Equational Reasoning using AC Constraints

David A. Plaisted and Yunshan Zhu
Computer Science Department
University of North Carolina
Chapel Hill, NC 27599-3175
{plaisted,zhu}@cs.unc.edu
Fax: (919)962-1799

Abstract

Unfailing completion is a commonly used technique for equational reasoning. For equational problems with associative and commutative functions, unfailing completion often generates a large number of rewrite rules. By comparing it with a ground completion procedure, we show that many of the rewrite rules generated are redundant. A set of consistency constraints is formulated to detect redundant rewrite rules. We propose a new completion algorithm, consistent unfailing completion, in which only consistent rewrite rules are used for critical pair generation and rewriting. Our approach does not need to use flattened terms. Thus it avoids the double exponential worst case complexity of AC unification. It also allows the use of more flexible termination orderings. We present some sufficient conditions for detecting inconsistent rewrite rules. The proposed algorithm is implemented in PROLOG.

1 Introduction

Knuth-Bendix completion [Knuth and Bendix, 1970] and its extensions [Bachmair *et al.*, 1989] have been widely used for equational reasoning. One of the main bottlenecks of completion-based inference strategies is the large number of critical pairs generated. This is particularly evident when dealing with equational problems involving associative and commutative functions, i.e. AC equational problems. In this paper, we present a technique for reducing redundant equational inferences using constraints.

AC equational problems represent an important class of problems in theorem proving. Many mathematical functions of interest are associative and commutative. For example, the union and intersection operations in set theory and the addition operations in ring structures are all associative and commutative. Furthermore, addition and multiplication in arithmetic are both AC. Some form of AC equational reasoning is necessary if a term rewriting system is used to perform integer arithmetic.

AC equational reasoning is often difficult. One of the most recognized recent successes of automated reasoning, perhaps of AI in general, is the solution of the Robbins Problem. The Robbins Problem is in fact formulated as an equational problem with AC functions [McCune, 1996]. Most term rewriting systems do inferences by generating rewrite rules using Knuth-Bendix completion or unfailing completion. Associativity axioms and commutativity axioms often cause an explosion of new rewrite rules in the completion process. When overlapped with the AC axioms, an existing rewrite rule can generate an exponential number of AC equivalent rewrite rules. Since commutativity axioms cannot be oriented in any termination ordering, many of these new rewrite rules generated from the AC axioms cannot be simplified.

Special completion procedures are developed to handle AC equational problems [Lankford and Ballantyne, 1977; Peterson and Stickel, 1981]. Most of the approaches in the literature use flattened terms. For example, +(a,+(6,c)) is represented as +(a,b,c), and terms +(a,c, b) and +(a,b, c) are considered identical. A flattened term represents all terms equivalent up to the AC axioms. Since flattening breaks the original term structure, special unification techniques are needed. AC unification algorithms [Stickel, 1981; Domenjoud, 1991; Boudet *et al.*, 1996] are developed to compute all possible most general unifiers of two terms up to associativity and commutativity axioms. Special AC termination orderings are also needed to show termination of AC rewriting systems. Many commonly used orderings, such as recursive path ordering and lexicographic path ordering, are no longer well founded when flattened terms are used. Several AC termination ordering have been devised [Dershowitz *et al.*, 1983; Bachmair and Plaisted, 1985; Kapur *et al.*, 1990].

We study the problem of AC equational reasoning with a different approach. We propose a procedure called consistent unfailing completion in which only consistent rules and equations are used for critical pair generation and rewriting. A consistent unfailing completion procedure can be regarded as the lifted version of a ground completion procedure. In a standard unfailing completion procedure, some rewrite rules do not correspond to any rewrite rules in the ground procedure. We call such rules inconsistent. In consistent unfailing completion, inconsistent rules are replaced by its consistent permutations. For example, $x * y * x \to x$ is inconsistent with respect to the lexicographic ordering. This is because $a_1 * a_1 * a_2$, an arbitrary instance of $x * y * x$, is either simplified to $a_1 * a_1 * a_2$ or $a_2 * a_1 * a_1$ depending on the ordering of a_1 and a_2. $y * x * x \to x$ and $x * x * y \to x$ are the only two consistent permutations of $x * y * x \to x$. We derive two sufficient conditions for detecting inconsistent rules. Consistency checking based on these conditions greatly reduces the number of critical pairs generated. In the examples that we tested, it shows a factor of 2 to 4 reduction. Our implementation of the algorithm is able to solve all six of equality benchmark problems proposed by [Lusk and Overbeek, 1985].

