Rewrite Methods for Clausal and Non-Clausal Theorem Proving

Jieh Hsiang
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794
U.S.A.

Nachum Dershowitz
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
U.S.A.

1. Abstract

Effective theorem provers are essential for automatic verification and generation of programs. The conventional resolution strategies, albeit complete, are inefficient. On the other hand, special purpose methods, such as term rewriting systems for solving word problems, are relatively efficient but applicable to only limited classes of problems.

In this paper, a simple canonical set of rewrite rules for Boolean algebra is presented. Based on this set of rules, the notion of term rewriting systems is generalized to provide complete proof strategies for first order predicate calculus. The methods are conceptually simple and can frequently utilize lemmas in proofs. Moreover, when the variables of the predicates involve some domain that has a canonical system, that system can be incorporated as rewrite rules, with the algebraic simplifications being done simultaneously with the merging of clauses. This feature is particularly useful in program verification, data type specification, and programming language design, where axioms can be expressed as equations (rewrite rules). Preliminary results from our implementation indicate that the methods are space-efficient with respect to the number of rules generated (as compared to the number of resolvents in resolution provers).

2. Introduction

Given an equational theory E, a *term rewriting system for* E is a finite set of rewrite rules $R=\{l_i \rightarrow r_i\}_{i=1}^{n}$ such that $\{l_i = r_i\}_{i=1}^{n}$ and E are equivalent (i.e., $s=t$ is true in $\{l_i=r_i\}_{i=1}^{n}$ if and only if $s=t$ in E). A term t is *reduced* using rule $l \rightarrow r$ if a subterm s of t, which is an instance of the left hand side l, is replaced by the corresponding instance of the right hand side r. A term s is *reachable* from t if t can be reduced to s after a finite number of reductions. A term is *irreducible* if no rule can be applied to it. We use t^* to denote an irreducible form of t. We call a term rewriting system *terminating* if there is no infinite sequence of reductions from any term, and *confluent* if for any distinct terms t, r, and s, if r and s are both reachable from t, then there is another term u which is reachable from both r and s. A rewriting system satisfying these two properties is called a *canonical term rewriting system*. It is easy to see that if R is a canonical term rewriting system, then every term has a unique irreducible form with respect to R. Thus, to check if equation $s =_E t$ is valid, all that needs be done is to reduce both s and t to their irreducible forms and see if they are identical.

A canonical system for an equational theory, if it exists, will not only have the same theoretical power as E, but can also eliminate the unmanageable search space often encountered in equational theorem proving. The reasons for such improvement are twofold: (1) the equations are used in one direction, so terms are "simplified" when rewrite rules are applied; (2) rules may be used in arbitrary order and no backtracking is needed (since all sequences of reductions lead to the same irreducible form).

Knuth & Bendix ([KnBe70]) gave a necessary and sufficient condition for a terminating term rewriting system to be confluent (and therefore canonical). They also presented a completion procedure for extending a non-canonical system to a canonical one without changing the original theory (although the method does not always terminate successfully). Their idea has been generalized by Lankford & Ballantyne ([LaBa77]) and Peterson & Stickel ([PeSt81]) to handle the case where some operators are commutative or associative and commutative.

3. A Canonical System for Boolean Algebra

Although very effective for solving word problems, the term rewriting method has been largely ignored by the theorem proving community on account of its relatively small problem domain. Therefore it would be desirable to extend this idea to handle the full first order theory. In order to achieve this goal, we need a canonical system for the logical connectives, Boolean algebra, and a complete strategy for the first order predicate calculus.

Attempts to find a canonical system for Boolean algebra have been reported in [Hul80] and [PeSt81], where conventional axioms for Boolean algebra were converted into rewrite rules and the AC-Completion Algorithm was used. Some stronger axioms, such as the Absorption Law, have also been used ([Hul80]). The Completion Algorithm, nevertheless, failed to terminate in all experiments (i.e. generated infinitely many rules) due to the well-known fact that the prime implicant representation of Boolean terms is not unique.

The problem of non-unique representation can be finessed, however, by choosing the right kind of operators. In our approach, we use "EXCLUSIVE-OR" in place of the usual operator "OR", and construct a canonical system for Boolean algebra with the help of this operator. (A system that simplifies Boolean expressions using EXCLUSIVE-OR was also discussed in [WaCo80].) The notion of EXCLUSIVE-OR was discussed by Stone ([St36]), who defined a *Boolean ring* (B,+,*,0) to be a ring which is idempotent with respect to *, i.e. $x*x=x$ for all x in B. He proved the following:

Theorem (Stone):

(1) *Every Boolean ring is commutative (i.e. $x*y=y*x$).*

(2) *Every Boolean ring is nilpotent with respect to + (i.e. $x+x=0$).*

(3) *Let $(B,+,*,0,1)$ be a Boolean ring with unit 1. Introduce the following operators :*

$$x\vee y=x+y+x*y$$
$$x\wedge y=x*y$$
$$\neg x=x+1$$

then $(B,\wedge,\vee,\neg,0,1)$ is a Boolean algebra. Conversely, given a Boolean algebra $(B,\wedge,\vee,\neg,0,1)$, define

$$a+b=(a\wedge\neg b)\vee(\neg a\wedge b)$$
$$a*b=a\wedge b$$

then the corresponding $(B,+,*,0,1)$ is a Boolean ring.

