Role of Term Symmetry in E-Completion Procedures

Ralph W. Wilkerson
Missouri University of Science and Technology, ralphw@mst.edu

Blayne E. Mayfield

The Role of Term Symmetry in E-completion Procedures

Ralph W. Wilkerson
Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401

Blayne E. Mayfield
Department of Computer Science
Oklahoma State University
Stillwater, OK 74078

Abstract

A major portion of the work and time involved in completing an incomplete set of reductions using an E-completion procedure such as the one described by Knuth and Bendix or its extension to associative-commutative equational theories as described by Peterson and Stickel is spent calculating critical pairs and subsequently testing them for confluence and coherence. A pruning technique which removes from consideration those critical pairs that represent redundant or superfluous information can make a marked difference in the run time and efficiency of an E-completion procedure to which it is applied. In this paper, a technique is proposed for removing critical pairs from consideration at various points before, during, or after their formation. This method is based on the property of term symmetry, which will be defined and explored with respect to E-unification and E-completion procedures. Informally, term symmetry exists between two terms when one can be transformed into the other through variable renaming. By identifying and eliminating various forms of term symmetry which arise between syntactic structures such as, reductions, critical pairs, subterms, and unifiers, it is possible to derive an E-completion procedure that produces the same results without processing these symmetric redundancies.

1 Introduction

The concept of term symmetry is based on the realization that variable names used in a term are just symbols acting as placeholders for actual variables, and mapping those symbols to a different set of symbols will not change any aspect of the term, other than the variable names. This is the same idea that permits variables to be renamed in order to assure that terms involved in unification are variable-name disjoint.

Throughout let E be an equational theory. The congruence relation on terms is denoted by $s \overset{}{\underset{E}{=}} t$. The results of this paper were achieved through the development of of a flexible E-unification algorithm which is able to process pairs of terms which may contain any combination of null-E (null equational theory), C (Commutative), AC (Associative-Commutative), and ACI (Associative-Commutative with Identity) operators [*Ye*85], [*St*85]. Terms are represented as trees following the notation used by [*PS*81] or [*HO*80].

Definition 1.1: A *set of variable renaming substitutions* or a *variable renaming* is a set of substitutions, $\sigma = \{x_1 \leftarrow y_1, \ \ldots, \ x_n \leftarrow y_n\}$, which is a one-to-one, onto mapping from the set of variables $\{x_1, \ \ldots, \ x_n\}$ to the set of variables $\{y_1, \ \ldots, \ y_n\}$. Any substitution , $x_i \leftarrow y_i$, such that $x_i = y_i$ is an identity substitution and may be dropped from σ. The *identity variable renaming* is the empty set, $\{\}$. The application of a variable renaming, σ, to a syntactic entity, t, is written as t^σ.

Definition 1.2: Two terms, s and t, are *symmetric by* σ, written as $s \underset{\sigma}{\approx} t$, if there exists a (possibly empty) variable renaming from **vars**(s) (i.e., the set of all variables occurring in s) to **vars**(t), $\sigma = \{x_1 \leftarrow y_1, \ \ldots, \ x_n \leftarrow y_n\}$, and its inverse, $\sigma^{-1} = \{y_1 \leftarrow x_1, \ \ldots, \ y_n \leftarrow x_n\}$, such that $s^\sigma \underset{E}{=} t$ and $s \underset{E}{=} t^{\sigma^{-1}}$. Such a variable renaming is said to be a *symmetry* of s and t. Two terms for which no symmetry exists are *asymmetric*. Note that if σ is empty then $s \underset{E}{=} t$. Also, note that if s and t are variable disjoint, as is usually the case, then σ is a match between s and t.

Example 1.1: Let $+$ be a commutative operator (C, AC, or ACI). The two terms $s = +(x_1, \ x_1, \ x_2, \ x_3)$ and $t = +(y_1, \ y_2, \ y_2, \ y_3)$ are symmetric by the variable renamings $\sigma_1 = (x_1 \leftarrow y_2,\ x_2 \leftarrow y_1,\ x_3 \leftarrow y_3)$ and $\sigma_2 = (x_1 \leftarrow y_2,\ x_2 \leftarrow y_3,\ x_3 \leftarrow y_1)$.

