ASSOCIATIVE-COMMUTATIVE REWRITING*

Nachum Dershowitz
Department of Computer Science
University of Illinois
Urbana, IL 61801

Jieh Hsiang
Department of Computer Science
State University of New York
Stony brook, NY 11794

N. Alan Josephson
Department of Computer Science
University of Illinois
Urbana, IL 01801

David A. Plaisted**
Computer Science Laboratory
SRI International
Menlo Park, CA 91025

Abstract

We are currently extending the rewrite system laboratory *REVE* to handle associative-commutative operators. In particular, we are incorporating a set of rules for Boolean algebra that provides a refutationally-complete theorem prover and a new programming paradigm. To that end, we describe methods for proving termination of associative-commutative systems.

1. Introduction

Term-rewriting systems have been widely used in formula-manipulation and theorem-proving systems. As programs, they have a very simple syntax and semantics based on equalities, and are declarative with no explicit control. In addition, canonical systems, i.e. systems that always rewrite a term to a unique normal form, are used as decision procedures for equational theories. (For a survey of the theory and application of rewrite systems, see Huet and Oppen [1980].)

Example: The following rewrite system differentiates an expression (Knuth [1968]):

$$\boxed{\begin{array}{rcl}
D_x x & \rightarrow & 1 \\
D_x a & \rightarrow & 0 \\
D_x(u+v) & \rightarrow & D_x u + D_x v \\
D_x(u-v) & \rightarrow & D_x u - D_x v \\
D_x(-u) & \rightarrow & -D_x u \\
D_x(uv) & \rightarrow & vD_x u + uD_x v \\
D_x(\frac{u}{v}) & \rightarrow & D_x \frac{u}{v} \quad uD_x \frac{v}{v^2} \\
D_x(\ln u) & \rightarrow & D_x \frac{u}{u} \\
D_x(u^v) & \rightarrow & vu^{v-1} D_x u + u^v (\ln u) D_x v
\end{array}}$$

where *u* and *v* are variables of the rewrite system and match any term, x is the symbol with respect to which an expression is differentiated, and *a* is any atomic symbol other than *z.*

Unfortunately, associativity and commutativity of functions cannot be handled by including these axioms as rules. Instead, special unification algorithms are used to take associativity and commutativity into account. Let *R* be a rewrite system for a set of terms *T(FUG),* constructed from associative-commutative function symbols *F* and regular symbols g. The equat ional theory *AC* consists of the axioms

$$f(u,v) \quad = \quad /(v.u)$$
$$f(u,f(v,w)) \quad - \quad f(f(u,v),w),$$

for each symbol *fEF.* A rule *l+r* in *R* is applied to a term *tET* if / matches any subterm *s* of t, under the assumption of associativity and commutativity of symbols in *F,* i.e. if there is a term *t'* that is-equal to t in the theory *AC,* and there is a substitution *a* for the variables of /, such that *(T(1)=«* for some subterm *s* of /' .

In the next section, we describe two applications of *AC*-rewriting in the context of the *REVE* term rewriting environment (Lescaune [1982]): a refutat ionally complete theorem proving strategy based on a canonical rewrite system for Boolean algebra, and a programming paradigm for computing with *AC* arithmetic operators. In Section 3, we suggest methods for proving termination of rewrite systems containing *A C* operators.

2. Applications and Implementations

The Kuuth-Bcndix [1970] *completion procedure* was introduced as a means of deriving canonical term-rewriting systems to serve as decision procedures for given equational theories. The procedure creates new rewrite rules to resolve ambiguities resulting from existing rules that overlap. The *REVE* environment provides an interactive mechanism for generation of canonical rewriting systems, allowing the user to specify and construct (semi-automatically) any of several different orderings for completion of his system. Using completion, *REVE* is able to prove inductive theorems without explicitly invoking induction (see, for example, Huet and Hullot [I980]). An implementation of *A* C-rewriting has been incorporated into the *REVE* term rewriting environment. To extend the system to our purposes, we have added an *AC* unification algorithm (Stickel [1981]), and a mechanism for efficiently finding basis solutions to the linear Diophantine equations which arise from it (Huet [1978]). In addition, the completion procedure has been modified to handle associative-commutative operators (see Peterson and Stickel [1981]). We plan a series of experiments using this system. In particular, we are interested in using *AC* unification and a complete rewrite system for Boolean algebra for refutational theorem proving and logic programming.

•This work was supported in part by NSF Grant MCS 79-08497.
♦♦On leave from University of Illinois.