Consistent unfailing completion does not require the use of flattened terms. AC unification is therefore not needed. Detecting inconsistency takes quadratic time and generating consistent permutations takes single exponential time. Thus the worse case complexity of an inference rule in consistent unfailing completion is single exponential. The worse case complexity for AC unification is double exponential. A special AC termination ordering is not needed either. Consistent unfailing completion works for most of the commonly used orderings, such as recursive path ordering, lexicographic path ordering, etc. In fact, we have a built-in ordering in our implementation, and all of our examples are solved using the same ordering. The built-in ordering enables us to fully automate the equational reasoning process and incorporate it in a general first-order theorem prover.

The rest of the paper is organized as follows. We provide some background in the next section. We then present the consistent unfailing completion algorithm and show its completeness. We then show how the algorithm can be refined and implemented. We give some test results in the end.

2 Background

In this section, we define some relevant concepts. For surveys on equational reasoning, see [Plaisted, 1993; Dershowitz and Jouannaud, 1990; Klop, 1992]. For introductions to the area of theorem proving, see [Chang and Lee, 1973; Loveland, 1978; Wos *et al.*, 1984]. We follow the conventions of [Plaisted, 1993] in this paper.

We use the standard definitions of term, substitution, instance, the most general unifier etc. Readers may consult the aforementioned references for more details. We use $f, g, h, \ldots$ as function symbols, $a, b, c, \ldots$ as constant symbols, and $x, y, z, \ldots$ as variables.

An equation is an expression of the form $s = t$ where s and t are terms. An equational system is a set of equations. Assume E is an equational system, we use $E \models s = t$ to represent that $s = t$ is a logical consequence of E. The problem of equational reasoning can be reduced to the problem of first-order theorem proving. $E \models s = t$ iff $\{E \cup A_E\} \vdash_L s = t$, where A_E is the set of equality axioms and L is any sound and complete first-order strategy. In practice, term rewriting based approaches are often much more efficient.

A rewrite rule is written as $r \to s$, where r and s are terms. A rule $r \to s$ indicates that an instance of r can be replaced by an instance of s, but not vice versa. $r \leftrightarrow s$ if $r \to s$ or $s \to r$. Sometimes $r \to s$ is also rewritten as $s \leftarrow r$. A term rewriting system R is a set of rewrite rules. We use $\to_R$ to represent the rewrite relation. For example, a term rewriting system R is $\{f(x) \to b, g(b) \to a\}$. Then $g(f(x)) \to_R g(b)$. We define $\to_R^*$ as the reflexive transitive closure of $\to_R$. Thus $g(f(x)) \to_R^* a$. We define $\leftrightarrow_R^*$ as the reflexive transitive closure of $\leftrightarrow_R$. Suppose R is a term rewriting system $\{r_1 \to s_1, \ldots, r_n \to s_n\}$, $R^=$ is defined as the associated equational system $\{r_1 = s_1, \ldots, r_n = s_n\}$. By Birkhoff's theorem [Birkhoff, 1935], $R^= \models r = s$ iff $r \leftrightarrow_R^* s$. This essentially shows the soundness and completeness of equational reasoning based on term rewriting. However, the rewriting system is not yet satisfactory for theorem proving purposes: to show $r \leftrightarrow_R^* s$, rewriting has to be done in both directions, and it is highly non-deterministic and inefficient.