The operator '+' used by Stone later became XOR in switching theory. We use + for XOR, $\vee$ for OR, $*$ for AND, $\neg$ for NOT, 1 for TRUE, and 0 for FALSE throughout this paper. We say a term s is a *normal expression* of a Boolean term t if s is 1, 0 or $s=s_1+...+s_n$ where all s_i's are distinct, non-zero products of distinct positive literals. For example, a normal expression of $\neg x\vee y$ is $x*y+x+1$. It is not hard to prove that the normal expression of a Boolean term is unique up to permutation of arguments ([Hs82]).

A six-rule canonical system for Boolean rings can be obtained by executing the AC-Completion Algorithm (i.e. the Knuth-Bendix Completion Algorithm with a commutative-associative unification algorithm for finding critical pairs [PeSt81]) over the axioms of ring theory, the idempotence of $*$, and the nilpotence of +. By adding four more rules for transforming the usual Boolean connectives into Boolean ring forms, we have the following canonical term rewriting system for Boolean algebra:

$$\text{BA}\left\{\begin{array}{ll}
x\vee y \to x*y+x+y & R1\\
x\supset y \to x*y+x+1 & R2\\
x\equiv y \to x+y+1 & R3\\
\neg x \to x+1 & R4\\
x+0 \to x & R5\\
x+x \to 0 & R6\\
x*1 \to x & R7\\
x*x \to x & R8\\
x*0 \to 0 & R9\\
x*(y+z) \to x*y+x*z & R10
\end{array}\right.$$

This system, like any canonical term rewriting system, yields very simple proofs for the word problems of the underlying theory (in this case, the propositional calculus). As an example, the absorption law (i.e. $p\vee(p\wedge q)=p$) can be easily proved by observing that

$$p\vee(p\wedge q) \to p*(p*q)+p+p*q \to p*q+p+p*q \to 0+p \to p.$$

The dual system, based on $\vee$ and $\equiv$ rather than $\wedge$ and +, is also canonical and can be used in a similar fashion. This reduction process actually gives us an effective method for determining the validity problem of the propositional calculus. To be more precise, a Boolean term is valid if and only if its irreducible expression is 1; unsatisfiable if and only if its irreducible expression is 0; and satisfiable but not valid if and only if its irreducible expression is neither 1 nor 0. By the NP-completeness of the satisfiability problem ([Co71]), any systematic procedure for reducing Boolean terms to a canonical form requires exponential time in the worst case if $P \neq NP$.

4. A Complete Clausal Strategy for First Order Predicate Calculus

The "brute-force" reduction method described above is not the most convenient way of handling the validity problem for several reasons:

(1) For complicated statements with a lot of implications and disjunctions, the sentences may be expanded into long expressions by the procedure (using R1-R4 and R10) before most of the "simplification" reductions (using the rest of the rules) begin.

(2) The method cannot be easily generalized to effectively handle first order predicate calculus.

These problems can be circumvented by using a refutational proof technique. The scenario can be roughly described as follows: To prove that a first order sentence ϕ is valid, we first skolemize its negation, convert it into clause form $C_1 \wedge \cdots \wedge C_n$, add a rule $C_i + 1 \rightarrow 0$ for each clause C_i, then run the AC-Completion Algorithm over this set of rules until $1 \rightarrow 0$ is generated. The canonical system BA is used throughout to keep Boolean terms always in their normal form.

One potential problem with this approach is that both of the Boolean operators (+ and *) are AC and the unification algorithm used for finding critical pairs must deal with two AC operators at the same time. However, no AC-unification algorithm is presently known to be finite and complete (see [St81]). Fortunately, this problem does not arise in our method since we *do not* need to generate all critical pairs. In fact, by using the canonical system BA as inference rules, a new unification algorithm (BN-unification), which is considerably simpler than the AC-unification for one operator, can be achieved. Detail about this unification algorithm is in the appendices. The reader can treat the BN-unifications in the following definitions as AC-unifications and the results still hold.

Definition: *A Boolean term is an* **N-term** *if it is a conjunction of literals. A rule $l \rightarrow 0$ is an* **N-rule** *if l is an N-term.*

Definition: *Let $l_1[t] \rightarrow 0$ and $l_2 \rightarrow 0$ be two rules converted from clauses C_1 and C_2. If*

(1) *$l_2 \rightarrow 0$ is an N-rule,*

(2) *there is a (most general) BN-unifier σ such that $(ut)\sigma = (vl_2)\sigma$ (where u and v are two extra variables in the BN-unification not in l_1 and l_2),*

then

$$<(ul_1[0])\sigma, 0>$$

is an **N-critical pair** *of $l_1 \rightarrow 0$ and $l_2 \rightarrow 0$, with $(ul_1[t])\sigma$ as its superposition.*

In the above definition, $l_1[t]$ indicates that t is a nonvariable subterm of l_1. Note that since unifications are done only between N-terms, no unification involves the operator $+$.

The theorem proving strategy is the following:

```
N-Strategy: To prove that a sentence $\Phi$ is valid:
    Convert the (skolemized) negation of $\Phi$ into a set of clauses $S$;
    transform $S$ into a set of rewrite rules $R$ using $BA$;
    repeat
        Find a nontrivial N-critical pair $<t,0>$ between rules in $R$;
          ($<t,0>$ is nontrivial if $t$ is irreducible w.r.t. $BA$ and $R$, and $t \neq 0$)
        Convert $<t,0>$ into the rule $t \rightarrow 0$;
        $R := R \cup \{t \rightarrow 0\}$;
        Use $t \rightarrow 0$ to simplify and delete rules in $R$;
    until (no N-critical pair can be found) or ($1 \rightarrow 0$ is generated);
    if $1 \rightarrow 0$ is generated then return "proved"
                                      else return "consistent".
```

The N-strategy is essentially the same as the Knuth-Bendix Completion Algorithm except that (1) it looks only for N-critical pairs, and (2) it stops when a contradiction is generated. The canonical system BA is used throughout the process to reduce every term (such as the terms in the generated critical pairs) to its irreducible form. Like the Knuth-Bendix Completion Algorithm, the N-strategy can also produce different outcomes. It may:

(1) generate $1 \rightarrow 0$: In this case, the input clauses are inconsistent.

