A Practical Ordering for AC Rewrite Systems

Stephen J. Garland
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
garland@lcs.mit.edu

Deepak Kapur
Department of Computer Science
State University of New York
Albany, NY 12222
kapur@cs.albany.edu

Abstract

The AC-DSMPOS ordering orients a large class of associative-commutative (AC) equational systems automatically and efficiently into terminating rewrite systems. This ordering enhances DSMPOS, the "dynamically suggested monotonic path ordering with status" based on RPOS and used since 1985 in LP (the Larch Prover) with features developed by Kapur, Sivakumar and Zhang in 1990 to handle AC functions. The new ordering, which has been implemented in LP, automatically generates and uses precedence suggestions to orient AC equational systems incrementally into terminating rewrite systems. Experimental evidence indicates that AC-DSMPOS adequately and efficiently handles many examples arising in practical applications involving distributed algorithms, hardware and software descriptions and specifications, and equational axiomatizations of well-known algebraic structures and abstract data types that involve AC functions. (It also orients the distributivity axiom in the right direction.)

1 Introduction and Motivation

Orienting equations into terminating rewrite rules has been a challenging problem theoretically as well as pragmatically. Theoretically, even though the termination problem for rewrite systems is undecidable (even for a single rule system), there are many heuristics and algorithms for showing termination of a large class of term rewriting systems (see [5] for a survey of termination problems as well as [18] for a detailed comparison of the expressive power and efficiency of different orderings). Pragmatically, users of rewrite-rule based provers such as LP (the Larch Prover), *RRL* (the Rewrite Rule Laboratory), and REVEAL, must confront the problem of orienting input equations, as well as new equations generated by applications of inference rules, into terminating rewrite rules. To help users determine appropriate orientations, LP automatically generates precedence suggestions for orienting equations using a variant DSMPOS of the recursive path ordering with status (RPOS) [10]. LP's algorithm attempts to preserve the orientation with which an equation was entered by the user, but overrides this orientation when necessary to preserve termination. This feature is particularly attractive to users of LP, who are not familiar with rewrite rule theory, in general, or with termination orderings, in particular.

This paper addresses the problem of extending LP to handle termination of equations that contain associative-commutative (AC) function symbols. Many problem formulations involve AC function symbols, but the commutative property cannot be expressed by a terminating rewrite rule. LP and other rewrite-rule based provers have long reasoned about such problems using associative-commutative rewriting (simplification) and unification, as suggested in [14, 16].

However, developing termination orderings for associative-commutative rewrite systems has been a challenge. This has been an active area of research, and a number of proposals exist in the literature. Orderings based on associating polynomial interpretations with function symbols [13, 3] are not sufficiently expressive. A more serious problem with these orderings is that the user has to design appropriate polynomial interpretations (see however [20], where heuristics for designing polynomial interpretations are discussed). In fact, LP provides an implementation of polynomial orderings, but these orderings are not widely used because of this problem. Instead, most LP users use DSMPOS to orient equations, even when they contain AC operators, with the hope that the resulting rewrite system will still terminate. Surprisingly, this has worked well: users who rely on DSMPOS have almost never encountered LP's warning for a possibly infinite rewrite sequence. One purpose of this paper is to explain why this is so.

A number of termination orderings based on paths have been proposed in the literature for AC rewrite systems [2, 8, 1, 4]. Until recently, these orderings were hard to implement, or they restricted the number of AC operators in a term or the precedence of AC operators with respect to other operators. Most promising among path orderings are one by Kapur, Sivakumar and Zhang [12] in 1990, henceforth called the KSZ ordering, and another by Rubio and Nieuwenhuis [17] in 1993, both of which extend RPOS. Rubio and Neiuwenhuis's ordering is attractive because it is total on (equivalent) ground terms, assuming that the precedence relation on function symbols is total; however, a major weakness of this ordering is that it orients the distributivity axiom in the reverse direction. More importantly, it has been unclear how precedence suggestions can be easily and efficiently generated for these orderings following the approach used in [10] to aid users in orienting equations with AC function symbols into terminating rewrite rules.