A term r is reducible if there is a term s such that $r \to s$, otherwise r is irreducible. If $r \to^* s$ and s is irreducible then we call s a normal form of r. We write $r \downarrow s$ if there is a term u such that $r \to^* u$ and $s \to^* u$. We write $r \uparrow s$ if there is a term u such that $u \to^* r$ and $u \to^* s$. We say a rewriting system R is confluent if for all terms r and s, $r \uparrow s$ implies $r \downarrow s$. It can be shown that if R is confluent and $r \leftrightarrow_R^* s$ then $r \downarrow s$. A rewriting system R is terminating if it has no infinite rewriting sequence. If there exists a termination ordering $>$ such that for all rules $l \to r$ in R, $l > r$, then R is terminating. If R is confluent and terminating, $r \leftrightarrow_R^* s$ can be decided by rewriting r and s to normal forms and compare the normal forms.

Definition 2.1 *Suppose that $l_1 \to r_1$ and $l_2 \to r_2$ are two rewrite rules with no common variables. Suppose u is a non-variable subterm of l_1, (if l_1 is a variable, u may be a variable). Suppose u and l_2 are unifiable and θ is the most general unifier. We call the pair $(l_1[u \leftarrow r_2]\theta, r_1\theta)$ a critical pair for rules $l_1 \to r_1$ and $l_2 \to r_2$. $l_1[u \leftarrow r_2]$ denotes the term obtained by replacing a specified occurrence of the subterm u in l_1 with r_2.*

Theorem 2.2 *Suppose a term rewriting system R is terminating. R is confluent iff for all critical pairs (s, t) in R, $s \downarrow t$.*

If a rewriting system is not confluent, an equivalent rewriting system might be obtained by adding new rewriting rules. Theorem 2.2 can be used to generate such new rules. The idea is to add rule $s \to t$ or $s \leftarrow t$ to R while preserving termination for all critical pairs (s, t) in R. The algorithm was initially proposed in [Knuth and Bendix, 1970], and is often called Knuth-Bendix completion, or simply, completion. Sometimes, it is not possible to orient an equation, i.e. $f(x, y) =$

$f(y,x)$, in either direction while maintaining termination. Thus Knuth- Bendix completion is extended to unfailing completion[Bachmair *et al.*, 1989]. In unfailing completion, if a critical pair (s,t) can not be oriented then it is added as an equation $s = t$. Equation $s = t$ may participate in further critical pair generation. It is regarded as containing rules $s \rightarrow t$ and $t \rightarrow s$. However, the use of equation in rewriting is restricted : $s\theta$ can be replaced by $t\theta$ only if $s\theta > t\theta$, where $>$ is the termination ordering.

We give names to a set of inference rules. We follow the convention used in [Plaisted, 1993]. The rule $UCP(\rightarrow,\rightarrow)$ is the operation of generating critical pairs between two rewrite rules. If a critical pair can be oriented, a new rewrite rule is added to the rewrite system; otherwise, an equation is added. UCP stands for Unfailing Critical Pair generation. Similarly, $UCP(=,\rightarrow)$ generates critical pairs between a rewrite rule and an equation; $UCP(=,=)$ generates critical pairs between two equations. Inference rule $SIMPRULE$ is the operation of simplifying a rewrite rule or an equation with other existing rewrite rules or equations. For example, if a rewriting system R contains $\{f(x) \rightarrow x,\ f(b) \rightarrow f(a)\}$, the second rewrite rule can be simplified to $b \rightarrow a$, and thus R contains $\{f(x) \rightarrow x,\ b \rightarrow a\}$ after rule $SIMPRULE$ is applied. $E \models s = t$ iff $E \cup \{s' \neq t'\}$ is unsatisfiable, where s' and t' are skolemized terms of s and t, respectively. Thus we may include $\{s' \neq t'\}$ as an inequation in a rewriting system R and show a contradiction can be derived from R. $SIMPGOAL$ is the operation of simplifying an inequation using existing rewrite rules or equations. The rule $CONTRA$ derives $FALSE$ when the inequation $s \neq s$ is derived. In applying these inference rules, the concept of *fairness* is needed. Fairness means that every henceforth possible inference will eventually be performed.

Theorem 2.3 *Fair unfailing completion starting from a set of R of rules, equations and ground inequation and using the inference rule $\{UCP(\rightarrow,\rightarrow), UCP(=,\rightarrow), UCP(=,=), SIMPRULE, SIMPGOAL, CONTRA\}$ will eventually generate $FALSE$ if $R^=$ is unsatisfiable.*

Theorem 2.3 show the completeness of an equational reasoning strategy based on unfailing completion. We will also use the fact that *SIMPRULE* is inessential to the completeness of the strategy.