Another form of term symmetry that is of interest is the symmetry which can exist within a single term. Symmetry within a term is a consequence of the presence of commutative operators.

Definition 1.3: A term, s, is *self-symmetric by* σ, written as $s \underset{\sigma}{\approx} s$, if there exists a variable renaming from **vars**(s) to **vars**(s), $\sigma = \{x_1 \leftarrow y_1, \ldots, x_n \leftarrow y_n\}$, such that $s^\sigma \underset{E}{=} s$. Such a variable renaming is said to be a *self-symmetry* of term s.

All terms are self-symmetric by the identity variable renaming. Since self-symmetry is a consequence of commutativity, a self-symmetry other than that described by the identity variable renaming can only exist if the term contains one or more commutative operators.

Example 1.2: Let + be a commutative operator (C, AC, or ACI). Then the term $s = +(x_1, x_1, x_2, x_3, x_4)$ is self-symmetric by the variable renamings
$\sigma_1 = \{x_2 \leftarrow x_3,\ x_3 \leftarrow x_2\}$,
$\sigma_2 = \{x_2 \leftarrow x_4,\ x_4 \leftarrow x_2\}$,
$\sigma_3 = \{x_3 \leftarrow x_4,\ x_4 \leftarrow x_3\}$,
$\sigma_4 = \{x_2 \leftarrow x_3,\ x_3 \leftarrow x_4,\ x_4 \leftarrow x_2\}$,
$\sigma_5 = \{x_2 \leftarrow x_4,\ x_4 \leftarrow x_3,\ x_3 \leftarrow x_2\}$.

2 Term Symmetry in E-Unification and E-completion

There are four types of term symmetry which may be observed in an E-completion procedure: symmetric reductions in the set of reductions being completed by the procedure, symmetric critical pairs, symmetric subterms used in the formation of critical pairs, and symmetric unifiers produced during the formation of critical pairs. The nature of term symmetry suggests that these symmetric syntactic structures may be redundant, hence it should be possible to derive from the Peterson-Stickel [*PS*81] E-completion procedure an *asymmetric E-completion procedure* that produces the same results without processing symmetric redundancies. For a more general approach to E-completion see [*JK*86].

In this section we will outline the fundamental ideas necessary for the development of an asymmetric E-completion procedure. Due to space limitations, we provide the basic results without proofs, and refer the reader to [*Ma*88] for these proofs and a more detailed analysis. In order to accomplish this, two points must be proven: first, that symmetry between syntactic structures, such as reductions, critical pairs, subterms, and unifiers, can be detected, and second, that the processing of a set of pairwise symmetric syntactic structures can be replaced by the processing of any one member of the set without changing the results produced by the E-completion procedure.

One method for detecting symmetries between syntactic structures, though inefficient, is to generate all matches that exist between the structures. If one of the matches is a variable renaming, there exists a symmetry between the structures. A more efficient algorithm for symmetry detection will be presented in the full paper.

2.1 Symmetric Reductions

A reduction, $\lambda \rightarrow \rho$, is an ordered pair of the terms λ and ρ. Two reductions, $\lambda_1 \rightarrow \rho_1$ and $\lambda_2 \rightarrow \rho_1$, are *symmetric reductions* if there exists a variable renaming, σ, such that $\lambda_1 \underset{\sigma}{\approx} \lambda_2$ and $\rho_1 \underset{\sigma}{\approx} \rho_2$. The redundancies introduced into the E-completion procedure by reduction symmetry are removed by the process of inter-reduction simplification.

Inter-reduction simplification is an integral part of the Peterson-Stickel E-completion procedure. When a new reduction is added to a set of reductions being completed, the two component terms of each reduction in the set are reduced to terminal form using the other reductions in the set. Any reduction reduced to an identity is discarded to preserve the finite termination property. If it reduces to an identity, then any information carried by the reduction must be embodied within the remainder of the set.