(2) generate finitely many rules and terminate: The clauses are consistent.

(3) generate infinitely many rules, i.e. never stop: The clauses are consistent.

Note that the case of abort (i.e. generation of a critical pair with incomparable terms) never happens in the N-strategy since one of the terms in any N-critical pair is always 0.

The completeness of the N-strategy is stated by the following theorem:

Theorem 4.1: *Given a set of clauses S in first order predicate calculus, S is inconsistent if and only if* $1 \rightarrow 0$ *can be produced using the N-strategy.*

We show how the strategy works by giving a proof for a verification condition of Hoare's FIND program:

$$\begin{aligned}
&\{\ \forall x,y(x \leq y \vee y < x) \wedge j < i \wedge m \leq p \leq q \leq n \\
&\ \wedge\ \forall x,y(m \leq x < i \wedge j < y \leq n \supset A[x] \leq A[y]) \\
&\ \wedge\ \forall x,y(m \leq x \leq y \leq j \supset A[x] \leq A[y]) \\
&\ \wedge\ \forall x,y(i \leq x \leq y \leq n \supset A[x] \leq A[y])\ \}\ \supset\ A[p] \leq A[q].
\end{aligned}$$

The first mechanical proof of this problem was reported in [SlNo73] where resolution and some special inference rules for partial ordering were used. The proof was completed after 62 resolvents were generated. In [GrNaOrPl82], experiments using locking resolution and a certain version of natural deduction (a simplified problem reduction format) were reported.

Locking generated 173 resolvents and stopped without obtaining a proof, the natural deduction strategy found a proof after producing 223 subgoals. Using the N-strategy, the negation of the sentence is converted into 9 rules:

$A[p] \le A[q] \to 0$ c1
$q \le n \to 1$ c2
$p \le q \to 1$ c3
$m \le p \to 1$ c4
$j < i \to 1$ c5
$(A[x] \le A[y])(i \le x)(x \le y)(y \le n) + (i \le x)(x \le y)(y \le n) \to 0$ c6
$(A[x] \le A[y])(m \le x)(x \le y)(y \le j) + (m \le x)(x \le y)(y \le j) \to 0$ c7
$(A[x] \le A[y])(m \le x)(y \le n)(j < y)(x < i) + (m \le x)(y \le n)(j < y)(x < i) \to 0$ c8
$(x \le y)(y < x) + (x \le y) + (y < x) \to 1$ c9

For convenience, we have omitted all the $*$'s between predicates. We have also moved all the 1's to the right hand sides. The latter change will not increase the number of critical pairs since such rules are not N-rules. On the contrary, it will improve the efficiency, since occurrences of the corresponding left hand sides can now be replaced by 1, and each $l_1 * l_2 * \ldots * l_n \to 1$ can be split into n new rules $l_1 \to 1, \ldots, l_n \to 1$.

By unifying p and q in c1 with x and y in c6, we have a critical pair $<(i \le p)(p \le q)(q \le n), 0>$. However, this critical pair is not irreducible since subterms $p \le q$ and $q \le n$ can be reduced to 1 by c3 and c2. Thus we produce a rule:

$i \le p \to 0.$ c10

By the same token, c1 and c7 yield a new rule

$q \le j \to 0,$ c11

while c1 and c8 create

$(j < q)(p < i) \to 0.$ c12

Now c10, c11, and c12 are the N-rules we can use. The only nontrivial critical pair between c10 and the rest of the set is with c9. By unifying i and p with x and y, we get

$p < i \to 1.$ c13

Rule c13 simplifies c12 into

$j < q \to 0.$ c12'

And c11, similar to c10, also produces a new rule with c9:

$j < q \to 1.$ c14

A contradiction is reached immediately by c12' and c14. Note that the proof is obtained after producing only 6 rules (or 5 rules to be precise, since c12' is simplified from c12), and that this proof is exhaustive!

4.1. Comparisons with Resolution

The N-strategy is in itself similar to the all-negative resolution method (i.e. one of the parents of a resolvent must contain only negative literals). When the constant 1 is placed on the right hand side (as it was in the previous example), the method also contains some features of unit-resolution. Thus, our system works well in some cases, such as Horn clauses, where the all-negative strategy does not. In general, however, the efficiency of the N-strategy, as a purely syntactic method, should not be drastically different from that of other resolution strategies. Nevertheless, the example we showed seems to reveal a dramatic gain in efficiency using the N-strategy over resolution or natural deduction: achieving a proof after generating 6 rules as opposed to generating 173 resolvents without a proof (locking resolution) and generating 223 subgoals (natural deduction). The reason for such a surprising improvement is certainly worth investigating.

The basic difference between N-strategy and other resolution strategies lies in the adoption of the rewriting (reduction) method. To be more precise, the rewriting method *requires* everything to be reduced as much as possible. For example, when the critical pair $<(i \le p)(p \le q)(q \le n),0>$ was generated, it was not converted immediately into a rule, since the first term can still be reduced using rules c2 and c3. When the critical pair was finally reduced to $<i \le p,0>$ and made into a rule, all the possible intermediate "resolvents" (in this case, $(i \le p)(p \le q)(q \le n) \rightarrow 0$, $(i \le p)(p \le q) \rightarrow 0$, and $(i \le p)(q \le n) \rightarrow 0$) were excluded. In a resolution strategy, however, all those resolvents are kept, and even if subsumption is included as part of the strategy, the resolvents will not be deleted until some subsuming clause is generated. In a breadth-first implementation (which seems to be the most commonly used technique), such a subsuming clause usually will not be generated early. Subsequently, the more useful resolvents (such as c10 in our example) are not generated until much later.