In this paper, we are concerned with equational systems that involve associative and commutative functions, or AC functions. An AC function f satisfies the following axioms $\{f(f(x,y),z) = f(x,f(y,z)), f(x,y) = f(y,x)\}$.

3 AC-Consistent Completion

In this section, we introduce the concept of consistency for AC equational problems. We present a new completion algorithm in which only consistent rules can be used for critical pair generation and simplification. We call the new algorithm consistent unfailing completion. We show the correspondence between a consistent unfailing completion and a ground completion procedure. The correspondence helps us in deriving the consistency constraints and establishing the completeness proof of the consistent unfailing completion algorithm.

Definition 3.1 *Suppose that $>$ is a termination ordering and $>$ is total on ground terms. A ground term s is an AC minimal term if s is the minimal term of its AC-equivalence class with respect to the ordering $>$. A rewrite rule $s \rightarrow t$ is consistent if there exists an instance $s\theta \rightarrow t\theta$ such that both $s\theta$ and $t\theta$ are AC minimal terms, and that θ assigns distinct ground terms to distinct variables in s and t. The consistency of an equation (a critical pair) is similarly defined.*

Example 3.2 *In this example, we use the lexicographic ordering as the termination ordering. We assume f is an AC function and g is not an AC function. $f(a,f(b,c))$ is an AC minimal term. $f(f(a,b),c)$ is not an AC minimal term, because $f(f(a,b),c) > f(a,f(b,c))$ and they are AC equivalent. We sometimes say $f(a,f(b,c))$ is the AC minimal term of $f(f(a,b),c)$. The rewrite rule $g(f(x,f(y,z))) \rightarrow a$ is consistent, $g(f(a,f(b,c)) \rightarrow a$ is a consistent instance of the rule. Rewrite rule $g(f(x,y)) \rightarrow f(y,x)$ is not consistent, neither are $f(g(x),g(y)) = f(y,x)$ and critical pair $(f(x,a),f(a,x))$.*

Definition 3.3 *Inference rule $CUCP(\rightarrow,\rightarrow)$ is the operation that generates critical pairs between two consistent rewrite rules. The critical pairs generated are oriented as rewrite rules or added as equations. $CUCP(\rightarrow,\rightarrow)$ is the same as $UCP(\rightarrow,\rightarrow)$ except that only consistent rewrite rules are used for critical pair generation in $CUCP$. $CUCP(=,\rightarrow)$ and $CUCP(=,=)$ are similarly defined. We say the extended set of AC axioms is the set $\{f(f(x,y),z) = f(x,f(y,z)),\ f(x,y) = f(y,x),\ f(x,f(y,z)) = f(y,f(x,z))\}$ for all AC functions f. Inference rule UCP_{AC} is the operation that generates critical pairs between a rule or an equation and an equation from the extended set of AC axioms.*

Theorem 3.4 *Fair consistent unfailing completion starting from a set of R of rules, equations and ground inequation and using the inference rule $\{CUCP(\rightarrow,\rightarrow), CUCP(=,\rightarrow), CUCP(=,=), UCP_{AC}, SIMPGOAL, CONTRA\}$ will eventually generate $FALSE$ if $R^=$ is unsatisfiable.*

Theorem 3.4 outlines the consistent unfailing completion procedure. The soundness of the procedure is obvious, as each inference rule generates only logical consequences. We show that consistent unfailing completion can be regarded as a lifted version of the ground completion procedure outlined in Theorem 3.6, and thus establish its completeness proof.