Using an idea in the KSZ ordering, we have developed an ordering AC-DSMPOS for AC rewrite systems that almost behaves like RPOS. The ordering has been implemented in LP using a max-flow algorithm. Most interestingly, LP's algorithm for generating precedence and status suggestions for RPOS has been extended to equations with AC function symbols. Even though the AC-DSMPOS cannot handle as many equational systems as the KSZ ordering, its expressive power is adequate to handle many interesting systems. For example, it handles all the examples in the LP library, many nontrivial specifications of and proofs about distributed algorithms, and many equational axiomatizations of well-known algebraic structures and abstract data types. As we will show, the extended algorithm for generating precedences performs well. The time needed to check for termination of AC rewrite system is comparable to the time needed to orient equations using RPOS, and it provides the additional guarantee that the rewrite system is terminating.

Outline of the paper can go here.

2 An RPOS-based AC-Path Ordering

The AC-DSMPOS ordering extends RPOS, the recursive path ordering with status.

2.1 Preliminaries

Let F be a set of function symbols, and let $\mathcal{F}_{AC}$ be the subset of F consisting of all AC function symbols in F. Given an AC function symbol f and a multiset $S = \{\!\{s_1, \ldots, s_m\}\!\}$ of terms, by $f(S)$ we mean any of the AC-equivalent terms $f(s_1, \cdots, s_m)$.

2.2 Definitions

Terms are assumed to be *flattened*: when $*$ is AC, terms like $(a*b)*c$ are rewritten as $*(a,b,c)$. In other words, if an argument of an AC function symbol has the same AC function as its root, then that argument is replaced by its arguments. Formally,

Definition 2.1 *The flattened form $\overline{t}$ of a term t is defined below.*

$$\overline{t} = \begin{cases} x & \text{if } t = x, \text{ a variable} \\ f(\overline{t_1}, \ldots, \overline{t_n}) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \notin \mathcal{F}_{AC} \\ f(T_1 \cup \cdots \cup T_n) & \text{if } t = f(t_1, \ldots, t_n), f \in \mathcal{F}_{AC} \text{ and } T_i = \begin{cases} \{\!\{s_1, \ldots, s_m\}\!\} & \text{if } \overline{t_i} = f(s_1, \ldots, s_m) \\ \{\!\{\overline{t_i}\}\!\} & \text{otherwise} \end{cases} \end{cases}$$

Let $t \underset{\sim}{\succ}_{ac} s$ stand for $t \sim_{ac} s$ or $t \succ_{ac} s$.

Definition 2.2 *A non-variable term $t \succ_{ac} x$ (a variable) iff x occurs in t. Otherwise, $t = f(t_1, \ldots, t_n) \succ_{ac} g(s_1, \ldots, s_m) = s$, iff one of the following holds.*

1. *$f \succ_f g$, and $t \succ_{ac} s_j$ for all j $(1 \le j \le m)$.*
2. *$f \sim_f g$, $f, g \notin \mathcal{F}_{AC}$, f and g have multiset status, and $\{\!\{t_1, \ldots, t_n\}\!\} \succ_{mul} \{\!\{s_1, \ldots, s_m\}\!\}$.*
3. *$f \sim_f g$, $f, g \notin \mathcal{F}_{AC}$, f and g have left-to-right (similarly, right-to-left) status, $(t_1, \ldots, t_n) \succ_{lex} (s_1, \ldots, s_m)$, and $t \succ_{ac} s_j$ for all j, $(1 \le j \le m)$.*
4. *$t_i \underset{\sim}{\succ}_{ac} s$ for some i $(1 \le i \le n)$.*
5. *$f \sim_f g$, $f, g \in \mathcal{F}_{AC}$, one of the following two cases holds:*
   (a) *$\overline{t} = f(T_1)$, $\overline{s} = g(S_1)$, $T = T_1 - S_1, S = S_1 - T_1$, either*
   - *$S = \emptyset$ and $T \neq \emptyset$, or*
   - *there is a $t_i \in T$ and $s_j \in S$ such that $t_i \succ s_j$, and*
     - *$root(t_i) \succ f$ and $f(T) \succ g(S - \{s_j\})$, or*
     - *$(root(t) \not\succ f$ or for all $t'_i \in T - \{t_i\}$, $t'_i \not\succ s_j)$ and $f(T - \{t_i\}) \succ g(S - \{s_j\})$;*

   (b) *there is a proper subterm t'_i of some $t_i \in T$, which is at position p in $\overline{t}$, such that $\overline{t}[t'_i]|p \underset{\sim}{\succ}_{ac} s$.*