The following is a canonical rewrite system for Boolean algebra (Hsiang [1982], Watts and Cohen [1980]):

$$\boxed{\begin{array}{rcl}
\sim u & \rightarrow & u \oplus \mathbf{true} \\
u \vee v & \rightarrow & u \wedge v \oplus u \oplus v \\
u \supset v & \rightarrow & u \wedge v \oplus u \oplus \mathbf{true} \\
(u \oplus v) \wedge w & \rightarrow & u \wedge w \oplus v \wedge w \\
u \wedge \mathbf{true} & \rightarrow & u \\
u \wedge \mathbf{false} & \rightarrow & \mathbf{false} \\
u \wedge u & \rightarrow & u \\
u \oplus \mathbf{false} & \rightarrow & u \\
u \oplus u & \rightarrow & \mathbf{false}
\end{array}}$$

where ~ is 'not', A is 'and', V is 'inclusive-or\ © is Vxclusive-or\ and > is 'implies'. Both A and © are implicitly *AC* operators. That means, for example, that the rule uAU-»U applied to *(p/\q)/\p* yields *pf\q.* Since these functions are associative, there is no significance to the parent hesitation, and accordingly terms are "flattened" by removing embeddings of associative functions symbols, e.g. *[p/\q)/\p* is written *p/\ q/\p.*

That this system is sound follows from the fact that each rule is a proposit ional equivalence and A and © are in fact associative and commutative. Any term that is not a sum of conjunctions is reducible, since it must either contain a symbol other than A, in which case one of the first three rules can reduce it, or else it must contain a conjunction of a sum, in which case it is reducible by the fourth rule. The methods of the following section can be used to prove termination of the rules. The system is confluent, since all its critical pairs reduce to the same term. When, as in this example, some of the functions on the left-hand sides of / or /' are associative and commutative, then an associative-commutative unification algorithm (Livesey and Siekmann [1970], Stickel [1981]) is used to find sigma such that /[sigma] and /' *[sigma]* overlap. The definition of "overlap" must include cases in which two rules have overlapping subtcrms of the same associative-commutative symbol (Lankford and Ballantyne [1977], Peterson and Stickel [1981]). To do this, *pseudo-rules* /(/, u')—►/(r ,u') are considered for each rule whose left-hand side / has an associative-commutative outermost symbol /. All such critical pairs must reduce to the same terra up to permutation of arguments of the associative-commutative symbols.

Since the Boolean algebra system is sound, terminating, and confluent, it is a canonical system for Boolean algebra and provides a means of deciding the validity of propositional equivalences. Thus, the system may be used to check for both validity and unsatisfiability of propositional terms by reducing terms to their unique normal form: a term that reduces to **true** is valid, a term that reduces to **false** is unsatisfiable, while a term that reduces to neither is satisfiable but not valid.

In Hsiang and Dershowitz [1983] we have shown how the Boolean algebra system may be used for resolution-like theorem proving in the first-order predicate calculus. For example, adding the rule

$$\sim[p(z,a)\wedge p(z,x)\wedge p(x,z)]\wedge[(p(z,fz)\wedge p(fz,z))\vee p(z,a)] \rightarrow \mathbf{true}$$

(where f is a unary function symbol), and applying the *AC* completion procedure, generates

$$\begin{array}{rcl}
p(z,fz)\wedge p(fz,z) & \rightarrow & p(z,a)\oplus\mathbf{true} \\
p(z,a)\wedge p(z,x)\wedge p(x,z)\oplus\mathbf{true} & \rightarrow & \mathbf{true} \\
p(a,a)\oplus\mathbf{true} & \rightarrow & \mathbf{true} \\
\mathbf{false} & \rightarrow & \mathbf{true},
\end{array}$$

proving that

$$(\forall y)(\exists z)\sim[p(z,y)\equiv(\forall x)\sim(p(z,x)\wedge p(x,z))]$$

Rewrite systems may be used as "logic programs" (Kowliski [1974]), in addition to their straightforward use for computation by rewriting. The programming paradigm described below allows for the advantageous combination of both computing modes. The result is a Prolog-like programming language the main differences being that rewrite rules are equivalences, rather than implications in Horn-clause form, and that the completion procedure acts as the interpreter, rather than resolution. Hogger [1981] suggested the use of equivalences to specify Prolog programs.