5. A Non-Clausal Strategy

Since the canonical system for Boolean algebra always gives us an irreducible form for any Boolean term, it seems superfluous to insist on converting sentences into clause form. In fact, by lifting the restriction of generating only N-critical pairs, non-clausal strategies can be obtained:

A non-clausal strategy: *To prove a sentence Φ is valid:*

(1) *Skolemize $\neg\Phi$; assume that the result is $M_1 \wedge M_2 \wedge \cdots \wedge M_n$ (the M_i's are not necessarily clauses).*

(2) *Add rewrite rules $M_1 \rightarrow 1$, $M_2 \rightarrow 1, \ldots, M_n \rightarrow 1$.*

(3) *Run the AC Knuth-Bendix Completion algorithm on the set of rules, using BA as additional simplification rules, until no more critical pairs can be found or $1 \rightarrow 0$ is generated.*

(4) *If $1 \rightarrow 0$ is found, then return "proved"; otherwise return "consistent".*

The AC-unification algorithm needed for finding critical pairs is, as in the N-strategy, weaker than the full AC-unification, since no variable will appear as an argument of an AC-operator. Nevertheless, both AC-operators ($*$ and $+$) are still involved in the unification, and the completeness of such a restricted algorithm remains unknown at this moment. The method can be proven complete for first order predicate calculus (as a direct consequence of the Theorem in [HuHu80]) once the completeness problem of the unification is resolved affirmatively.

We demonstrate the use of the strategy by the following example. Let Φ be:

$$\forall x[A(x)\equiv B(x)]\supset [\forall xA(x)\equiv\forall xB(x)].$$

The Skolemized negation of Φ is:

$$[A(z)\equiv B(z)]\wedge[A(x)\vee B(y)]\wedge[\neg B(a)\vee\neg A(b)]\wedge[B(a)\supset B(y)]\wedge[A(b)\supset A(x)],$$

where a and b are new Skolem constants. Since the first $\equiv$ remains intact, we can make use of the rule R4: $x\equiv y\rightarrow x+y+1$ in BA. The formulas, when converted into rules, are as follows:

$$A(z)+B(z)\rightarrow 0 \qquad r1$$
$$A(x)B(y)+A(x)+B(y)\rightarrow 1 \qquad r2$$
$$B(a)A(b)\rightarrow 0 \qquad r3$$
$$B(a)B(y)+B(a)\rightarrow 0 \qquad r4$$
$$A(b)A(x)+A(b)\rightarrow 0 \qquad r5$$

From r1 and r2, we have a superposition $A(z)B(z)+A(z)+B(z)$ by unifying both x and y with z, and a critical pair $<A(z)B(z),1>$. Since $M_1\cdots M_n=1$ can be split into equations $M_1=1,\ldots, M_n=1$, instead of converting the critical pair into one rule, we have:

$$A(z)\rightarrow 1 \qquad r6$$
$$B(z)\rightarrow 1. \qquad r7$$

Rules r6 and r7 mean that every instance of $A(z)$ and $B(z)$ in the set of rules can be replaced by 1. Therefore, all of the original five rules are deleted, and r3 becomes:

$$1\rightarrow 0,$$

the contradiction.

6. Complete Strategies for First Order Built-in Theories

In this section we generalize the previous results to a larger class of problems, namely, the first order calculus whose predicates involve some special domains. We further assume that there exist canonical term rewriting systems for such domains. Such problems are common in practice, such as theorems in group theory (or other algebraic structures), properties of data types, etc. The conventional method for dealing with such problems is to treat the axioms of the domain as extra clauses and equations, and use resolution and/or paramodulation. This method is very inefficient in general. Another possible solution is the following: since the rewriting method is very effective in manipulating terms, why not use resolution on the clauses and the rewriting method (with the canonical system for the theory) on the arguments? Feasible as it sounds, this approach is not complete ([La75]). Research along these lines has been conducted by Plotkin [Pl73], who merged the domain axioms into the unification algorithm (the same idea was adopted later by [PeSt81] and [LaBa77] for extending the Knuth-Bendix method), and Slagle [Sl74], Lankford [La75], and Lankford-

Ballantyne [LaBa79] who introduced the concept of "narrowing" to find useful instances of the arguments in the clauses.

The problem can be easily solved, with the help of BA, by using only the term rewriting method. Henceforth, S stands for a set of rewrite rules converted from a set of clauses, and R stands for a (canonical) rewriting system for the domain theory T.

