4 33 Basic Test Problems: A Practical Evaluation of Some Paramodulation Strategies

William McCune[1]
Argonne National Laboratory

4.1 Introduction

Many researchers who study the theoretical aspects of inference systems believe that if inference rule A is complete and more restrictive than inference rule B, then the use of A will lead more quickly to proofs than will the use of B. The literature contains statements of the sort "our rule is complete and it heavily prunes the search space; therefore it is efficient".[2] These positions are highly questionable and indicate that the authors have little or no experience with the practical use of automated inference systems. Restrictive rules (1) can block short, easy-to-find proofs, (2) can block proofs involving simple clauses, the type of clause on which many practical searches focus, (3) can require weakening of redundancy control such as subsumption and demodulation, and (4) can require the use of complex checks in deciding whether such rules should be applied. The only way to determine the practical value of inference rules and search strategies is to experiment on problems in which long-term target users are interested.

In this chapter we present a new theorem prover for equational logic, a set of 33 equational theorems for evaluating paramodulation strategies, a large set of experiments with several paramodulation strategies, and two equational proofs in Robbins algebra. The new theorem prover, EQP, includes associative-commutative unification and is restricted to equational logic, but in many other ways it is similar to our production theorem prover Otter[10]. The 33 equational theorems, which are mostly about lattice-like and group-like structures, are taken from a recent interdisciplinary study on application of Otter to problems in equational logic [11]. The experiments are with basic paramodulation, blocked paramodulation, ordered-instance paramodulation, functional subsumption, a heuristic for eliminating associative-commutative unifiers, and methods for directing the search. The two Robbins algebra theorems, which involve the hypotheses $\exists C \exists D\ (C + D = C)$

[1]Supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

[2]This is a very general statement. We are not referring in particular to the paramodulation strategies on which we focus in this chapter

and $\exists C \exists D\ (n(C + D) = n(C))$, were previously known, but we believe the proofs we present here are the first equational proofs and the first ones found by computer.

4.2 The Test Problems

Our 33 test problems are taken from the recent monograph *Automated Deduction in Equational Logic and Cubic Curves* [11], which contains a collection of first-order theorems proved by Otter [10]. The theorems are in a narrow area: from the syntactic view, all are equational, in languages with small sets of symbols, and most have small sets of small equations; from the semantic view, most are about simple lattice-like algebras and simple group-like algebras. Otter can already prove most of the theorems, so this work sheds little light on new kinds of theorem (such as theorems involving rich theories or complicated definitions) for which the paramodulation strategies of these experiments might be useful. Nevertheless, we believe the theorems are a good set for evaluating paramodulation strategies because they are real and nontrivial theorems that arose in practice when a mathematician (R. Padmanabhan) was studying algebraic structures with equational logic.

From the set of theorems in [11], we have excluded those involving the geometry inference rule $=\!(gL)\!\Rightarrow$, those with nonequational hypotheses such as cancellation, those that required extremely specialized strategies to find a proof, and most of those that were proved by Otter in less than a second or two. Also, many of the theorems in [11] have multiple goals (nonunit denials), and in those cases we used only the most difficult goal. That left 33 equational theorems[3] (each with a unit goal), of which 13 have at least one associative commutative (AC) operation. The names we use here are similar to those in [11], with the minor modification that if we use just one goal of a multiple-goal theorem, we append a lower-case letter to the name: "a" means that we use the first goal, "b" the second, and so on.

This brings us to a dirty part of these experiments—the *max-weight* parameter. It is a practical fact that Otter and EQP need a limit on the size of retained clauses for most nontrivial searches. Otherwise, most retained clauses never enter the search, wasting memory and wasting the time used trying to rewrite them with new demodulators. (We assume that most other programs that generate and retain a lot of clauses have the same problem.) When faced with a new conjecture or theorem, we generally start with no limit or with a limit that was successful for a similar problem. We usually learn quickly if the limit is a bad choice (the program runs out of tasks to do if the limit is too small, and it runs out of memory if the

[3] The number is a coincidence, really, Larry.

limit is too big), and we iterate a few times to find a limit that results in a well-behaved search.[4] We used this method for the problems in [11], and we simply copy the limits for these experiments. The danger in doing this is that a good limit for one strategy is not necessarily acceptable for another strategy; for example, a paramodulation restriction may require the use of a longer clause. Even reordering the search with the *pick-given-ratio* (see 4.5.1, p. 8) parameter may require the use of a higher limit, for example, if the generation of a greatly-simplifying demodulator is delayed. Therefore, these experiments are biased somewhat toward a simple starting strategy.