For example, the following is a program to compute the quotient and remainder of two integers:

$$\boxed{\begin{array}{rcl}
div(u+v+1,v+1,q+1,r) & \rightarrow & div(u,v+1,q,r) \\
div(u,u+w+1,0,u) & \rightarrow & \mathbf{true} \\
x+0 & \rightarrow & x \\
div(u+v+1,v+1,1,r) & \rightarrow & div(u,v+1,0,r) \\
div(v+1,v+1,q+1,r) & \rightarrow & div(0,v+1,q,r) \\
div(v+1,v+1,1,r) & \rightarrow & div(0,v+1,0,r) \\
div(u+1,1,q+1,r) & \rightarrow & div(u,1,q,r) \\
div(u+1,1,1,r) & \rightarrow & div(u,1,0,r) \\
div(1,1,q+1,r) & \rightarrow & div(0,1,q,r) \\
div(1,1,1,r) & \rightarrow & div(0,1,0,r) \\
div(u,u+1,0,u) & \rightarrow & \mathbf{true} \\
div(0,1,0,0) & \rightarrow & \mathbf{true}
\end{array}}$$

where + is associative and commutative. The first rule is the main recursive case; the second is the main base case; the third simplifies sums; the remainder are special cases. For example, to compute the quotient and remainder of the two numbers 7 and 3 with this system (the numerals are just abbreviations for their unary representation as sums of ones), the rule

$$div(7,3,q,r) \rightarrow ans(q,r)$$

is added, meaning that *q* are r are the answer if and only if they are the quotient and remainder, respectively, of 7 and 3. The *AC* completion procedure then generates the rules:

$$\begin{array}{rcl}
div(4,3,q,r) & \rightarrow & ans(q+1,r) \\
div(1,3,q,r) & \rightarrow & ans(q+2,r) \\
ans(2,1) & \rightarrow & \mathbf{true},
\end{array}$$

where the *ans* predicate contains the answer values 2 and 1 for q and r, respectively.

3. Termination

The usual methods for proving termination of rewrite systems are frequently difficult to apply in the presence of associative and commutative symbols. Since AC completion requires an ordering for associative-commutative terms, it is important that we investigate such orderings. This section presents methods for proving termination of term-rewriting systems containing implicitly AC function symbols.

3.1. Characterization of AC Orderings

The following lemma is straightforward. (It is slightly stronger in the interesting direction than the method in Manna and Ness [1970]):

Lemma 1 . *A rewrite system R over a set of terms T terminates, if and only if there exists a well-founded set $(W,>)$ and a function τ mapping terms from T into W, such that*

$$u \Rightarrow v \wedge \tau(u) > \tau(v) \supset \tau(f(\cdots u \cdots)) > \tau(f(\cdots v \cdots))$$

for all such terms in T and for each rule $l \rightarrow r$ in R

$$\tau(l) > \tau(r),$$

for every substitution of terms in T for the variables of l.

Let $\overline{t}$ denote the flattened version of a term t, with all nested occurrences of symbols in F stripped, and where the order of arguments of symbols in F is not significant. Two terms u and v are equal in AC, if and only if $\overline{u}$ and $\overline{v}$ are the same. Let $\overline{T}=\{\overline{t}:t\in T\}$. We assume that R includes *pseudo rules* $f(l,x) \rightarrow f(r,x)$, where x is a variable otherwise not occurring in the rule, for each rule $l \rightarrow r$ in R whose left-hand side l or right-hand side r has outermost associative-commutative symbol $f \in F$ or whose right-hand side is just a variable. For example, the rule $x \oplus 0 \rightarrow x$ in the Boolean algebra system implies the two pseudo-rules $x \oplus 0 \oplus u \rightarrow x \oplus u$ and $(x \oplus 0) \wedge u \rightarrow x \wedge u$. (We are including more pseudo-rules than necessary for "compatibility." See Peterson and Stickel [April 1981].)

Theorem 1. *An AC rewrite system R terminates, if and only if there exists a well-founded ordering $>$ on $\overline{T}$ such that*

$$u \Rightarrow v \wedge \overline{u} > \overline{v} \supset f(\cdots \overline{u} \cdots) > f(\cdots \overline{v} \cdots)$$

for $\overline{u},\overline{v} \in \overline{T}$, and

$$\overline{l} > \overline{r}$$

for each rule and pseudo-rule $l \rightarrow r$ of R.