Definition 3.5 *Inference rule $INST$ is the operation of generating an instance of a rewrite rule or an equation and orienting it as a new rewrite rule. Inference rule $GCP(\rightarrow,\rightarrow)$ is the operation of generating critical pairs* *between two ground rewrite rules and orienting them as new rewrite rules. GSIMPAC is the operation of replacing a ground rewrite rule s —► t by a new rewrite rule s' -t' where s' and t' are AC minimal terms of s and t, respectively.*

Theorem 3.6 *Fair instance-based completion starting from a set of R of rules, equations and ground inequation and using the inference rule $\{INST, GCP(\rightarrow,\rightarrow), GSIMP_{AC}, SIMPGOAL, CONTRA\}$ will eventually generate $FALSE$ if $R^{=}$ is unsatisfiable.*

PROOF: If $R^{=}$ is unsatisfiable, by Herbrand's theorem, there is a finite unsatisfiable set of instances $R^{=}_{I}$ of $R^{=}$. By the fairness assumption, all instances in the set $R^{=}_{I}$ will be generated by $INST$. As a consequence of Theorem 2.3, $\{GCP(\rightarrow,\rightarrow), GSIMP_{AC}, SIMPGOAL, CONTRA\}$ will eventually generate $FALSE$ from the set $R^{=}_{I}$.

Remark 3.7 *The consistent unfailing completion procedure in Theorem 3.4 is a lifted version of the instance-based completion procedure in Theorem 3.6.*

We now give an example to show the effect of the consistency constraints.

Example 3.8 *Consider a rewriting system with the following rules, $\{(x*y)*z \rightarrow x*(y*z), x*y = y*x\}$. We use an ordering in which the associativity axiom can be oriented [1]. We refer to the associativity axiom as R_1, and the commutativity axiom as R_2.*

Applying $UCP(\rightarrow,=)$ on R_1 and R_2, equations $(y*x)*z = (x*y)*z$ and $z*(x*y) = (x*y)*z$ will be generated. Applying rule $SIMPRULE$, they are simplified to R_3 $y*(x*z) = x*(y*z)$ and R_4 $z*(x*y) = x*(y*z)$. Applying $UCP(=,=)$ and $SIMPRULE$ on R1 and R3, equation R_5 $x*(y*(z*w)) = y*(x*(z*w))$ will be generated.

Now let's consider the permutations of the variables on each side of the equation R_5. In unfailing completion, all such permutations, e.g. $x*(y*(z*w)) = y*(x*(w*z))$, will eventually be generated from R_1 and R_2. There are 24 such permutations up to the renaming of variables. Since none of the permutations except $x*(y*(z*w)) = x*(y*(z*w))$ can be simplified, 23 equations will be added to the rewriting system. However, none of the permuted equations are consistent, and $x*(y*(z*w)) = x*(y*(z*w))$ is an instance of $x = x$ and can be deleted. Thus in consistent unfailing completion, no permutations of the equation R_5 will be added to the rewriting system.

4 Refinements

In this section, we present some refinements of Theorem 3.4. Theorem 4.2 outlines the refined consistent unfailing completion algorithm. We then show how the algorithm can be efficiently implemented.

Definition 4,1 *Inference rule PERM is the operation that generates all consistent AC equivalent rewrite rules (equations) of an existing rewrite rule (equation). Inference rule $CUCP_A$ is the operation that generates critical pairs between a consistent rule or equation and the associativity axioms $\{f(f(x,y),z) = f(x,f(y,z))\}$.*

[1]one such ordering is defined in Definition 4.3.

The inference rule $PERM$ and $CUCP_A$ combined are equivalent to the rule UCP_{AC} in Theorem 3.4, and they are more efficient. $SIMPRULE$ often greatly reduces the number of rewrite rules and equations in a rewriting system. $SIMPRULE$ can be added to the consistent unfailing completion procedure in Theorem 3.4 without affecting its completeness. The proof involves extending the instance-based completion procedure as in Theorem 3.6 with a restricted version of $SIMPRULE$ and then lifting it to the nonground case.

Theorem 4.2 *Fair consistent unfailing completion starting from a set of R of rules, equations and ground inequation and using the inference rule $\{CUCP(\rightarrow,\rightarrow), CUCP(=,\rightarrow), CUCP(=,=), PERM, CUCP_A, SIMPRULE, SIMPGOAL, CONTRA\}$ will eventually generate $FALSE$ if $R^{=}$ is unsatisfiable.*

We now define an ordering that's used in our implementation.