To demonstrate how this takes place, consider a member, $\lambda_1 \rightarrow \rho_1$, of the set of reductions that is symmetric by a variable renaming, σ, to a newly added reduction, $\lambda_2 \rightarrow \rho_2$. By the definition of reduction symmetry, $\lambda_1 \underset{\sigma}{\approx} \lambda_2$, or $\lambda_1^\sigma \underset{E}{=} \lambda_2$. The variable renaming is, therefore, a term match between λ_1 and λ_2, so $\lambda_1 \rightarrow \rho_1$ can be used to rewrite $\lambda_2 \rightarrow \rho_2$ into a new reduction, $\sigma(\rho_1) \rightarrow \rho_2$. But another consequence of the symmetry of the two reductions by σ is that $\rho_1 \underset{\sigma}{\approx} \rho_2$, or $\rho_1^\sigma = \sigma(\rho_1) \underset{E}{=} \rho_2$. Therefore, the new reduction is reduced to an identity and is discarded. Thus, the removal of reduction symmetry already takes place in the E-completion procedure as part of the inter-reduction simplification process.

Example 2.1.1 Let the set of reductions at some point in an execution of the E-completion procedure be the reductions describing an Abelian group,

r_1: $x + (-x) \rightarrow 0$,

r_2: $-(-x) \rightarrow x$, and
r_3: $-(x+y) \rightarrow (-x) + (-y)$,
such that $+$ is an ACI operator and $-$ is a null-E operator. Let
r_4: $y + (-y) \rightarrow 0$
be a reduction newly added to the set of reductions. It is the case that $r_1 \underset{\sigma}{\approx} r_4$ by $\sigma = \{x \leftarrow y\}$. Thus $\sigma(x + (-x)) = y + (-y)$, and the left-hand side of r_4 can be replaced by $\sigma(0)$, or 0. The reduced form of r_4 is $0 \rightarrow 0$, which is an identity and must be removed from the set of reductions.

2.2 Symmetric Critical Pairs

A critical pair, $< s,\ t >$, is an unordered pair of the terms s and t which are reduced to terminal form and then compared as part of the E-completion process. Two critical pairs, $< s_1,\ t_1 >$ and $< s_2,\ t_2 >$, are *symmetric critical pairs*, written as $< s_1,\ t_1 > \underset{\sigma}{\approx} < s_2,\ t_2 >$, if there exists a variable renaming, σ, such that $s_1 \underset{\sigma}{\approx} s_2$ and $t_1 \underset{\sigma}{\approx} t_2$, or $s_1 \underset{\sigma}{\approx} t_2$ and $t_1 \underset{\sigma}{\approx} s_2$. Without loss of generality, we shall assume the former for the duration of this discussion.

Critical pair symmetry is the lowest level of term symmetry in the E-completion procedure, that is, most term symmetries between reductions, subterms used in forming critical pairs, or unifiers will ultimately show up in the form of symmetric critical pairs. Removal of the other three types of term symmetry will result in the elimination of most, but not all, symmetric critical pairs.

In order to eradicate the remaining symmetric critical pairs, and to lay a foundation for use in proving that symmetric subterms and unifiers can be removed, it must be shown that discarding symmetric critical pairs will not change the results of the E-completion process. We shall begin by establishing some basic facts about the terminal forms of terms and critical pairs. Here, $t{\downarrow}^R$ denotes the result of reducing t by R to an irreducible or terminal form (i.e. a term which cannot be rewritten by any reduction in the set of reductions R).

Lemma 2.2.1: If s and t are terms and R is a set of reductions such that $s \underset{\sigma}{\approx} t$, then $(\forall s{\downarrow}^R)\,(\exists t{\downarrow}^R)\ s{\downarrow}^R \underset{\sigma}{\approx} t{\downarrow}^R$.