Definition: *$l \to r$ is an* **RN-rule** *if it is an N-rule in S or it is a rule in R.*

Definition: *Given a rule $t_1[s] \to 0$ in S and an RN-rule $l \to r$, if either*
(i) $l \to r$ is an N-rule and there is an N-critical pair $<t_2,0>$ between the two rules,
or (ii) $l \to r$ is in R and there is a most general unifier σ such that $s\sigma = l\sigma$ (let $t_2 = (t_1[r])\sigma$),
then $<t_2,0>$ is an **RN-critical pair.**

In the above definition, the subterm s in $t_1[s]$ is a nonvariable subterm of an argument of one of the predicates in t_1. In case (ii), the most general unifier between an argument of a clause and a rule in R is obtained by applying whatever unification algorithm is appropriate for the theory T. The only difference between the making of an N-critical pair and an RN-critical pair is that in the latter, superposition is also allowed between a rule about the formulas and one about the domain axioms. The N-strategy can also be generalized to the RN-strategy by requiring the generation of RN-critical pairs:

```
RN-Strategy:
Given sets S and R as described above,
repeat
    Find a nontrivial RN-critical pair $<t,0>$ between rules in $R \cup S$;
    Convert $<t,0>$ into the rule $t \to 0$;
    $S := S \cup \{t \to 0\}$;
    Use $t \to 0$ to simplify and delete rules in S;
until (no RN-critical pair can be found) or ($1 \to 0$ is generated);
if $1 \to 0$ is generated then return "proved"
                           else return "consistent".
```

The possible outcomes of the algorithm are similar to those for the N-strategy, and the method is also complete for its target problems:

Theorem 6.1: *Given a set of clauses S and a canonical system R, $S \cup R$ is E-unsatisfiable if and only if $1 \to 0$ can be produced from $S \cup R$ using the RN-strategy.*

To demonstrate how RN-strategy works, we proof the following problem in group theory. To save space, we list only the first three rules of the ten-rule canonical system for group theory ([KnBe70]) that is used in the proof.

$xe \rightarrow x$		r1
$(xy)z \rightarrow x(yz)$		r2
$xx^{-1} \rightarrow e$		r3
...		
$P(a) \rightarrow 0$	(*i.e.* $\neg P(a)$)	c1
$P(xb^{-1}) + 1 \rightarrow 0$	(*i.e.* $P(xb^{-1})$)	c2

The RN-strategy found a proof after generating five rules. Among them, the following three derives the proof directly:

$P(x(yb^{-1})) + 1 \rightarrow 0$	(r2 and c2)	c3
$P(x) + 1 \rightarrow 0$	(c3 and r3)	c4
$1 \rightarrow 0$	(c4 and c1)	c5

The other two generated rules are $P(b^{-1}) + 1 \rightarrow 0$ and $P(e) + 1 \rightarrow 0$.

The theorem (and method) can also be generalized to the case where R is an AC-canonical system (such as the theory of abelian groups).

6.1. Extensions and Discussions

The RN-strategy will improve the efficiency (as compared with resolution and paramodulation type provers) in at least the following ways:

(1) The clauses for describing the axioms of the domain are eliminated; thus, the number of clauses and resolvents is reduced.

(2) Replacements of terms in the literals using the equational axioms are done strictly in one direction, and no elaborate arrangement of choice of proper subterms is needed since every term has a unique irreducible form; thus, the search space is reduced considerably.

(3) Once a canonical system for the domain theory is found, it can be reused each time a theorem is to be proved, and no more work on the domain need be done. The savings are most significant when several theorems are to be proved at the same time. In resolution (paramodulation) theorem proving, however, resolvents among the axioms as well as those between the axioms and the similar clauses in the theorems will usually be generated every time.

(4) The useful lemmas for T that are generated by the Completion Algorithm are fully utilized.

Experiments comparing the performance of RN and N strategies have been conducted ([HsJo82]). As an example, for the problem:

If S is a nonempty subset of a group and $x,y \epsilon S \supset xy^{-1} \epsilon$S, then $x \epsilon S \supset x^{-1} \epsilon S$,

the N-strategy generated 18 rules while the RN-strategy only generated 6. (For comparison, the locking resolution generated 76 resolvents and the simplified reduction format, SPRF, generated 52 subgoals [GrNaOrPl82].) This seems to show that the method is more efficient when used with a canonical system for the domain theory.

The RN-strategy can also be modified into a non-clausal strategy in the same fashion as in the previous section. As before, the completeness of the strategy also depends on the finiteness and completeness of the employed unification algorithm.

The RN-strategy does not apply when a canonical system for the domain theory does not exist. This requirement can be relaxed somewhat ([Hs82]) by generating RN-critical pairs and critical pairs between rules in R (a noncanonical system for the domain theory) simultaneously. Such a strategy is called the RN^+-strategy, and it is a complete strategy as long as no incomparable critical pair between rules in R can be produced. A method suggested by Lankford ([La75]) for (partially) handling incomparable critical pairs can be incorporated into RN^+-strategy and increases its power to some extent. These extensions, however, are still weaker than resolution+ paramodulation since they cannot always handle equations in non-unit clauses. An obvious remedy is to employ the modification method of Brand ([Br75]) for non-unit equations. Unfortunately, this trivial solution will undoubtedly increase the search space considerably and thus destroy the major advantage we gained by using the rewrite method. Finding an effective way to deal with equations in non-unit clauses is something worth looking into.

Acknowledgement

The authors would like to thank David Plaisted for many stimulating conversations during the development of this work, Dallas Lankford for his lively correspondence and encouragement, Francois Fages for his comments, and Alan Josephson for implementing TeRSe, a theorem prover that contains some of the strategies discussed here.

7. References

[Br75] Brand, D., Proving theorems with the modification method. *SIAM J. of Computing,* Vol 4, 1975.

[Co71] Cook, S.A., The complexity of theorem-proving procedures. *3rd. ACM Symp. on Theory of Computing,* pp151-158, 1971.

[GrNaOrPl82] Greenbaum, S., Nagasaka, A., O'Rorke, P., Plaisted, D., Comparison of natural deduction and locking resolution implementations. *6th Conf. on Automated Deduction,* Lecture Notes in CS, No. 138, 1982.