Above, the multiset and lexicographic extensions of $\succ_{ac}$ and $\underset{\sim}{\succ}_{ac}$ are mutually recursively defined in the usual way. Also, the definition of $\sim_{ac}$ on flattened terms is defined in the same way as in RPOS

In the above definition, cases 1, 2, 3, and 4 are identical to the corresponding cases in RPOS. Case 5 is the new one, which compares two terms that begin with the same (or equivalent in precedence) AC operators. We first eliminate common arguments. Then we check whether the first multiset properly contains the other multiset. Otherwise, there are two subcases. In the first case, we look for an element of the multiset T with a bigger root symbol than f so that this element can be repeatedly use to take care of smaller elements in multiset S just as in RPOS. Elements of T whose root symbol is not bigger than f are then used to take care of remaining smaller elements in S, with exactly one element from T taking care of exactly one element in S. In the second case, a proper subterm of an element in T replaces the element, and the resulting term is used for comparison instead.

2.3 Examples

Let us consider the distributivity axiom: $t = x * (y + z)$ with $s = (x * y) + (x * z)$, where $+, *$ are AC. If $* \succ_f +$, then just like in RPOS, for t to be $\succ_{ac}$ than s, t must be $\succ_{ac}$ than both $x * y$ as well as $x * z$ which is the case. If $+ \succ_f *$, then t cannot be $\succ_{ac}$ than s; for s to be $\succ_{ac}$ than t, s must be $\succ_{ac}$ than both x and $y + z$ which is indeed the case.

As the reader would have noticed, the ordering works in the same way as RPOS. Here is another example illustrating the relationship with RPOS: $t = f(g(a, b), b)$ and $s = g(f(b, b), f(g(b, b), a))$. For t to be $\succ_{ac}$ than s, $f \succ_f g$; further, t must be greater than both $f(b, b)$ and $f(g(b, b), a)$, which is possible provided $b \succ_f a$. For $s \succ_{ac} t$, $g \succ_f f$ and s must be greater than $g(a, b)$ as well as b, which is indeed the case.

As is well-known, RPOS cannot be used to detect termination of AC rwrite systems. Consider $t_1 = f(a, b), s_1 = g(a, b)$ and $t_2 = f(x, g(y, z)), s_2 = f(x, f(y, z))$, where f is an AC function symbol. If f is made $\succ_f g$ to make $t_1 \succ_{ac} s_1$, then $t_2 \not\succ_{ac} s_2$ since the subterm $g(y, z)$ can either take care of y or z in the multiset $\{\!\{x, y, z\}\!\}$, but not both since the root g of $g(y, z)$ is $\not\succ_f f$. RPOS would declare $t_2 \succ_{ac} \overline{s_2}$ under the precedence $f \succ_f g$. In the proposed ordering, s_2 is also $\not\succ_{ac} t_2$. In contrast, KSZ's ordering would make $s_2 \succ_{ac} t_2$ because of partitioning.

The following examples from [12] further illustrate the difference between the expressive power of the proposed ordering and KSZ's ordering. They are also helpful in understanding the relationship and difference in the two orderings.

Let $t_1 = f(g(x), g(y))$, $t_2 = f(x, x, y, y)$, $t_3 = f(c(x, y), c(x, y))$, where f is AC and $g \succ_f f \succ_f c$. Since the subterm $g(x)$ in t_1 has the root $\succ_f f$, it can be repeatedly used to take care of 2 occurrences of x in t_2; similarly for $g(y)$, thus $t_1 \succ_{ac} t_2$. But t_2 is not bigger than t_3; also note that t_3 is not bigger than t_2 because the subterm $c(x, y)$ can only take care of one of x and y. Similarly, t_1 is not bigger than t_3.

Consider $t_4 = f(c(g(x), x), c(f(y, y), y))$. Using the precedence relation of the previous example, $t_4 \succ_{ac} t_2$ because subterms $g(x)$ and $f(y, y)$ can be elevated to give $t_4' = f(g(x), y, y)$ which is $\succ_{ac} t_2$, thus making $t_4 \succ_{ac} t_2$.

In $t = f(a, c), s = f(b, b, b)$ with $a \succ_f b \succ_f c$, t can be made $\succ_{ac} s$ by making $a \succ_f f$ so that a can take care of both occurrences of b in s. We cannot make $s \succ_{ac} t$, with the above precedence relation.