3.2. Unsuccessful Extensions

A *multiset* is a set along with a multiplicity assigned to each element. Any ordering $>^{*}$ on terms can be extended to a *multiset ordering* on terms $\gg^{*}$ in the following way (Dershowitz and Manna [1979]):

$$S=\{s_1, \ldots, s_m\} \gg^{*} \{t_1, \ldots, t_n\}=T$$

if and only if

$$(\exists M \subset S)(\forall t_j \in T-M)(\exists s_i \in S-M) s_i >^{*} t_j.$$

Given an ordering $>$ on function symbols, the *recursive path ordering* $>_{rpo}$ on terms containing those symbols is defined as follows (Plaisted [1978], Dershowitz [1982]): For two terms $s=f(s_1, \cdots, s_m)$ and $t=g(t_1, \cdots, t_n)$, we say

$$s >_{rpo} t$$

if and only if

$$f=g \text{ and } \{s_1, \ldots, s_m\} \gg_{rpo} \{t_1, \ldots, t_n\}$$

or

$$f>g \text{ and } (\forall t_j) s >_{rpo} t_j$$

or

$$(\exists s_i) s_i \geq_{rpo} t,$$

where $\gg_{rpo}$ is the multiset extension of $>_{rpo}$.

The following unsuccessful extensions of the recursive path ordering indicate some of the difficulties in finding useful AC orderings.

1) $>_{rpo}$ on flattened terms.
   Consider two AC operators f and g with $f>g$. We certainly have $f(a,b) >_{rpo} g(a,b)$. However, we cannot show that $f(a,b,c) >_{rpo} f(g(a,b),c)$.

2) $>_{rpo}$ on flattened terms, refining operators by their arity.
   Consider two AC operators f and g with $f>g$. Although we have $f(a,b) >_{rpo} g(a,b)$, we cannot show $g(f(a,b),c) >_{rpo} g(a,b,c)$.

3) $>_{rpo}$ on flattened terms, eliminating identical subterms.
   Consider one function symbol f with $f>a>b>c>d$. This ordering however is not well-founded:
   $$f(b,b,c) >_{rpo} f(a,c) >_{rpo} f(a,d) >_{rpo} f(b,b,c).$$

4) $>_{rpo}$ on flattened terms, looking at powersets of subterms.
   Considering the collection of terms that can be obtained from a term $f(s_1, \cdots, s_m)$ by deleting subterms also does not give a valid ordering.

3.3. Polynomial Interpretations

Since addition and multiplication are themselves associative and commutative, polynomial interpretations, as used, for example, in Lank ford [1975], are sometimes helpful. For example, to show termination of the Boolean algebra example, it is easily verified that *T(l)>r(r)* for each of the rules and pseudo-rules /—>r, where *T\l)* is defined inductively on the structure of J:

$$\begin{aligned}
\tau(a) &= 2 \\
\tau(\sim\alpha) &= \tau(\alpha)+3 \\
\tau(\alpha \vee \beta) &= \tau(\alpha)*\tau(\beta)+\tau(\alpha)+\tau(\beta)+3 \\
\tau(\alpha \supset \beta) &= \tau(\alpha)*\tau(\beta)+\tau(\alpha)+\tau(\beta)+3 \\
\tau(\alpha \oplus \beta) &= \tau(\alpha)+\tau(\beta)+1 \\
\tau(\alpha \wedge \beta) &= \tau(\alpha)*\tau(\beta),
\end{aligned}$$

and a is any atomic symbol, including 0 and 1.

.3.4. Semantic Path Ordering

Intuitively, the Boolean algebra system terminates, since distributivity docs, and the other rules only simplify matters. The following ordering attempts to capture that intuition.

Let *p(t)* be some function mapping terms *t* into the natural numbers (or any well-founded set) and let > be a well-founded ordering of the function symbols, one of which we denote $\otimes$. The *semantic path ordering* (Kamin and Levy [1980]) is an extension of the recursive path ordering and is defined recursively as follows:

$$s = f(s_1 \cdots s_m) >_{spo} g(t_1 \cdots t_n) = t$$

if and only if

$$s >_{spo} t_j,\ j = 1, \cdots, n$$

and one the following holds:

$$f > g \ \text{ or } [f = g = \otimes \wedge \rho(s) > \rho(t)]$$

or

$$s_i \geq_{spo} t \ \text{ for some } i = 1, \cdots, m$$

or

$$f = g \neq \otimes \ \text{ and } \{s_1 \cdots s_m\} \gg_{spo} \{t_1 \cdots t_n\}$$

or

$$(f = g = \otimes \wedge \rho(s) = \rho(t)) \ \text{ and } \{s_1 \cdots s_m\} \gg_{spo} \{t_1 \cdots t_n\}$$

where $\gg_{spo}$ is the multiset extension of $>_{spo}$.