[Hs82] Hsiang, J., Topics in automated theorem proving and program generation. Ph.D. Thesis, U. of Illinois at Urbana-Champaign, 1982.

[HsJo82] Hsiang, J. & Josephson, N.A., A term rewriting theorem prover. Unpublished manuscript, 1982.

[Hul80] Hullot, J.-M., A catalogue of canonical term rewriting systems. Report CSL-113, SRI International, 1980.

[HuHu80] Huet, G. & Hullot, J.-M., Proofs by induction in equational theories with constructors. *21st FOCS,* 1980.

[KnBe70] Knuth, D.E. & Bendix, P.B., Simple word problems in universal algebra. *Computational Problems in Abstract Algebra,* J.Leech Ed. Pergamon Press, Oxford, 1970.

[KoHa69] Kowalski, R. & Hayes, P., Semantic trees in automatic theorem proving. *Machine Intelligence 4* Meltzer & Michie eds. pp87-101, 1969.

[La75] Lankford, D.S., Canonical inference. Report ATP-32, Univ. of Texas at Austin, 1975.

[LaBa77] Lankford, D.S. & Ballantyne, A.M., Decision procedure for simple equational theories with commutative-associative axioms. Report ATP-39, Univ. of Texas at Austin, 1977.

[LaBa79] Lankford, D.S. & Ballantyne, A.M., The refutation completeness of blocked permutative narrowing and resolution. *4th Conf. on Automated Deduction,* 1979.

[PeSt81] Peterson, G.E. & Stickel, M.E., Complete sets of reductions for some equational theories. *J.ACM* Vol 28, pp233-264, 1981.

[Pl73] Plotkin, G., Building in equational theories. *Machine Intelligence 7* Meltzer & Michie Eds., pp73-90

[Ro65] Robinson, J.A., A machine oriented logic based on resolution principle. *J.ACM* Vol 12, pp23-41, 1965.

[Sl74] Slagle, J., Automated theorem proving with simplifiers, commutativity, associativity. *J.ACM* Vol 21, pp622-642, 1974.

[SlNo73] Slagle, J. & Norton, Experiments with an automatic theorem prover having partial ordering inference rules. *C.ACM* Vol 16, pp682-688, 1973.

[St81] Stickel, M.E., A unification algorithm for associative-commutative functions. *J.ACM* Vol 28, pp233-264, 1981.

[St36] Stone, M., The theory of representations for Boolean algebra. *Trans. AMS* Vol 40, pp37-111, 1936.

[WaCo80] Watts, D.E. & Cohen, J.K., Computer implemented set theory. *American Mathematical Monthly,* Vol 87, No. 7, pp557-560.

8. Appendix

8.1. BN-Unification

The unification problem for Boolean terms is considerably simpler than AC-unification since (1) a Boolean term is a sum of products of predicates and different predicates symbols cannot unify with each other, and (2) identical predicates never appear twice in a term (from the idempotence of $*$). However, in order to achieve completeness for the N-strategy, a variable needs to be attached to each term (similar treatment is also needed for obtaining canonical systems for AC-theories, see [PeSt81]). Since the N-strategy only requires unifications between N-terms, we may reformulate the unification problem into the following:

Given two (irreducible) Boolean N-terms s' and t', find the complete set of most general unifiers between $s=us'$ and $t=vt'$, where u and v are new variables.

We call this process *BN-unification.*

We now give a straightforward BN-unification algorithm. The algorithm has as inputs s' and t', which are two seperated (i.e. they do not share common literals) N-terms. The output of the algorithm is Σ, the complete set of most general unifiers. For simplicity of notation, we always assume that the variables in the predicates are properly renamed before substitution.

```
Proc $UNIFY(s',t')$:
    $\Sigma := \phi$;
    $s := us'$;            /* $u$ and $v$ are the extra variables */
    $t := vt'$;
    call $UNIFY1(s,t,\phi)$;
    return $\Sigma$.

Proc $UNIFY1(s,t,unifier)$:
    separate $s$ and $t$, assume that the resulting terms are:
        $s=us_1...s_n$ and $t=vt_1...t_m$;
    $\Sigma := \Sigma + unifier \cup \{u \leftarrow wt_1...t_m, v \leftarrow ws_1...s_n\}$;
    for $i=1$ to $n$ do
        for $j=1$ to $m$ do
            if $\exists\sigma$ s.t. $s_i\sigma=t_j\sigma$        /* $\sigma$ is a unifier between literals */
                then $UNIFY1(s\sigma,t\sigma,unifier \cup \sigma)$.
```

The unifier "σ" in the algorithm is a most general unifier between the target literals, and is *not* a unifier between the N-terms. One way to reduce the number of loops is to sort the literals in the terms beforehand. Then literals with the same predicate symbols will be grouped together and most of the useless unification attempts between different predicates will be eliminated.

As an example, terms $<s',t'> = <P(x,y)P(y,z)P(z,x),P(a,b)>$ have the following four most general unifiers: $\{u \leftarrow wP(a,b), v \leftarrow wP(x,y)P(y,z)P(z,x)\}$, $\{x \leftarrow a, y \leftarrow b, u \leftarrow w, v \leftarrow wP(b,z)P(z,a)\}$, $\{y \leftarrow a, z \leftarrow b, u \leftarrow w, v \leftarrow wP(x,a)P(b,x)\}$, and $\{z \leftarrow a, x \leftarrow b, u \leftarrow w, v \leftarrow wP(b,y)P(y,a)\}$. Note that in this example there are actually two types of variables and two kinds of unifications. The variables u, v, and w are Boolean variables; x, y, and z are variables in the arguments of the predicates, and the unification between the predicates is the conventional unification.