Otter and EQP (without AC unification) use a total order on function and constant symbols (which, by the lexicographic recursive path order, induces a partial simplification order on terms and a total simplification order on ground terms) to orient equations and decide which are to be demodulators. The default symbol order is $\mathit{constants} \prec \mathit{high\text{-}arity} \prec \cdots \prec \mathit{binary} \prec \mathit{unary}$, and within arity, the lexicographic ASCII ordering is used. We override the default order for the same cases and in the same ways as in [11]: In two cases (D-BA-2a and SD-2a), the purpose is to unfold defined terms, and in the cases MFL-1, MFL-2, and MFL-3, the purpose is to eliminate intuitively undesirable Skolem functions. For the experiments with AC unification, a different term order was used (see 4.5.2, p. 9).

When we write "non-AC problems", we mean all 33 problems with ordinary unification, with AC axioms included for the 13 problems that have AC function symbols; and when we write "AC problems", we mean AC unification with the 13 problems that have AC function symbols. The test problems are listed in the Appendix, starting on page 33.

4.3 Our Paramodulation Paradigm

Through the years, the members of the Argonne's automated reasoning group have invented and refined many inference rules and strategies for automated deduction in first-order logic with equality. In the early days of automated theorem proving, Larry Wos recognized the need for special treatment of equality if the field was to be of practical use to mathematicians. As a result, he invented demodulation [22] and then paramodulation [16]. Larry's continuous emphasis on experimentation

[4] We have attempted to automate the selection of a good limit with Otter's *control-memory* flag [12]. This flag is part of Otter's autonomous mode, which is used when the user has only one chance to prove a theorem, i.e., in automated theorem proving competitions, in demonstrations, for novice users, or when we're too lazy to make any decisions. For practical work, however, we find an appropriate limit by iteration.

with nontrivial problems has led the Argonne group to many advances in practical automated reasoning.

Our "starting"[5] paramodulation inference system in both Otter and EQP has the following properties.[6]

- A term ordering is used to compare terms. If one side of an equation is greater than the other, the equation is arranged so that greater side is on the left, and the equation becomes a demodulator; it is then said to be *oriented*.
- When paramodulating from (respectively into) an oriented equation, we paramodulate from (respectively into) the left side only. With nonoriented equations, we paramodulate from and into both sides. We never paramodulate from or into variables.
- Paramodulants are demodulated, tested for max-weight and subsumption, and oriented if possible; then the variables are renumbered. Oriented equations are added to the set of demodulators and are used to back demodulate all existing equations (including demodulators). Back-demodulated equations are processed in the same way as paramodulants.
- For this set of experiments, we do not use the set of support restriction.

We believe our implementations of this starting inference system are complete for equational theorems when used with an agenda that explores the search space fairly (see 4.5.1, p. 8). In the literature, most of the restrictions and strategies that are the focus of this chapter have been proved complete within specific settings. However, we have not carefully analyzed for completeness the compatibility of the restrictions with our starting strategy or with each other. Our (extensive) use of the max-weight parameter obviously causes incompleteness.

4.4 EQP: The New Theorem Prover

We have been working on Otter [10] since 1988. It is a mature and stable program, with perhaps forty serious users. However, it has become difficult to work on because of its size and its many kludges, patches, and extraneous features. When a new capability is to be added, difficult questions arise such as "how does this affect

[5] We would like to write "basic" instead of "starting", but "basic" is used for another purpose in this chapter. The (unfortunately named) "basic paramodulation" is one of the strategies we evaluate in this chapter.