Lemma 2.2.2: If cp_1 and cp_2 are critical pairs such that $cp_1 \underset{\sigma}{\approx} cp_2$, then $(\forall cp_1{\downarrow}^R)\,(\exists cp_2{\downarrow}^R)\ cp_1{\downarrow}^R \underset{\sigma}{\approx} cp_2{\downarrow}^R$.

If two symmetric critical pairs truly represent redundant information, then it will be possible to prove that either one of them is sufficient for the proper operation of the E-completion procedure.

Lemma 2.2.3: If cp_1 and cp_2 are critical pairs such that $cp_1 \underset{\sigma}{\approx} cp_2$, then either cp_1 or cp_2 may be discarded without changing the results produced by the E-completion procedure.

This result may be generalized to deal with a set of symmetric reductions, rather than just a pair.

Theorem 2.2.1: A set of pairwise symmetric critical pairs encountered during the E-completion process may be replaced by any single member of that set without affecting the results of the process.

2.3 Symmetric Unifiers

As shown in the previous section, symmetric critical pairs may be discarded without affecting the results of the E-completion procedure. However, creating critical pairs which are then thrown out is a waste of processing time: Unifiers must be generated and applied to form these unneeded critical pairs. A better approach is to search for symmetric redundancies and to remove them from the components from which the critical pairs are built before much processing effort has been expended. One of the components that can be examined for term symmetry is the unifier associated with each critical pair. We would like to show that discarding symmetric unifiers has no effect on the results of the E-completion procedure. In order to prove this, it must be shown that symmetric unifiers produce symmetric critical pairs.

Definition 2.3.1: Let s and s' be terms. Assume, without loss of generality, that s and s' are variable disjoint. Two unifiers, $\theta_1,\ \theta_2 \in \mathbf{csu}(s,\ s')$, are *symmetric unifiers*, written as $\theta_1 \underset{\sigma}{\approx} \theta_2$, if there exists a variable renaming, σ, such that $\theta_1^{\sigma} \underset{E}{=} \theta_2$, and, for all terms, t, to which θ_1 and θ_2 will be applied, $t \underset{\sigma}{\approx} t$ and $\theta_1(t) \underset{\sigma}{\approx} \theta_2(t)$.

The definition of symmetric unifiers is more complicated than those of symmetric critical pairs and symmetric terms. In fact, the final condition of the definition--that is, the requirement that, for all terms t to which the unifiers will be applied, $\theta_1(t) \underset{\sigma}{\approx} \theta_2(t)$--seems to be self-defeating: Checking this condition for a given value of σ requires the application of θ_1 and θ_2 to a term, which is exactly the process that detecting and discarding symmetric unifiers is supposed to eliminate. However, there is a way to show that any variable renaming that meets the first two conditions of the definition will meet the third condition. The following theorem is obtained.

Theorem 2.3.1: Let $\lambda_1 \to \rho_1$ and $\lambda_2 \to \rho_2$ be reductions. A pairwise symmetric subset of **csu**(λ_1/i, λ_2), for $i \in$ **sdom**(λ_1), encountered during the E-completion process may be replaced by any single member of that set without affecting the results of the process.

2.4 Symmetric Subterms

Another component of critical pair formation that can be examined for term symmetry is the subterm chosen from the left-hand side of a reduction. It is easily shown that if s and t are terms and $r = \lambda \to \rho$ is a reduction, such that $s \underset{\sigma}{\approx} t$ and $s \underset{r}{\to} s'$, then $t \underset{r}{\to} t'$ such that $s' \underset{\sigma}{\approx} t'$. Since σ is merely a variable renaming, it follows that there must exist an $i \in$ **dom**(s) and a $j \in$ **dom**(t) such that $(s/i)^\sigma \underset{E}{=} t/j$, s/i matches λ by θ_i, t/j matches λ by θ_j, $s' = s[i \leftarrow \theta_i(\rho)]$, and $t' = t[j \leftarrow \theta_j(\rho)]$.