The extra variable w, which is added to achieve the most general unifiers, will always be replaced by 1 when used in the theorem proving strategies.

8.2. Proofs of Theorems

In order to present the proofs effectively, we use equations $l=0$ instead of rules $l \rightarrow 0$. We also need a modified notion of the semantic tree ([Ro65], [KoHa69]) for the proof.

Definition: *The* **E-atom set** *of a set of clauses S is the set $\{P(a_1,...,a_n)=0, P(a_1,...,a_n)=1 : a_i$ is in the Herbrand universe of S and P is a positive literal in $S\}$.*
The elements in the E-atom sets are called the **E-atoms.**

For example, the E-atom set of $\{P(a), \neg Q(fx)\}$ is

$$\{P(a)=0, P(a)=1, Q(a)=0, Q(a)=1, P(fa)=0, P(fa)=1, ...\}.$$

An *E-semantic tree* over a set of clauses S is a conventional semantic tree except that the two arcs of a node are labelled with complementary E-atoms from the E-atom set of S (i.e. $L=0$ and $L=1$ instead of L and $\neg L$). A node N of an E-semantic tree is a *failure node* if there is a ground instance $G=0$ of $C^*=0$ (where C is a clause in S, and C^* is the irreducible form of C with respect to BA), whose normal expression becomes 1=0 when each literal L in G is replaced by 0 (resp. 1) if $L=0$ (resp. $L=1$) labels an arc on the path from the root to N. An E-semantic tree is *closed* if every leaf is a failure node. Other definitions concerning the semantic tree follow in a similar way. The Herbrand Theorem is also true for the new definitions:

Theorem (Herbrand): *A set of clauses S is unsatisfiable if and only if every complete E-semantic tree of S has a finite closed subtree.*

8.3. Completeness of N-strategy

We now prove the completeness of the N-strategy (Theorem 4.1). The 'if' part is easy since every reduction step is sound. We prove the 'only if' part by induction on the size of the closed E-semantic tree. For the simplicity of notation, we prove only the propositional case. The proof can be generalized to first order predicate calculus without difficulty.

Induction basis: If the E-semantic tree has only one node, then the node must be 1=0 and we are done.

Induction step: We order the closed E-semantic tree in such a way that the right arc is always positive, i.e. $L=0\bigwedge L=1$. Since each leaf (failure node) of the closed E-semantic tree is an instance of the canonical form of a clause in S, the rightmost leaf must be an equation with an N-term as the left hand side (which corresponds to an all-negative clause in S). Let $LL_1...L_n=0$ be the leaf of the rightmost path with $L=1$ as the last arc (see Figure 1). If every leaf in its neighboring subtree does not contain the literal L, the last fork labelled with L can be eliminated and the E-semantic tree will be "shrunk" to become Figure 2 (with D' the same as D). Then by the induction hypothesis, we are done. So the problem lies in the case where some of the leaves in D do contain L. Let $L*s_1+L*s_2+...+L*s_m+t=0$ be such a leaf where s_i's are N-terms and t does not contain L. By the BN-unification, $L*s_1$ and $LL_1...L_n$ have a most general unifier which unifies the

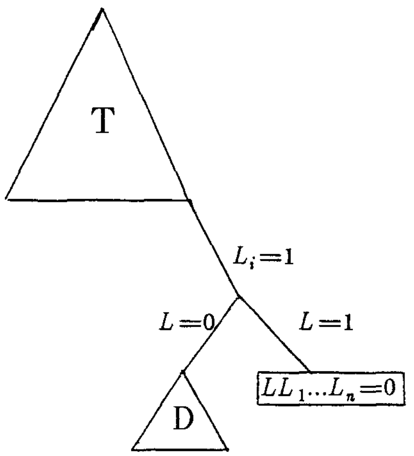

Figure 1
The E-semantic tree

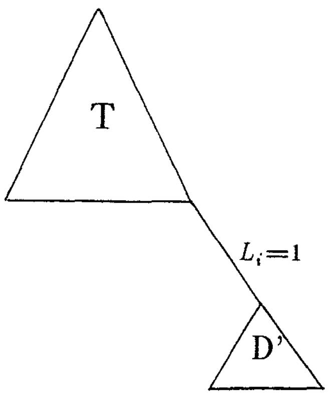

Figure 2
The E-tree after deleting the L branch

extra variable v of $L*s_1$ with part of $L_1...L_n$ (or with all of $L_1...L_n$ if none of the L_i's occur in s_1). Without loss of generality, we assume that only the literals L_{i+1} through L_n appear in s_1, and v is unified with $L_1...L_i$. We then have a superposition $LL_1...L_i*s_1+...+LL_1...L_i*s_m+L_1...L_i*t$, and a new equation (from the corresponding critical pair) $LL_1...L_i*s_2+...+LL_1...L_i*s_m+L_1...L_i*t=0$. Note that (1) this step is a legitimate rule-generating step in the N-strategy, and (2) the first subterm $L*s_1$ has been eliminated. It is not hard to show that it takes at most n superpositions to eliminate the literal L completely from this leaf and that the new failure node thus becomes $L_{i_1}...L_{i_j}*t=0$ for some j and $i_1,..., i_j$. After applying the above process to every equation in D with literal L, the subtree D becomes a new tree D' without the occurrence of L in any of its leaves. The fork labelled with L can then be eliminated and the E-semantic tree is also shrunk accordingly (as shown in Figure 2). By the induction hypothesis, we are done.