[6] This inference system has been in use at Argonne in various forms since about 1977. It is similar to G. E. Peterson's system [15] and unfailing Knuth-Bendix completion [3].

negative linked superhyperparamodulation, added in 1991, probably not used by anyone, but something I'd like to keep anyway?"

In 1992, we started an efficient implementation of AC unification, and that code has grown, very slowly, into the new theorem prover EQP. The most important two differences from Otter are that EQP includes AC unification and it applies to equational problems only. Also, EQP is much simpler and easier to extend. An early version of EQP was used for several experiments with algorithms for distributed equational reasoning (see, for example, [2]).

Similarities between Otter and EQP

1. Design philosophy and the C programming language.
2. The overall structure of the programs.
3. Unification and matching algorithms (excluding AC unification).
4. The term and literal indexing algorithms. Both programs (excluding EQP with AC unification) use discrimination tree indexing for retrieval of demodulators and subsuming clauses, and both use FPA-Path indexing for retrieval of unifiable terms during paramodulation and instance terms during back demodulation [9].
5. Input and output languages.
6. Noninteractive use of the programs.
7. Performance of the programs.

Differences between Otter and EQP

1. Otter and EQP share very little code.
2. Otter uses extensive structure sharing, which decreases memory consumption but complicates the algorithms and code. EQP uses simpler nonshared data structures for terms.
3. EQP has AC unification and matching.
4. Otter drives the search with the given clause algorithm; EQP can use either the given clause algorithm or the pair algorithm (see 4.5.1, p. 8).
5. EQP has many experimental paramodulation strategies (which are the focus of this chapter).
6. EQP applies to equational problems only; nonunit clauses are not accepted.

7. EQP lacks the following Otter features: user-defined weight functions, resolution inference rules, Skolemization and clausification, answer literals, autonomous mode, evaluable operations, the hot list, and back subsumption.

The lack of structure sharing was a major concern in the design of EQP. Consider a term t that must be indexed for back demodulation (i.e., we must be able to find it quickly when it can be rewritten by a newly derived demodulator). Say t has 1000 occurrences in our current set of clauses. Without structure sharing, each occurrence is indexed and retrieved separately; but with structure sharing, the one physical copy (to which all containing terms refer) is indexed and retrieved, and all containing clauses are then accessed by superterm lists. Indexing a term can require more memory than the term itself occupies, so lack of structure sharing causes the indexes to use even more memory than the terms use. However, nonshared terms are smaller, simpler, and faster to build, traverse, and recycle.

Table 4.1 compares Otter with EQP on the 33 test problems. The time to proof (seconds on an IBM RS/6000 processor[7]), proof length, and number of clauses generated are listed; "(M)" means that the memory limit of 24 megabytes was inadequate; "(T)" means that the time limit of 1800 seconds was inadequate; and "(S)" means that the program ran out of inferences to make, usually because the weight limit was too low for the strategy.

Both programs used the same simple paramodulation strategy, and they used similar search strategies (with a selection ratio of 4, see the following section), rewriting strategies, and clause-processing strategies. The (perhaps surprising) differences between the programs are due mostly to minor differences in the order in which sets of clauses are generated and retained. For example, say paramodulation of A into B generates ten clauses, and two, C and D, of equal length, are retained. Otter may select C as the next given clause, and EQP may select D—that is all it takes to cause the programs to search different areas of the space. The different generation orders are caused by differences in term data structures and term indexing. The counts of generated clauses are a rough measure of the amount of work done and thus can be used to compare the speed of the two programs; many of the problems indicate that EQP is 10%–20% faster than Otter. Both programs were allowed 24 megabytes for storage of clauses and indexes; problems LT-5, QLT-3, QLT-5, and QLT-6 indicate EQP's larger appetite for memory (which is not as large as we had feared).

[7] The RS/6000 processors we used run EQP about three times faster than a SPARC 2 does.