2.4 Proof of Well-foundedness

In this subsection, we give a proof of well-foundedness of the ordering. We also show that the ordering is transitive, preserved under substitutions as well as monotonic, so that it can be used for showing termination of the rewriting relation induced by AC rewrite system. The proof of well-foundedness is done by relating this ordering to KSZ's ordering, in particular, showing that if $s \succ_{ac} t$ using a precedence relation $\succ_f$, then $s \succ_{KSZ} t$ also.

2.4.1 KSZ's Ordering

We reproduce the definition of the ordering defined in [12].

Definition 2.3 *A non-variable term $t \succ_{KSZ} x$ (a variable) iff x occurs in t. Otherwise, $t = f(t_1, \ldots, t_n) \succ_{KSZ} g(s_1, \ldots, s_m) = s$, iff one of the following holds.*

1. $f \succ_f g$, *and* $t \succ_{KSZ} s_j$ *for all* j $(1 \leq j \leq m)$.
2. $f \sim_f g$, f *and* g *have multiset status, and* $\{\!\{t_1, \ldots, t_n\}\!\} \succ_{mul} \{\!\{s_1, \ldots, s_m\}\!\}$.
3. $f \sim_f g$, f *and* g *have left-to-right (similarly, right-to-left) status,* $(t_1, \ldots, t_n) \succ_{lex} (s_1, \ldots, s_m)$, *and* $t \succ_{KSZ} s_j$ *for all* j, $(1 \leq j \leq m)$.
4. $t_i \underset{\sim}{\succ}_{ac} s$ *for some* i $(1 \leq i \leq n)$, *where* $t_i \underset{\sim}{\succ}_{ac} s$ *if and only if* $t_i \sim_{ac} s$ *or* $t_i \succ_{KSZ} s$.
5. $f \sim_f g$, $f, g \in \mathcal{F}_{AC}$, $t = f(T)$, $s = g(S)$, $S' = S - T = \{\!\{s'_1, ..., s'_k\}\!\}$, *either*
   - $k = 0$ *and* $n > m$ *(i.e.* $S - T = \emptyset$ *and* $T - S \neq \emptyset$*), or*
   - $f(T - S) \Rightarrow^* f(T')$, *and* $T' = T_1 \cup \cdots \cup T_k$ *and for all* i $(1 \leq i \leq k)$ *either*
     - $T_i = \{u\}$ *and* $u \underset{\sim}{\succ}_{ac} s'_i$, *or*
     - $T_i = \{\!\{u_1, \ldots, u_l\}\!\}$ *and* $f(u_1, \ldots, u_l) \underset{\sim}{\succ}_{ac} s'_i$.

   Also, in this case, either $f(T - S) \Rightarrow^+ f(T')$, *or one of* $\underset{\sim}{\succ}_{ac}$ *is strict.*

The binary relation $\Rightarrow$ on terms with the same (or equivalent) AC operator f as root is defined below.

Definition 2.4 *Let* $f \in \mathcal{F}_{AC}$, $t = f(T)$ *be a term, where* $T = \{\!\{t_1, \ldots, t_n\}\!\}$. $t \Rightarrow f(T_1)$ *in one step iff there is* $t_i = g(r_1, \ldots, r_k)$ *in* T, $T_1 = T - \{\!\{t_i\}\!\} \cup T_2$, *and one of the following holds.*

1. **Pseudocopying** $g \succ_f f$, *and* $T_2 = \{\!\{gg(r_1, \ldots, r_k), \ldots, gg(r_1, \ldots, r_k)\}\!\}$ *for some finite number of pseudocopies* $gg(r_1, \ldots, r_k)$.
2. **Flattening** $g \sim_f f$, *and* $T_2 = \{\!\{r_1, \ldots, r_k\}\!\}$.
3. **Elevating** $g \not\underset{\sim}{\succ}_f f$, *and* $T_2 = \{r_j\}$ *for some* j.

The reader should have noticed considerable similarity between the above definition and the definition of our ordering; that is not surprising given that our ordering is an attempt to simplify KSZ's ordering. The key difference is partitioning the arguments of an AC function symbol into multi-subsets to jointly take care of subterms of an equivalent AC function symbol; trying all possible partitions and comparisons using them can be expensive especially if $t \not\succ_{ac} s$. An advantage of using partitioning is the flexibility and generality, in particular, because of pseudo-copying, different paths in different subterms can be used together to take care of a subterm. Some of the examples discussed in the previous subsection illustrated this difference.