8.4. Completeness of RN-strategy

Now we prove Theorem 6.1. The 'if' part is true since all the reduction steps are sound. For the 'only if' part, we proceed by induction on the size of the E-semantic tree. As in the proof of the completeness of the N-strategy, we use equations instead of rules. First, let us look at the case of ground formulas. Let B=$\{t[a_1,...,a_m]=0\}$ be an inconsistent set of ground instances of rules in S, where a_1 to a_m are all the arguments in the literals of t. We may convert B into B'=$\{t[a_1^*,...,a_m^*]=0\}$ where a_i^* is the irreducible form of a_i. (The reductions can be done by finding RN-critical pairs between rules in R and rules in B. Note that a_i^* is unique since R is canonical.) By Theorem 4.1 we know that 1=0 can be derived from B' using BA. Let Figure 3 be an E-semantic tree corresponding to such a proof. We assume, as in the proof of Theorem 4.1, that the right branch of a node is always positive. Let $L_1L_2...L_nP(a_1^*,...,a_m^*)=0$ be the leaf of the rightmost path in the E-semantic tree. Take all the leaves of its neighboring subtree D that contain $P(a_1^*,...,a_m^*)$ and perform

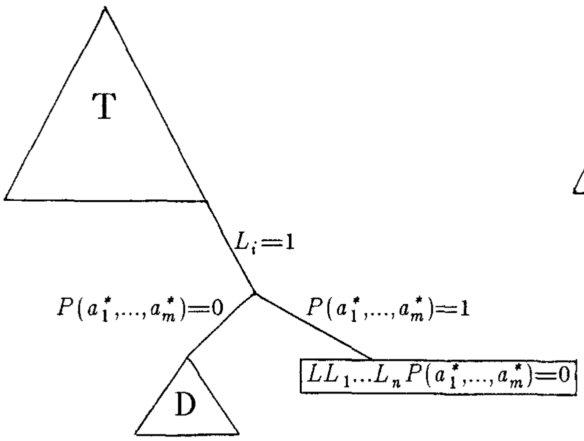

Figure 3
The E-semantic tree

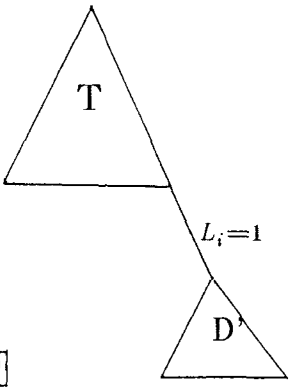

Figure 4
The E-tree after deleting the $P(a_1^*,...,a_m^*)$ branch

superpositions between each of these leaves and $L_1L_2...L_nP(a_1^*,...,a_m^*)=0$ as in the proof of Theorem 4.1. Then the literal $P(a_1^*,...,a_m^*)$ will be eliminated from the tree D, and the branch labelled $P(a_1^*,...,a_m^*)$ can be eliminated (see Figure 4) accordingly. Then by the induction hypothesis, we are done.

The following problem, however, may appear when lifting the above argument to the corresponding nonground formulas of B': There might be some $P(b_1^*,...,b_m^*)t_1+t_2=0$ where $b_i^*=a_i^*$, but $b_i \neq a_i$. To be more precise, if $P(s_1,..,s_m)$ is the nonground literal corresponding to $P(b_1,...,b_m)$ and $P(t_1,...,t_m)$ is the one corresponding to $P(a_1,...,a_m)$, then $P(b_1^*,...,b_m^*)=P(a_1^*,...,a_m^*)$ does not guarantee that s_i and t_i are unifiable. This is exactly the difficulty for which Slagle and Lankford devised additional techniques (such as narrowing).

We claim that by superimposing rules in R on $P(t_1,...,t_m)$ (i.e., finding RN-critical pairs between rules in R and S), some $P(t'_1,...,t'_m)$ which has $P(a_1^*,...,a_m^*)$ as an instance will eventually be generated. If this is true, let $P(t'_1,...,t'_m)$ and $P(s'_1,...,s'_m)$, corresponding to $P(a_1^*,...,a_m^*)$ and $P(b_1^*,...b_m^*)$ respectively, be such generated literals. They can certainly unify with each other and produce the corresponding critical pair we desire. Thus, it remains only to prove the claim.

For simplicity, we use $P(t_1,...,t_m)\rightarrow 0$ as a rule.

Lemma 6.2: *Given a canonical system R and an extra rule $P(t_1,...,t_m)\rightarrow 0$ with $P(a_1,...,a_m)\rightarrow 0$ as a ground instance, by performing the RN-strategy on $R \cup \{P(t_1,...,t_m)\rightarrow 0\}$, we will generate a rule $P(t'_1,...,t'_m)\rightarrow 0$ which has $P(a_1^*,...,a_m^*)$ as a ground instance.*

Proof: We proceed by induction on the total number (n) of steps needed to reduce all a_i to a_i^*. If n=0, nothing need be proved.

If $a_i \neq a_i^*$, there must be a subterm c of a_i which is a ground instance of the left hand side of some rule $l\rightarrow r$ in R. We denote this a_i by $a_i[c]$ and the corresponding nonground term by $t_i[l']$ (where l' is an instance of l). $P(t_1,...,t_i[l'],...,t_m)\rightarrow 0$ and $l\rightarrow r$ will produce a critical pair $<P(t_1,...,t_i[r'],...,t_m),0>$ and, thus, a new rule $P(t_1,...,t_i[r'],...,t_m)\rightarrow 0$. Since the corresponding ground instance of $t_i[r']$ is at least one reduction step closer to a_i^* than a_i, by the induction hypothesis, we are done.

We thus have completed lifting the argument to the nonground case and finished the proof of Theorem 6.1.