Table 4.1
Otter vs. EQP

	Otter			EQP		
Problem	time	(len.)	generated	time	(len.)	generated
CS-2	30	(7)	27368	25	(6)	18608
CS-6a	1	(7)	231	1	(7)	256
D-BA-1b	9	(28)	10822	(M)	(—)	149644
D-BA-2a	144	(124)	103327	140	(174)	130998
D-BA-5a	27	(84)	33943	30	(93)	39314
D-BA-5c	8	(89)	9473	6	(75)	8624
D-BA-8a	16	(106)	20731	30	(104)	40046
LT-10a	4	(15)	3566	3	(15)	2863
LT-2	(T)	(—)	2867810	924	(66)	1843691
LT-3e	43	(61)	56726	39	(56)	65398
LT-4	2	(13)	4011	2	(14)	3092
LT-5	(M)	(—)	261493	(M)	(—)	211985
LT-6	67	(97)	28390	46	(110)	30757
LT-8	45	(19)	75979	(T)	(—)	2530477
LT-9b	291	(53)	279260	287	(61)	338618
MFL-1	6	(36)	8574	4	(30)	5293
MFL-2	7	(29)	9666	1	(34)	2021
MFL-3	8	(26)	10789	7	(30)	9194
MFL-7	19	(39)	19283	15	(51)	14523
QLT-1	3	(47)	2999	3	(45)	3861
QLT-2	4	(19)	4372	4	(27)	5390
QLT-3	(M)	(—)	138808	(M)	(—)	106284
QLT-4	4	(54)	5022	5	(55)	7029
QLT-5	(M)	(—)	128442	(M)	(—)	84576
QLT-6	121	(108)	56904	(M)	(—)	92438
RBA-2	12	(37)	15910	19	(42)	27709
SD-2a	3	(33)	2326	2	(52)	2029
SD-3-e1	2	(17)	1941	1	(15)	821
SD-3-e2	8	(23)	10277	6	(27)	9484
TBA-1-e1	29	(42)	26822	126	(39)	78796
TBA-1-e2d	2	(30)	1328	2	(29)	1564
WAL-1a	149	(37)	233175	137	(31)	250483
WAL-2	8	(19)	14411	7	(16)	15005

4.5 Algorithms and Strategies

This section contains descriptions of the methods that are the focus of the main body of experiments.

4.5.1 The Given Clause and Pair Algorithms

The most important decision our theorem provers make when searching for proofs is selection of clauses for application of inference rules. We have considered two issues for these experiments: (1) the given clause algorithm vs. the pair algorithm, and (2) the ratio of best-first and breadth-first search.

Both the given clause algorithm and the pair algorithm can be thought of as simple loops that drive the search for a proof.

Repeat
1. decide on a set S of inferences to make;
2. make all inferences in S and process the results;

until a proof has been found.

The difference between the given clause and the pair algorithms is in the size of S. The given clause algorithm selects a clause C (the given clause) and makes inferences using C and all clauses previously selected as given clauses. The pair algorithm selects a pair of clauses (not previously selected) and makes inferences between those two clauses only.[8]

Our interest in the pair algorithm arose in 1992 when we started experimenting with AC unification and matching. Since a pair of terms can have a great number of most general AC unifiers, the set S in the given clause algorithm can be unmanageably large. Our intuition told us that the smaller granularity of the pair algorithm would offset somewhat the prolific nature of inference rules using AC unification.

For strictly breadth-first search, one might think that two algorithms should produce similar results. For example, if the sequence of retained clauses is $\langle 1, 2, 3 \rangle$, the pair algorithm considers the sequence of pairs $(1, 1), (2, 1), (2, 2), (3, 1), (3, 2), (3, 3)$, and the given clause algorithm considers the sequence $(1, \{1\}), (2, \{1, 2\}), (3, \{1, 2, 3\})$. However, the given clause algorithms in Otter and EQP order the set of previous given clauses in different ways, and neither orders them in increasing order. Such

[8] The pair algorithm has been used in many Knuth-Bendix completion systems. Its earliest use in automated deduction appears to have been by Larry Wos et al in 1964 with the unit preference strategy [21]. Perhaps the earliest use of the given clause algorithm was by Ross Overbeek in 1970 in the context of hyperresolution[14]. Most automated deduction at Argonne since 1970 has been with the given clause algorithm.

apparently small differences in the order of generated clauses can have a great effect on the character and outcome of the search.