Now consider the case of $s \underset{\sigma}{\approx} s$, such that $(s/i)^\sigma \underset{E}{=} s/j$ and $i \neq j$, for some $i, j \in$ **dom**(s). If s/i matches λ by θ_i and s/j matches λ by θ_j, then is it also true that $s[i \leftarrow \theta_i(\rho)] \underset{\sigma}{\approx} s[j \leftarrow \theta_j(\rho)]$? If s/i and s/j are rooted at different depths in the tree representation of s, the two subterms cannot be considered symmetric. They are also not symmetric if they are sibling operands of a common non-commutative operator. If s/i and s/j are in distinct subtrees of s, then they can only be symmetric if the subtrees in which they appear are symmetric. Thus, the determination of symmetry is pushed upward in the tree to the level at which the two subtrees have a common parent node, and once again becomes a matter of determining the symmetry of sibling operands. This leads to a definition of symmetric subterms.

Definition 2.4.1: Let s be a term. Two subterms, s/i and s/j, are *symmetric subterms of s*, written as $s/i \underset{\sigma}{\approx} s/j$, if there exists a variable renaming σ such that $(s/i)^\sigma \underset{E}{=} s/j$, $s \underset{\sigma}{\approx} s$, and s/i and s/j are sibling operands of a common commutative (C, AC, or ACI) operator.

This definition must be modified slightly to be used with subterms of the left-hand side of a reduction. If $r = \lambda \to \rho$ is a reduction, then two subterms λ/i and λ/j are symmetric by σ if $(\lambda/i)^\sigma \underset{E}{=} \lambda/j$, $r \underset{\sigma}{\approx} r$, and λ/i and λ/j are sibling operators of a common commutative operator. The reason that $r \underset{\sigma}{\approx} r$ is required in place of $\lambda \underset{\sigma}{\approx} \lambda$ is that we want to show that symmetric subterms of λ produce symmetric critical pairs, and both λ and ρ are used in forming critical pairs. Hence, we obtain the following theorem.

Theorem 2.4.1: Let $\lambda_1 \to \rho_1$ and $\lambda_2 \to \rho_2$ be reductions. The processing of a set of pairwise symmetric subterms of λ_1 encountered during the E-completion process may be replaced by that of any single member of the set without affecting the results of the process.

2.5 A Term Symmetry Decision Algorithm

The main algorithm developed in this section is a decision procedure for the symmetry of a pair of terms composed of commutative operators, null-E operators, constants, and variables. It can also be used to decide the symmetry of terms involving AC and ACI operators if those terms have been simplified to normal form, that is, the terms have been flattened and have had all identities removed through simplification. The full details and psuedocode for the algorithms which follow can be found in [*Ma*88].

The term symmetry decision algorithm is similar in concept to the tree isomorphism decision algorithm presented by Aho, Hopcroft, and Ullman [*AH*74]. Their algorithm ignores all node labels in its operation. Unfortunately, this fact makes it inappropriate for use in deciding term symmetry, because for terms to be symmetric, constants must map onto identical constants and variables must map onto variables. An extension of the tree isomorphism decision algorithm is also suggested by Aho et al. to handle node labels. However, it, too, cannot be used to decide term symmetry, since the extension requires that variables map onto identical variables. In addition, neither of these algorithms consider the possible presence of null-E operators along with the commutative operators in the tree.

We briefly describe an algorithm Symmetric? which takes two terms, t_1 and t_2, and returns a symmetry, σ, if the terms are symmetric. Otherwise, it returns a value of FALSE. The actual implementation of this algorithm can be made more efficient by the application of constraints. For example, comparing the sizes of **vars**(t_1) and **vars**(t_2) before calling Build-Term-Bag could save unnecessary processing, since a difference in these sizes means that t_1 and t_2 are definitely not symmetric.