Kamin and Levy [1980] have shown that if

$$s \Rightarrow t \supset \rho(f(\cdots s \cdots)) \geq \rho(f(\cdots t \cdots)) \qquad (*)$$

holds for all f, then a semantic recursive ordering (such as the above one) is well-founded and satisfies the monotonicity condition of Theorem 1.

The difficulty of using this method lies in picking $\rho(t)$ and in checking condition $(*)$. For the Boolean algebra example, let $\wedge > \oplus > 1 > 0$. We can prove the termination of the first four rules (including distributivity) by taking $\otimes$ to be $\wedge$ and $\rho(t_1 \wedge \cdots \wedge t_n)$ to be n. We can then prove the termination of the complete system by taking $\otimes$ to be $\wedge$, again, and $\rho(t)$ to be the length of the longest derivation from t using those first four rules. That requires verifying that $(*)$ holds and, for each rule or pseudo-rule $l \rightarrow r$, we have $l >_{spo} r$.

3.5. Associative Path Ordering

In the recursive path ordering, to show *s>rpot,* it is sometimes necessary to show that a given subterm of *s* is greater than more than one subterm of (, but it is *never* necessary to show that a given subterm of *t* is less than more than one subterm of s. Therefore, if we transform *s* and *t* and can make choices in the transformations, it is easy to show that for all choices for s there is a choice for *t* such that s>*t.* However, it is much harder to show the reverse (for all choices for *t* there is a choice for *s* such that *s>t).* With this motivation, we define the *associative path ordering* as follows: With term *t* we associate a set *M(t)* of *transforms* of *t.* We say $s >_{apo} t$ in the associative path ordering if $\forall u \in M(s)\ \exists v \in M(t)\ u >_{rpo} v$ Thus, the sets *M(t)* are ordered by their *small* elements. *M(t)* is formed by successively applying three transforms to the "flattened" version of *t.* (Note that during flattening, /(u) is replaced by *u* when / is associative.) The role of *M1* is to transform subterms, of M2 to transform terms at the top level, and of *M3* to again transform subterms. Composing the three operations, we get $M(t) = M_3(M_2(M_1(\bar{t})))$. The transforms are given by:

$$M_1(g(u_1, \ldots, u_m)) = \{\overline{g(v_1, \ldots, v_m)} : v_i \in M(u_i)\}$$

$$M_2(\{t_1, \ldots, t_n\}) = \bigcup_{1 \leq i \leq n} F(t_i)$$

$$M_3(\{t_1, \ldots, t_n\}) = \bigcup_{1 \leq i \leq n} M_1(t_i),$$

where

$$F(g(v_1, \ldots, v_m)) = \{g(v_1, \ldots, v_m)\}$$

if g is *not* associative or *no* subterm v_i has function symbol f for which $g > f$; otherwise,

$$F(g(v_1, \ldots, v_m)) = \{T_i(g(v_1, \ldots, v_m)) \mid 1 \leq i \leq m\},$$

with $T_i(g(v_1, \ldots, v_m))$ given by:

$$f(g(\ldots\ldots), \ldots, g(\ldots,\ldots), g(\ldots,s_1,\ldots), \ldots \underset{\bullet}{,} g(\ldots,s_k,\ldots))$$

if $v_i = f(s_1, \ldots, s_k)$, g *and* f are associative, $g > f$, where $g(\ldots,\ldots)$ appears as an argument $k \cdot 1$ times;

$$f(g(\ldots,\ldots), g(\ldots,s_1,\ldots), \ldots, g(\ldots,s_k,\ldots))$$

if $v_i = f(s_1, \ldots, s_k)$, g is associative, f is *not* associative, and $g > f$.

For example, *F(g(c ,d))={c(d),d(c)}* if *g>c,d* and *g* is associative.

Example: Consider the Boolean algebra system. Using the ordering 1 *,a,b,c* >0>A>© (i.e. constants are big), validates a®a-»0 since the ordering on function symbols implies *x>apo0* for any term /. (Making constants large is, in fact, necessary for *>apo* to work on this example.) Transforming the left and right hand sides of the distributivity rule yields

$$F(a \wedge (b \oplus c)) = \{a \oplus (a \wedge b) \oplus (a \wedge c)\}$$
$$F((a \wedge b) \oplus (a \wedge c)) = \{(a \wedge b) \oplus (a \wedge c)\}.$$

Thus $a \wedge (b \oplus c) >_{apo} (a \wedge b) \oplus (a \wedge c)$, and this rule, too, is a reduction. Similarly, for the other rules and pseudo-rules.