The best-first aspects of the searches are based on the lengths of clauses—smaller is better. The length of a clause is the number of variables, constant, function, and predicate symbols. A best-first given clause algorithm selects the smallest clause not yet selected, and a best-first pair algorithm selects the smallest pair not yet selected, where the length of a pair is the sum of the lengths of its members.

Our experience with previous experiments tells us that a combination of best-first and breadth-first search, with an emphasis on best-first search, is usually a good choice. The ratio strategy, introduced in [12], allows the user to specify the ratio of the two selection methods. In our implementations of the ratio strategy, the user gives a parameter, the *pick-given-ratio*, say n, in the range $[0, 1, \cdots]$, meaning that through n iterations of the main loop, the best clause or pair is selected, then the oldest clause or pair, and so on.

Although the ratio strategy was not intended to be a focal point of the experiments, we have included a lot of data on it because it has a great effect on the search and it is not well understood; we hope to find relationships between the ratio strategy and the various paramodulation strategies. For most of the paramodulation strategies we considered, we ran experiments with ratios 1, 4, 8, and ∞ (i.e., purely best-first). (We used a value of 4 for almost all of the experiments reported in [11].)

4.5.2 Associative-Commutative (AC) Operations

EQP has AC unification (Stickel's AC algorithm [17], with Huet's algorithm [4] for finding basis solutions) and AC matching (our own algorithm). The AC unification algorithm is quite complicated, but our implementation is straightforward. AC terms are stored as binary terms in canonical form, and they are flattened into arrays during the unification process. Unifiers are generated incrementally on demand by a backtracking algorithm. We have not implemented Kapur and Zhang's check for symmetry among AC unifiers [7], and we do not use any indexing for finding AC-unifiable terms. Experience continues to show us that the speed of AC unification is not particularly important, because in practice the percentage of time spent there is small, but elimination of unnecessary or undesirable AC unifiers is extremely important.

EQP's AC matching code is separate from its AC unification code. Our implementation, which does not solve Diophantine equations, uses some ideas from RRL [6]. It is a backtracking algorithm that binds variables to all appropriate combinations of terms. The speed of AC matching is extremely important, because we

find that in practice, most of the time is spent rewriting derived clauses. When AC terms are present, we use a modification of discrimination tree indexing that filters AC terms by number of arguments and number of nonvariable arguments; this type of indexing gives us a modest improvement over indexing AC terms by simply treating them as constants.

AC paramodulation requires the use of extensions of equations. Consider $\alpha = \beta$ in which α has the AC symbol $+$ at its root. Then the equation $\alpha + x = \beta + x$, where x is a new variable, must be considered as well. AC paramodulation with extensions is extremely prolific; Kapur and Zhang have found that many such AC unifiers are unnecessary, and they advocate delaying the use of extensions. This is probably a wise strategy, but we have not yet implemented it. We currently paramodulate with extensions right after using the nonextended equations.

Term ordering is another problem area for terms with AC operations. For completion problems and for theory, it is useful to have a simplification ordering that is total for ground terms. This tells us how to orient equations into rewrite rules and guarantees termination of rewriting. Several such orderings have been defined for AC terms, but they are complex and we have not implemented any of them. Instead, we use the following very weak simplification ordering for AC terms: $t_1 \succ t_2$ if $length(t_1) > length(t_2)$ and $mvars(t_1) \gg= mvars(t_2)$. (The second condition means that no variable has more occurrences in t_2.) A rewrite rule satisfying our ordering always reduces the number of symbols. This ordering was originally implemented as a quick hack for testing AC demodulation, but it has been adequate for this set of experiments. EQP also has some special-purpose polynomial orderings, but they were not used here.