In [12], it is proved:

Theorem 2.5 [12] $\succ_{KSZ}$ *is a reduction ordering, i.e. it is a simplification ordering (so is well-founded) and is stable under substitution.*

2.5 $\succ_{ac}$ is a subset of $\succ_{KSZ}$

The following theorem relates our ordering to the KSZ ordering, in particular, it proves that our ordering is a strict subset of KSZ ordering. Using the above theorem, the well-foundedness of our ordering follows.

Theorem 2.6 [12] *If $t \succ_{ac} s$, then $t \succ_{KSZ} s$.*

Proof. (Sketch). Assume $t \succ_{ac} s$. The proof is by induction on the sum of the sizes of t and s. The basis step is trivial.

In the induction step, if $t \succ_{ac} s$ because of any of the first four cases, then $t \succ_{KSZ} s$ also since each of these steps are in the definition of $\succ_{KSZ}$ also. The interesting case is that of 5 when the root of t is the same (or equivalent in precedence) as the root of s. In Step (a), corresponding to the first case, there is a case in the definition of $\succ_{KSZ}$.

In the second case, each element in T is included in its own partition; in addition, if the root of a subterm t_i is $\succ_f$ than the root of t, then include its pseudo-copy in every partition of T. Each partition is then used to take care of a single element in S in $\succ_{KSZ}$. The relation $\Rightarrow$ is used for making pseudo-copies and flattening.

If Step (b) is used, there is a corresponding step in the definition of $\succ_{KSZ}$ where elevation is done using $\Rightarrow$.

Properties similar to Lemmas 5.10, 5.11, and 5.12 in [12] for $\underset{\sim}{\succ}_{KSZ}$ can be proved for $\underset{\sim}{\succ}_{ac}$ also, which are useful in this proof and subsequent proofs. □

The irreflexivity of $\succ_{ac}$ follows from the above lemma and the fact that $\succ_{KSZ}$ is irreflexive.

Theorem 2.7 *$\succ_{ac}$ is transitive, i.e., for any terms t, s and r, if $t \underset{\sim}{\succ}_{ac} s$, $s \underset{\sim}{\succ}_{ac} r$, then $t \underset{\sim}{\succ}_{ac} r$.*

Proof. This proof is most involved. The argument given in section 5.2 in [12] are repeated to give a proof, which for the most part, involves doing a case analysis based on parts of the definition of $\succ_{ac}$ used to show $t \underset{\sim}{\succ}_{ac} s$ and $s \underset{\sim}{\succ}_{ac} r$. As stated above, properties similar to Lemmas 5.8-5.12 in [12] can be proved for $\succ_{ac}$, and are used in this proof.

Lemma 2.8 *$\succ_{ac}$ has the replacement property, i.e., if $t \succ_{ac} s$, then $f(...,t,...) \succ_{ac} f(...,s,...)$ for any function symbol f, where the missing arguments are assumed to be the same in both sides.*

Proof. If $f \notin \mathcal{F}_{AC}$, then the proof steps are the same as in RPOS, depending upon the status of f. If $f \in \mathcal{F}_{AC}$, then by case 5 of the definition, the property holds. □

Using the replacement property, by induction on the size of $u[\cdot]$, it follows:

Theorem 2.9 *$\succ_{ac}$ is monotonic, i.e., if $t \succ_{ac} s$ then $u[t] \succ_{ac} u[s]$ for any context $u[\cdot]$.*

Theorem 2.10 *$\succ_{ac}$ is stable under instantiation, i.e., $t \succ_{ac} s$ implies $t\sigma \succ_{ac} s\sigma$ for any substitution σ.*

Proof. By induction on the sum of sizes of s and t. The proof mimics the proof steps given in the proof of proposition 5.14 in [12]. The interesting case is 5, for which, an argument similar to the one in the proof of Theorem 2.6 applies. □

Theorem 2.11 *$\succ_{ac}$ is AC-compatible. That is, for any terms t, t_1, s, and s_1, if $t \sim^*_{ac} t_1$, $s \sim^*_{ac} s_1$, and $t \succ_{ac} s$, then $t_1 \succ_{ac} s_1$.*

Proof. Since $t \sim^*_{ac} t_1$ implies that t_1 can be obtaining from t by rearranging the arguments of AC function symbols, whatever elevation and flattening are done on t, can also be done on t_1. Similarly for s, s_1. Since no subterms can disappear, the steps used to show that $t \succ_{ac} s$ can be repeated on t_1, s_1 to show that $t_1 \succ_{ac} s_1$. □

3 Implementation in LP

References

[1] Bachmair, L. (1992): Associative-commutative reduction orderings. *Info. Proc. Letters.*

[2] Bachmair, L., and Plaisted, D.A. (1985): Termination orderings for associative-commutative rewriting systems. *J. Symbolic Computation,* 1, 329-349

[3] Ben Cherifa, A., and Lescanne, P. (1987): Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming,* 9, 2, 137-160.