The terms input to Symmetric? are passed successively into the function Build-Term-Bag. This function constructs a bag *tb*, or multiset, of terms from its input parameter, *Term*. The term bag contains exactly one new term for each distinct variable, $x_i \in$ **vars**(*Term*). This new term is a copy of *Term* in which all occurrences of x_i have been replaced by the constant c_1, and all other variable occurrences have been replaced by the constant c_2. These are *new* constants, that is, c_1 and c_2 do not appear in t_1 or t_2.

Associated with each new term is x_i, the variable that was replaced by c_1. If *Term* is ground, that is, contains no variables, then the term bag returned is empty.

Once the term bags for t_1 and t_2 have been constructed, they are compared to decide whether or not the two input terms are symmetric. If the term bags are both empty, that is, both t_1 and t_2 are ground, then t_1 and t_2 are each sorted with respect to their commutative operators, that is, only the operands of commutative operators are sorted. Then the sorted terms are compared. If they are equal, then t_1 and t_2 are symmetric by the identity symmetry, $\sigma = \{\}$. If unequal, the two terms are not symmetric, and a value of FALSE is returned.

On the other hand, if either of the term bags is non-empty, then each term in both term bags is sorted with respect to commutativity, and then each term bag is sorted. If the two sorted term bags are equal, then there is a one-to-one, onto mapping from each term in tb_1 to an equivalent term in tb_2. A term bag contains exactly one term for each variable in the term from which it was constructed, and each variable is associated with exactly one member of its term bag. Thus, the mapping from tb_1 to tb_2 can, and is, used to construct a one-to-one, onto mapping from **vars**(t_1) to **vars**(t_2). This mapping is returned as a symmetry of t_1 and t_2. If the two sorted term bags are not equal, then t_1 and t_2 are not symmetric, and a value of FALSE is returned.

It can be shown that Symmetric?(t_1, t_2) is an algorithm. There are a finite number of distinct variables in each of t_1 and t_2, thus Build-Term-Bag will halt for each. Also, since Symmetric? contains no loops, it will halt. The worst-case time complexity for this algorithm is $O(n^2 \log n)$.

Theorem 6.4: The function Symmetric?(t_1, t_2) returns a symmetry, σ, iff $t_1 \underset{\sigma}{\approx} t_2$.

By extending the basic term symmetry decision algorithm, we have developed algorithms which will prune the complete set of unifiers to an asymmetric complete set of unifiers and prune the strict domain of a term to an asymmetric strict domain.

3 Implementation Results

The ideas described in this paper have been implemented in Common Lisp and run on a variety of machines. Test runs were made for four cases: an abelian group, a commutative ring with identity, a group homomorphism, and a distributive lattice with identity using AC and ACI unification. There were six test runs in each group, based on different combinations of the levels of term symmetry removed from processing:

level 1--symmetric reductions,

levels 1 and 2--symmetric reductions and subterms,

levels 1 and 3--symmetric reductions and unifiers,

levels 1 and 4--symmetric reductions and critical pairs,

levels 1, 2, and 3, and

levels 1, 2, 3, and 4.

The removal of symmetric reductions was included in every test since, as discussed earlier, it is an integral part of the standard Peterson-Stickel E-completion procedure.

The results of the AC and ACI test groups for the abelian group, commutative ring with identity, group homomorphism, and distributive lattice with identity are similar. In each case, removing symmetric subterms and/or symmetric unifiers (levels 1 and 2, levels 1 and 3, and levels 1, 2, and 3) did not have a great impact on the number of critical pairs produced; that is, there were not many symmetric subterms or unifiers found. The total run times of these three tests are almost identical to that of the standard E-completion procedure. Thus, the run time saved by removing these symmetric redundancies was evidently consumed by the process of checking every subterm and/or unifier for symmetry.

The removal of symmetric critical pairs (levels 1 and 4, and levels 1, 2, 3, and 4) was, however, a different matter. The elimination of this type of term symmetry resulted in a significant reduction in the number of critical pairs (13% to 28%) and a corresponding reduction in the total run time (14% to 21%). The tests in which all four types of term symmetry were eliminated resulted in the same or fewer critical pairs retained than did the removal of just symmetric reductions and critical pairs, but once again, the overhead of removing symmetric subterms and unifiers destroyed any potential savings in total run time.