4.5.3 Paramodulation Strategies

Ordered-Instance Paramodulation. This strategy applies to non-AC experiments. (It applies in general to AC problems, but not in our implementations.) Ordered-instance paramodulation is a restriction that constrains the use of nonorientable equations. Ordinarily, equations that can be oriented have the larger side on the left and are added to the set of demodulators, and equations that cannot be oriented are not added to the set of demodulators, and they are stored in both orientations. Paramodulation is not allowed from the right side or into the right side of any equation. The preceding properties hold as well for ordered-instance paramodulation. As motivation for ordered-instance paramodulation, consider paramodulation from the nonorientable equation $\alpha = \beta$, with unifier σ such that $\alpha\sigma \prec \beta\sigma$; that is, the instance used for paramodulation is orientable *in the wrong direction*. The ordered-instance paramodulation restriction prevents paramodulation in

such cases. Nearly all current theoretical work on paramodulation strategies uses ordered-instance paramodulation, but Otter, for example does not. Early work in this area was in the context of completion [8].

Blocked Paramodulation. This strategy applies to non-AC and to AC experiments. Blocked paramodulation does not allow a paramodulation inference if any term in the unifier can be rewritten with the current set of demodulators. The strategy was discussed by Lankford and proved complete in the context of completion by Kapur, Musser, and Narendran [5]. The justification is that an unblocked inference is composite and thus can be factored into prime inferences, and that only prime inferences are needed for completion. See [5] for details. Our implementation of blocked paramodulation in EQP is straightforward—after unification or AC unification has generated a unifier and before generation of the corresponding paramodulant, each term in the substitution is tested to see if it can be rewritten with the current set of demodulators; if so, the paramodulant is not generated. Kapur and Zhang report that preventing unblocked inference gives great speedups on ring commutativity problems [5]. This is a fairly expensive test in our implementation; in the case of AC unification, we believe the test could be incorporated into the AC unification routines so that one test would apply to more than one unifier, but we have not attempted such a method.

Basic Paramodulation. This strategy applies to non-AC and to AC experiments. Basic paramodulation [1, 13] is a restriction that prevents paramodulation into terms that arise solely by instantiation. One way to view this (and a way to implement it) is to represent clauses as a pair, ⟨skeleton,substitution⟩. Input clauses are skeletons with empty substitutions, and paramodulation is done from skeletons into skeletons with the unifier of the inference composed with the substitutions of the parents to form the substitution of the paramodulant. The terms that arise by instantiation alone do not occur in the skeleton and are thus not available for paramodulation (are not *basic*). Another way to view this (and the way we implement it) is to mark terms that are not available for paramodulation. Input clauses have no terms marked. During paramodulation when the unifying substitution is applied to form the paramodulant, as variables are instantiated, the corresponding terms and all subterms are marked as unavailable for subsequent paramodulation; in addition, marks in the parents are inherited by the paramodulant.

Completeness of basic paramodulation requires special treatment for subsumption and demodulation. In the case of subsumption, consider clause C, with term t that is not basic (is unavailable for paramodulation); if C subsumes clause D and if

the term in D corresponding to t is basic, then t must be made basic. EQP without AC unification performs this procedure. EQP with AC unification, because of technical complications (corresponding terms may not be in corresponding positions), simply makes all of C basic; we believe the penalty for this extra allowance is small, because in practice, few clauses do most of the subsuming.

Demodulation with basic paramodulation is more problematic. The standard definitions of demodulation with basic paramodulation (e.g., [1]) do not allow demodulation of nonbasic terms. This restriction seems intuitively unwise, because we wish to simplify wherever possible; some preliminary experiments (not reported here) support our position, so EQP ignores that restriction. Demodulation is never prevented; and when a rewrite step occurs with $\alpha = \beta$, the instance of β and all of its subterms are made basic. This strategy can have the peculiar effect of creating nonbasic terms with basic subterms. We have not studied the completeness consequences of this method.