Definition 4.3 *We define size lexicographic path ordering $>_{slpo}$ on ground terms as follows. Suppose s and t are ground terms, $s = f_s(a_1,\ldots,a_m)$ and $t = f_t(b_1,\ldots,b_n)$, and $>_l$ is the lexicographic ordering. $s >_{slpo} t$ iff*
1) $size(s) > size(t)$, or
2) $size(s) = size(t)$ and $f_s >_l f_t$, or
3) $size(s) = size(t)$, $f_s = f_t$, $a_k = b_k$ $\forall k$ from 1 to $i-1$, and $a_i >_{slpo} b_i$.

$size(s)$ is defined as the length of s written as a character string(excluding commas and parentheses). The size lexicographic path ordering is extended to nonground terms as follows, $s >_{slpo} t$ iff $\forall\theta$ $s\theta >_{slpo} t\theta$. It can be difficult to order two nonground terms with respect to $>_{slpo}$. In practice, we use sufficient conditions based on special cases such as $g(x) >_{slpo} x$ and $f(x,x) >_{slpo} g(x)$. It can be showed that size lexicographic path ordering is a termination ordering.

Definition 4.4 *Suppose that f is an AC function, we call $f(s_1,s_2,\ldots,s_n)$ the flattened term of $f(s_1,f(s_2,\ldots f(s_{n-1},s_n),\ldots))$, where s_i does not contain f as a top level function symbol. Suppose that s and t are subterms of an expression e, we say s AC-precedes t in e, or $s << t$, if the flattened term of e contains a subterm of the form $f(\ldots,s,\ldots,t,\ldots)$, where f is an AC function.*

We now describe two sufficient conditions for detecting the inconsistency of rewrite rules(or equations).

Theorem 4.5 *Suppose a size lexicographic path ordering $<_{slpo}$ is used as the termination ordering. A rewrite rule $s \rightarrow t$ is inconsistent if*
1) $u_1 <_{slpo} u_2$ and $u_2 << u_1$
or 2) $u_1 << u_2$ and $u_2 << u_1$, where u_1 and u_2 are subterms of s and t, respectively.

Proof: For case 1): Equation $s = t$ contains a subterm $f(\ldots, u_2, \ldots, u_1, \ldots)$. Since $u_1 <_{slpo} u_2$, $f(\ldots, u_2, \ldots, u_1, \ldots)\theta >_{slpo} f(\ldots, u_1, \ldots, u_2, \ldots)\theta$, for all θ. That is, no instances of $f(\ldots, u_2, \ldots, u_1, \ldots)$ can be an AC minimal term. $s \rightarrow t$ is thus inconsistent. The proof for case 2) is similar.

The inference rule *PERM* generates all consistent AC-equivalent rewrite rules or equations of an existing rewrite rule or equation. It essentially involves solving a constraint satisfaction problems. The simplest solution is to enumerate and check. Namely, for each rewrite rule $s \rightarrow t$, all AC equivalent rules of $s \rightarrow t$ are enumerated, and then Theorem 4.5 is used to filter out the inconsistent rules. More specifically, to apply *PERM* on a rewrite rule $s \rightarrow t$, we flatten $s \rightarrow t$, collect new rewrite rules by permuting the arguments of AC functions in $s \rightarrow t$, delete all inconsistent rules, and finally unflatten the remaining rewrite rules in the collection. For a term $f(a_1, \ldots, a_n)$ where f is an AC function, unflattening the term generates $f(a_1, f(\ldots, f(a_{n-1}, a_n)))$. To unflatten a term or an expression, unflattening is done recursively for all its subterms. For example, $g(f(a, h(b), x)) \rightarrow c$ is unflattened to $g(f(a, f(h(b), x))) \rightarrow c$, assuming that f is the only AC function. Note that flattened terms are only used as an intermediate representation for detecting inconsistent rules, and they are not used in the rewrite rules and equations of the rewriting system.

Theorem 4.5 provides only sufficient conditions for inconsistency, and thus some inconsistent rules can also be generated. More deliberate approaches based on constraint satisfaction might be able to compute the exact set of consistent rules, and thus further reduce the number of critical pairs and rewrite rules generated in the completion procedure.

5 Test Examples

It is fairly straightforward to implement the consistent unfailing completion algorithm. We implemented the algorithm with several hundred lines of Prolog code. We tested a number of pure equality problems. The results are listed in Table 1.