Proposition . *The ordering $>_{apo}$ is well-founded since it satisfies the conditions for simplification orderings, as defined in Dershowitz [1982].*

Note that the condition VuEA/(*) ;Ir€A/(0 *u*>*rpo v* depends only on the *minimal* elements of *M(s)* and *M{t).* Hence we need only compute *minimal* elements of A/(/) at each step (including subterms of *t).* This can greatly speed up the computations. In fact, if the ordering on operators is *total,* only *one* minimum element need be kept. We have implemented the associative recursive path ordering and are making it interactive for *AC* completion in *REVE* and are working on comparing terms with variables in *>apo.*

REFERENCES

1. Dersbowitz, N. [1982], Orderings for term-rewriting systems, *J. of Theoretical Computer Science* 17:3, pp. 279-

301.

2. Dershowitz, N., and Z. Manna [1979], Proving termination with multiset orderings, Comm. ACM 22:8, pp. 405-470.

3. Hogger, C.J. [Apr. 1981], "Derivation of logic programs/[1] J. ACM, vol. 28, no. 2, pp. 372-392.

4. Ilsiang, J. [1982], Topics in automated theorem proving and program generation, Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana, Illinois.

5. Hsiang, J. and Dershowitz, N. [1983], Rewrite methods for clausal and non-clausal theorem proving, Proc. 10th EATCS Intl. Golloq. on Automata, Languages and Programming. Barcelona, Spain, to appear.

0. Huet, G, and J. Hullot [1980], Proofs by induction in equational theories with constructors, Proc. 21st Annual Symposium on Foundations of Computer Science, pp. 90-107.

7. Huet, G., and D. C. Oppen [1980], Equations and rewrite rules: a survey, in Formal Languages: Perspectives and Open Problems (R. Book, ed.) Academic Press, New York.

8. Kamin, S.. and J.J. Levy [1980], Two generalizations of the recursive path ordering, Unpublished note, Department of Computer Science, University of Illinois, Urbana, Illinois.

9. D. E. Knuth [1908], Fundamental algorithms. The Art of Computer Programming, vol. 1, Addison-Wcsley, Reading.

10. Knuth, D. E., and P. B. Bendix [1970], Simple word problems in universal algebras, Computational Problems in Abstract Algebra (J. Leech, ed.) Pergamon Press, Oxford, pp. 203-297.

11. Kowalski, R. [1974], Predicate logic as a programming language, D.C.L. Memo No. 70, Edinburgh University, Edinburgh, Scotland.

12. Lankford, D. S. [1975], Canonical algebraic simplification in computational logic, Report No. ATP-25, Department of Mathematics, Southwestern University, Georgetown, Texas.

13. Lankford, D. S., and A. M. Ballantyne [1977], Decision procedures for simple equational theories with commutative axioms: Complete sets of commutative reductions, Memo ATP-35, Department of Mathematics and Computer Sciences, University of Texas, Austin, Texas.

14. Lescanne, P. [1983], Computer experiments with the REVE term rewriting system generator, Proc. 10th Conf. on Principles of Programming Languages.

15. Livesey, M., and J. Siekmann [1976], Unification of A + C-terms (bags) and A+C+I-terms (sets), Intern. Ber. Nr. 5/70, Inst, fur Informatik, University Karlsruhe, Karlsruhe, W. Germany.

10. Manna, Z., and S. Ness [1970], On the termination of Markov algorithms, Proc. of the Third Hawaii International Conference on System Science, Honolulu, Hawaii.

17. Peterson, G. E., and M. E. Stickel [1981], Complete sets of reductions for some equational theories, J. ACM 28:2, pp. 233-204.

18. Plaisted, D. A. [1978], A recursively defined ordering for proving termination of term rewriting systems, University of Illinois Report No. UIUCDCS-R-78-943, Department Computer Science, University of Illinois, Urbana, Illinois.

19. Stickel, M.E. [1981], A unification algorithm for associative-commutative functions, J. ACM, 28:3, pp. 423-134.

20. Watts, D. E., and J. K. Cohen [1980], Computer implemented set theory, American Mathematical Monthly 87:7, pp. 557-500.