AC-superset-limit. This strategy applies to AC experiments. AC unification has two distinct effects when making terms identical: it permutes arguments and introduces new variables. We believe that permutation is the more useful effect and that introduction of variables is the more prolific effect. (The pair of terms $(x+x+x)$ and $(u+v+w+z)$ has 1,044,569 most general AC unifiers!) We have been thinking about heuristics for delaying or preventing the introduction of variables, and we have implemented one strategy along those lines. Very briefly stated, the AC unification algorithm works as follows to unify two terms with the same AC symbol at their roots. Arguments in common are removed, a linear Diophantine equation representing equality of the two terms is constructed, a basis of integer solutions is calculated such that all integer solutions are a linear combination of the basis solutions, then all of the subsets of the basis are considered, and those that satisfy certain constraints lead to potential AC unifiers. For the example above, the basis has 20 solutions, and most of the $2^{20} = 1048576$ subsets lead to unifiers. The EQP parameter *ac-superset-limit* limits the number of combinations that are considered. A value of 0 means that if a subset S leads to a potential unifier, then no supersets of S are considered; a value of 1 means to consider supersets of size $\leq |S| + 1$, and so on.[9] The strategy is obviously incomplete, because it eliminates most-general AC unifiers.

[9] A method similar to *ac-superset-limit=0* has been used in RRL [6].

Functional Subsumption. This strategy applies to non-AC and to AC experiments. Functional subsumption is a simple extension of ordinary subsumption. It takes into account the substitution property, $x = y \Rightarrow f(\cdots x \cdots) = f(\cdots y \cdots)$, of equality. For example, $\alpha = \beta$ functionally subsumes $g(h(t, \alpha)) = g(h(t, \beta))$. A recursive definition for equations is the following.

$\alpha = \beta$ functionally-subsumes $\gamma = \delta$ iff
  ($\alpha = \beta$ subsumes $\gamma = \delta$) or
  ($\gamma = \delta$ is $f(\cdots \zeta \cdots) = f(\cdots \eta \cdots)$ and $\alpha = \beta$ functionally-subsumes $\zeta = \eta$).

The notation $f(\cdots \zeta \cdots) = f(\cdots \eta \cdots)$ means that ζ and η are in the same argument position and everything else in the two terms is identical. The informal justification for the rule is similar to that for ordinary subsumption: anything useful that can be done with $f(\cdots \alpha \cdots) = f(\cdots \beta \cdots)$ can be done as well with $\alpha = \beta$. Note that if $\alpha = \beta$ is a demodulator, then it need not be used to check functional subsumption, because demodulation occurs before the subsumption tests. We believe the deletion of functionally subsumed clauses is complete for many equational inference systems.

4.6 Experiments

This section contains the results of the main body of experiments. Recall that there are 33 problems of which 13 have at least one AC function symbol. The first part is without AC unification, with data on all 33 problems (the 13 AC problems include AC axioms). The second part is with AC unification, with data on the 13 AC problems.

Each paramodulation strategy was run with four different *best:breadth* ratios for selecting the next unit of work during the search: 1 (1:1), 4 (4:1), 8 (8:1), and n (1:0). These are specified in the names of the strategies (e.g., "block-8-giv") in the tables below. Each paramodulation strategy was also run with the given algorithm and with the pair algorithm, which is specified with "-giv" or "-pair" in the tables.

The time (seconds on an IBM RS/6000 processor) to proof and proof length for each search are listed; "(M)" means that the memory limit of 24 megabytes was inadequate, "(T)" means that the time limit of 1800 seconds was inadequate, and "(S)" means that the program ran out of inferences to make, usually because the weight limit was too low for the strategy. The smallest time(s) for each problem is set in a box.

The proof lengths are listed because there has been recent interest in strategies for finding short proofs [20, 19]. The length includes paramodulation inferences and steps to flip equations, but it does not include demodulation steps. Thus,

comparing lengths on the same theorem can be misleading because paramodulation steps in one proof can correspond to demodulation proofs in another. Nevertheless, it is usually a good measure of the complexity of proofs, and it can shed light on strategies that lead to simpler proofs.

Experiments were run to evaluate functional subsumption, but the results were not interesting (see 4.7) and have been omitted.