As was stated earlier, the goal of this research was to develop a method of significantly reducing the processing needed to complete an incomplete set of reductions. We have been modestly successful in reaching this goal. The savings in processing time resulting from the removal of term symmetries were not as significant as we had hoped for. We had expected a sizable percentage of unifiers to be symmetric, but this was not so. In fact, the removal of symmetric unifiers or symmetric subterms generally resulted in a slower run time than with the symmetries left intact. The best method, in general, turned out to be the removal of symmetric critical pairs after their formation.

References

[*AH*74] Aho, A., Hopcroft, J., and Ullman, J. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, Reading, MA.

[*CL*88] Christian, J., and Lincoln, P. (1988) "Adventures in associative-commutative unification." Technical Report ACA-ST-275-87, Microelectronics and Computer Technology Corp., Austin, TX.

[*Fa*84] Fages, F. (1984). "Associative-commutative unification." Proceedings of the Seventh International Conference on Automated Deduction, R. Shostak, ed., *Lecture Notes in Computer Science*, volume 170, Springer-Verlag, Berlin, West Germany, pp. 194-208.

[*HO*80] Huet, G., and Oppen, D. (1980) "Equations and rewrite rules: a survey." *Perspectives and Open Problems*, R. Book, ed., Academic Press, Orlando, FL.

[*Hu*78] Huet, G. (1978). "An algorithm to generate the basis of solutions to homogeneous linear diophantine equations." *Information Processing Letters*, volume 7, pp. 144-147.

[*JK*86] Jouannaud, J.-P., and Kirchner, H. (1986). "Completion of a set of rules modulo a set of equations." *SIAM Journal of Computing*, volume 15, pp. 1155-1194.

[*KM*86] Kapur, D., Musser, D., and Narendran, P. (1986). "Only prime superpositions need be considered in the Knuth-Bendix completion procedure." Technical Report, General Electric Research and Development Center, Schenectady, NY.

[*KB*70] Knuth, D., and Bendix, P. (1970). "Simple word problems in universal algebras." *Computational Problems in Abstract Algebras*, J. Leech, ed., Pergamon Press, Oxford, England, pp. 263-297.

[*La*87] Lankford, D. (1987). "Non-negative basis algorithms for linear equations with integer coefficients." Technical Report, Louisiana Tech University, Ruston, LA.

[*LS*76] Livesey, M., and Siekmann, J. (1976). "Unification of A+C-terms (bags) and A+C+I-terms (sets)." Technical Report, Universitat Karlsruhe, Karlsruhe, West Germany.

[*Ma*88] Mayfield, B. (1988). "The role fo term symmetry in E-unification and E-completion." Ph. D. dissertation, University of Missouri-Rolla, Rolla, MO.

[*PS*81] Peterson, G., and Stickel, M. (1981) "Complete sets of reductions for some equational theories." *Journal of the Association for Computing Machinery*, volume 28, pp. 233-264.

[*Ro*65] Robinson, J.A. (1965). "A machine-oriented logic based on the resolution principle." *Journal of the Association for Computing Machinery*, volume 12, pp. 23-41.

[*Si*79] Siekmann, J. (1979). "Matching under commutativity." *Symbolic and Algebraic Computation*, Springer-Verlag, Berlin, West Germany, pp. 531-545.

[*St*75] Stickel, M. (1975). "A complete unification algorithm for associative-commutative functions." *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, Tbilisi, pp. 71-82.

[*Ye*85] Yelick, K. (1985). "Combining unification algorithms for confined regular equational theories." *Conference on Rewriting Techniques and Applications*, J. Jouannaud, ed., *Lecture Notes in Computer Science*, volume 202, Springer-Verlag, Berlin, West Germany, pp. 365-380.