E1-E6 is a set of benchmark problems for equality proposed by [Lusk and Overbeek, 1985]. E3 and E6 involve AC functions. wos21 (equality version), RNG015-6, RNG023-6 and RNG024-6 are problems from ring theory, all four of them involve AC functions. For non-AC problems, the procedure is same as unfailing completion. For AC problems, we study the effect of consistency checking by comparing the result from unfailing completion and consistent unfailing completion. Note that consistency checking significantly reduces the number of critical pairs generated, the number of rules kept and the time needed to obtain a proof. Our implementation serves the purpose of demonstrating the effect of consistency checking. A state of the art equality prover can solve most of the listed problems in tens of seconds[McCune, 1990]. Efficient data structures can be used to improve our current implementation.

Problem	without AC constraints			with AC constraints		
	CPs	Rules	Time	CPs	Rules	Time
E1(GRP001-2)	261	26	2.8	n/a	n/a	n/a
E2(GRP022-2)	191	18	1.9	n/a	n/a	n/a
E4(GRP002-4)	90830	401	801.1	n/a	n/a	n/a
E5(BOO002-1)	16550	119	83.6	n/a	n/a	n/a
E3(RNG008-7)	23970	260	865.9	12330	193	494.2
E6(RNG009-7)	–	–	>360000	626693	2590	182510
wos21	12288	84	192.0	5928	64	84.4
RNG015-6	42443	326	881.3	20257	205	410.4
RNG023-6	–	–	>15000	137138	591	3880.8
RNG024-6	–	–	>15000	138154	589	3993.7

Table 1: Timing on a set of equality problems. Size lexicographic path ordering is used for all problems. Column "CPs" shows the total number of critical pair generated. Column "Rules" shows the number of rewrite rules kept. Time is measured in seconds on a SPARC-20. n/a means that the problem has no AC functions. - means that time out occurred.

It is particularly encouraging that we are able to prove E6, which says a ring with $x^3 = x$ is commutative. The problem has a long history. Wos had the following remarks on the problem, "if one succeeds in having a reasoning program prove this theorem, and it can be shown that the success is the result of a new technique, then one has solid evidence of the potential value of the new idea"[Wos, 1988]. Veroff obtained a proof of the problem using AURA [Veroff, 1981]. However, the input contained many additional clauses that facilitated the proof. Stickel was the first to prove the problem with a natural set of input equations [Stickel, 1984]. The proof took over 14 hours. Zhang and Kapur could find a proof in a few minutes using RRL [Zhang and Kapur, 1990]. Both [Stickel, 1984] and [Zhang and Kapur, 1990] used approaches based on AC unifications. The use of the cancellation law for additional group was important for them to get the proof efficiently.

6 Discussion

In term rewriting systems with AC functions, a term can have an exponential number of AC equivalent terms, and a rewrite rule can generate an exponential number of new rules when combined with AC axioms for critical pairs generation. A common approach for avoiding the combinatorial explosion is to represent all AC equivar lence terms as a single term. Flattened terms are used to represent terms equivalent up to associativity, and terms with arguments permuted are considered identical. AC unification algorithm is used to unify two flattened terms. Flattening breaks the well-foundedness of most termination orderings. Special AC termination orderings are needed to handle flattened terms.

On the other hand, the combinatorial explosion occurs only because all AC equivalent terms of a term are kept in a rewriting system. If they are simplified to a single normal form, the explosion will not occur. This is exactly the case for ground completion procedures: all ground AC equivalent terms can be rewritten to a normal form using AC axioms when a total termination ordering on ground terms is used. Our approach is based on the observation that a nonground completion procedure can be constructed by lifting a ground completion procedure, and thus avoid much of the redundancy in representing AC equivalent terms. Our approach does not need a special AC unification. It does not need a special AC termination ordering either. The latter facilitates the complete automation of the equational reasoning process. We used a single termination ordering for all of our test examples. We were also able to use the equational prover as a component in a general first order theorem prover.

