are beyond the scope of this report.

6 Conclusion

We have just described an algorithm which we believe to be the most efficient way of solving a large class of associative-commutative matching and unification problems. The algorithm obviates the need for solving diophantine equations, and it utilizes a matrix representation which conveniently enforces powerful search constraints. Compared to Stickel's and Kapur's procedures, our method often yields a significant improvement in speed. Certainly, applications of AC unification stand to benefit from our research.

We would like to thank Dallas Lankford for introducing us to his diophantine basis generation algorithm, and for supplying us with pointers to some useful information. We would also like to thank Hassan Aït-Kaci, Mike Ballantyne, Woody Bledsoe, Bob Boyer, and Roger Nasr for their comments, criticisms, and *laissez-faire* supervision. Finally, we would like to thank Mark Stickel, Hantao Zhang, and Deepak Kapur, for their insightful criticisms of an earlier draft of this paper, and for supplying benchmark times.

References

[1] Wolfram Büttner. "Unification in Datastructure Multisets". *Journal of Automated Reasoning*, 2 (1986) 75-88.

[2] Jim Christian and Pat Lincoln "Adventures in Associative-Commutative Unification" MCC Technical Report Number ACA-ST-275-87, Microelectronics and Computer Technology Corp., Austin, TX, Oct 1987.

[3] François Fages. "Associative-Commutative Unification". *Proceedings 7th International Conference on Automated Deduction*, Springer Verlag. Lecture Notes in Computer Science, Napa Valley, (California), 1984.

[4] Albrecht Fortenbacher. "An Algebraic Approach to Unification Under Associativity and Commutativity" Rewriting Techniques and Applications, Dijon, France, May 1985, ed Jean-Pierre Jouannaud. Springer-Verlag Lecture Notes in Computer Science Vol. 202, (1985) pp. 381-397

[5] P. Gordan, "Ueber die Auflösung linearer Gleichungen mit reelen Coefficienten". *Mathematische Annalen*, VI Band, 1 Heft (1873), 23-28.

[6] Thomas Guckenbiehl and Alexander Herold. "Solving Linear Diophantine Equations". Universitat Kaiserslautern, Fachbereich Informatik, Postfach 3049, 6750 Kaiserslautern.

[7] Gérard Huet. "An Algorithm to Generate the Basis of Solutions to Homogeneous Linear Diophantine Equations". IRIA Research Report No. 274, January 1978.

[8] Gérard Huet and D.C.Oppen. "Equations and Rewrite Rules: a Survey". In Formal Languages: Perspectives and Open Problems, ed R. Book, Academic Press, 1980.

[9] J.M. Hullot. "Associative Commutative Pattern Matching". *Proceedings IJCAI-79*, Volume One, pp406-412, Tokyo, August 1979.

[10] Deepak Kapur, G. Sivakumar, H. Zhang. "RRL: A Rewrite Rule Laboratory". Proceedings of CADE-8, pp 691-692, Oxford, England, 1986.

[11] Claude Kirchner. "Methods and Tools for Equational Unification". in *Proceedings of the Colloquium on the Resolution of Equations in Algebraic Structures*, May 1987, Austin, Texas.

[12] Dallas Lankford. "New Non-negative Integer Basis Algorithms for Linear Equations with Integer Coefficients". May 1987. Unpublished. Available from the author, 903 Sherwood Drive, Ruston, LA 71270.

[13] M. Livesey and J. Siekmann. "Unification of A + C-terms (bags) and A + C + I-terms (sets)". Intern. Ber. Nr. 5/76, Institut für Informatik I, Unifersität Karsruhe, 1976.

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

[15] Mark Stickel. "A complete unification algorithm for associative-commutative functions" *Proc. 4th IJCAI*, Tbilisi (1975), pp.71-82.

[16] Mark Stickel. "A Unification Algorithm for Associative-Commutative Functions". *JACM*, Vol.28, No.3, July 1981, pp.423-434.

[17] Mark Stickel. "A Comparison of the Variable-Abstraction and Constant-Abstraction methods for Associative-Commutative Unification" *Journal of Automated Reasoning*, Sept 1987, pp.285-289.

[18] Hantao Zhang "An Efficient Algorithm for Simple Diophantine Equations", Tech. Rep. 87-26, Dept. of Computer Science, RPI, 1987.