[4] Delor, C., Puel, L. (1993): Extension of the associative path ordering to a chain of associative commutative symbols. In Kirchner, c. (ed.) Proc. of *5th Intl. Conf. on Rewrite Techniques and Applications (RTA-93)*, LNCS, Springer-Verlag, 389-404.

[5] Dershowitz, N. (1987): Termination of rewriting. *J. Symbolic Computation,* 3, 69-116.

[6] Garland, S., and Guttag, J. (1989): An overview of LP, the Larch Prover. In *Third International Conference on Rewriting Techniques and Applications*, pages 137–151, Chapel Hill, 1989. Springer-Verlag Lecture Notes in Computer Science 355.

[7] Garland, S., and Guttag, J. (1991): A guide to LP, the Larch Prover. Report 82, DEC Systems Research Center, Palo Alto, CA.

[8] Gnaeding, I., and Lescanne, P. (1986): Proving termination of associative-commutative rewriting systems by rewriting. Proc. of *8th Intl. Conf. on Automated Deduction (CADE-8)*, Oxford, LNCS 230 (ed. Siekmann), Springer Verlag, 52-60.

[9] Guttag, J., and Horning, J., editors (1993): *Larch: Languages and Tools for Formal Specification.* Texts and Monographs in Computer Science. Springer-Verlag, 1993 With Garland, S., Jones, D., Modet, A., and Wing, J.

[10] Detlefs, D. and Forgaard, R. (1985): A procedure for automatically proving the termination of a set of rewrite rules. In *Proc. 1st Int. Conf. on Rewriting Techniques and Applications*, Dijon, France, **LNCS 202**, 255–270.

[11] Kapur, D., and Zhang, H. (1995): An overview of Rewrite Rule Laboratory (RRL). *J. Computer and Mathematics with Applications,* 29, 2, 91-114.

[12] Kapur, D., Sivakumar, G. and Zhang, H. (1990): A new ordering for proving termination of AC-rewrite systems, Proc. of *10th Conf. on Foundations of Software Technology and Theoretical Computer Science*, Bangalore, India, Dec. 1990, Springer Verlag LNCS 472, 133–148. A revised version appeared in *J. Automated Reasoning,* 1995.

[13] Lankford, D.S. (1979): On proving term rewriting systems are noetherian. Memo MTP-3, Lousiana State University.

[14] Lankford, D.S., and Ballantyne, A.M. (1977): Decision procedures for simple equational theories with commutative-associative axioms: complete sets of commutative-associative reductions. Automatic Theorem Proving Project, Dept. of Math. and Computer Science, University of Texas, Austin, Texas, Report ATP-39.

[15] Narendran, P., and Rusinowitch, M.

[16] Peterson, G.L., and Stickel, M.E. (1981): Complete sets of reductions for some equational theories. *J. ACM*, 28, 2, 233-264.

[17] Rubio, A., Nieuwenhuis, R. (1993): A precedence-based total AC-compatible ordering. In Kirchner, C. (ed.) Proc. of *5th Intl. Conf. on Rewrite Techniques and Applications (RTA-93),* LNCS Springer-Verlag, 374-388.

[18] Steinbach, J. (1989): Extensions and comparisons of simplification orderings, in Proc. *3rd Intl. Conf. on Rewriting Techniques and Applications (RTA-89)*, Chapel Hill, NC, 434-448.

[19] Steinbach, J. (1989): Path and decomposition orderings for proving AC-termination. Seki-Report, SR-89-18, University of Kaiserslautern. See also "Improving associative path orderings," in: Proc. of *10th Intl. Conf. on Automated Deduction (CADE-10)*, Kaiserslautern, LNCS 449 (ed. Stickel), Springer Verlag, 411-425.

[20] Steinbach, J. (1991) Termination proofs of rewriting systems – Heuristics for generating polynomial orderings. SEKI Report SR-91-14 (SFB) University of Kaiserslautern.