There are some previous works on reducing unnecessary equational inferences, both for AC equational problems and equational problems in general[Zhang and Kapur, 1990; Bachmair *et a/.*, 1992]. Most of them are fundamentally different from our approach. They reduce the number of critical pairs generated by blocking out certain positions for overlapping two rewrite rules. Some of these works might be combined with our approach. It would also be interesting to study the applicability of our approach to other equational theories.

References

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

[Bachmair *et al.,* 1989] Leo Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In Hassan Ait-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures 2: Rewriting Techniques,* pages 1-30, New York, 1989. Academic Press.

[Bachmair *et al.,* 1992] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and basic strict superposition. In *Proceedings of the 11th International Conference on Automated Deduction,* pages 462-476, 1992.

[Birkhoff, 1935] G. Birkhoff. On the structure of abstract algebras. *Proc. Cambridge Philos. Soc,* 31:433-454, 1935.

[Boudet *et al.,* 1996] A. Boudet, E. Contejean, and C Marche. AC-complete unification and its application to theorem proving. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications,* pages 18-32, July 1996.

[Chang and Lee, 1973] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[Dershowitz and Jouannaud, 1990] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science.* North-Holland, Amsterdam, 1990.

[Dershowitz *et al.,* 1983] Nachum Dershowitz, J. Hsiang, N. Josephson, and David A. Plaisted. Associative-commutative rewriting. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* pages 940-944, August 1983.

[Domenjoud, 1991] E. Domenjoud. Ac-unification through order-sorted acl-unification. In *Proceedings of the 4th International Conference on rewriting techniques and applications.* Springer-Verlag, 1991.

[Kapur *et al.,* 1990] D. Kapur, G. Sivakumar, and H. Zhang. A new method for proving termination of ac-rewrite systems. In *Proc. of Tenth Conference on Foundations of Software Technology and Theoretical Computer Science,* pages 133-148, December 1990. Springer Verlag LNCS 472.

[Klop, 1992] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science,* volume 2, chapter 1, pages 1 - 117. Oxford University Press, Oxford, 1992.

[Knuth and Bendix, 1970] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra,* pages 263-297. Pergamon, Oxford, U.K., 1970.

[Lankford and Ballantyne, 1977] D. Lankford and A.M. Ballantyne. Decision problems for simple equational theories with commutative-associative axioms: Complete sets of commutative-associative reductions. Technical Report Memo ATP-39, Department of Mathematics and Computer Science, University of Texas, Austin, TX, 1977.

[Loveland, 1978] D. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, New York, 1978.

[Lusk and Overbeek, 1985] E. Lusk and R. Overbeek. Non-horn problems. *Journal of Automated Reasoning,* 1:103-114, 1985.

[McCune, 1990] William W. McCune. *OTTER 2.0 Users Guide.* Argonne National Laboratory, Argonne, Illinois, March 1990.

[McCune, 1996] W. McCune. Solution of the robbins problem, draft, 1996.

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

[Plaisted, 1993] D. Plaisted. Equational reasoning and term rewriting systems. In D. Gabbay, C. Hogger, J. A. Robinson, and J. Siekmann, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming,* volume 1, pages 273-364. Oxford University Press, 1993.

[Stickel, 1981] M.E. Stickel. A unification algorithm for associative-commutative functions. *Journal of the Association for Computing Machinery,* 28:423-434, 1981.

[Stickel, 1984] M Stickel. A case study of theorem proving by the knuth-bendix method:discovering that $x^3 = x$ implies ring commutativity. In *Proceedings of the 7th International Conference on Automated Deduction,* pages 248-258, 1984.

[Veroff, 1981] R.L. Veroff. Canonicalization and demodulation. Technical Report ANL-81-6, Argonne National Laboratory, Argonne, IL, 1981.

[Wos *et al.,* 1984] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications.* Prentice Hall, Englewood Cliffs, N.J., 1984.

[Wos, 1988] L. Wos. *Automated Reasoning: S3 Basic Research Problems.* Prentice Hall, Englewood Cliffs, N.J., 1988.

[Zhang and Kapur, 1990] H. Zhang and D. Kapur. Unnecessary inferences in associative-commutative completion procedures. *Mathematical Systems Theory,* 23:175-206, 1990.