Non-AC Experiments. We list results for five paramodulation strategies: the simple starting strategy (start), ordered-instance paramodulation (ord), blocked paramodulation (block), basic paramodulation (basic), and a combination of ordered-instance, blocked, basic paramodulation with functional subsumption (all). With each paramodulation strategy we list results for the four values of the ratio parameter and given vs. pairs; this gives 40 runs for each of the 33 non-AC problems. As an example of a strategy name, "block-8-giv" means blocked paramodulation, ratio 8, with the given clause algorithm. Tables 4.2 through 4.8 contain the results of the non-AC experiments.

AC Experiments. These are similar to the non-AC experiments except that ordered-instance paramodulation is omitted and AC superset limit 0 (super0) is included. Each of the 13 AC problems was run in 40 ways. Tables 4.9 through 4.11, starting on p. 22, contain the results of the AC experiments.

Table 4.2
Non-AC: Group A

	D-BA-1b		D-BA-2a		D-BA-5a		D-BA-5c		D-BA-8a	
Strategy	time	(len.)	time	(len.)	time	(len.)	time	(len.)	time	(len.)
start-1-giv	(M)	(—)	415	(158)	33	(52)	2	(57)	93	(207)
start-4-giv	(M)	(—)	140	(174)	30	(93)	6	(75)	30	(104)
start-8-giv	3	(27)	185	(206)	31	(116)	4	(17)	20	(136)
start-n-giv	(M)	(—)	100	(248)	35	(117)	4	(24)	(M)	(—)
start-1-pair	(M)	(—)	237	(175)	24	(39)	4	(42)	122	(81)
start-4-pair	(M)	(—)	113	(140)	15	(16)	2	(45)	44	(72)
start-8-pair	25	(49)	149	(214)	25	(82)	2	(18)	27	(92)
start-n-pair	104	(63)	196	(210)	32	(94)	3	(17)	(M)	(—)
block-1-giv	(M)	(—)	471	(158)	37	(127)	2	(57)	100	(158)
block-4-giv	(M)	(—)	163	(174)	35	(82)	6	(72)	33	(104)
block-8-giv	3	(27)	232	(214)	35	(126)	4	(17)	21	(136)
block-n-giv	(M)	(—)	115	(271)	38	(112)	4	(24)	(M)	(—)
block-1-pair	(M)	(—)	305	(242)	26	(39)	4	(42)	122	(81)
block-4-pair	(M)	(—)	132	(141)	17	(16)	2	(45)	47	(72)
block-8-pair	26	(49)	237	(285)	29	(84)	2	(18)	31	(91)
block-n-pair	123	(63)	311	(231)	24	(104)	3	(17)	(M)	(—)
basic-1-giv	(M)	(—)	118	(228)	12	(83)	7	(24)	25	(80)
basic-4-giv	3	(28)	59	(241)	9	(18)	3	(67)	17	(140)
basic-8-giv	1	(27)	43	(235)	9	(55)	2	(22)	13	(126)
basic-n-giv	(M)	(—)	38	(195)	11	(84)	2	(23)	(M)	(—)
basic-1-pair	(M)	(—)	71	(237)	12	(65)	3	(12)	5	(62)
basic-4-pair	15	(33)	58	(234)	10	(90)	1	(12)	26	(69)
basic-8-pair	10	(38)	52	(339)	11	(118)	1	(18)	16	(71)
basic-n-pair	33	(78)	155	(265)	22	(108)	1	(17)	(M)	(—)
ord-1-giv	(M)	(—)	(M)	(—)	109	(60)	14	(35)	129	(71)
ord-4-giv	(M)	(—)	142	(164)	37	(59)	6	(75)	46	(80)
ord-8-giv	2	(42)	214	(248)	31	(110)	4	(16)	43	(81)
ord-n-giv	(M)	(—)	115	(275)	36	(116)	4	(24)	(M)	(—)
ord-1-pair	(M)	(—)	236	(175)	24	(39)	4	(42)	122	(81)
ord-4-pair	(M)	(—)	113	(140)	15	(16)	2	(45)	44	(72)
ord-8-pair	25	(49)	149	(214)	26	(82)	2	(18)	27	(92)
ord-n-pair	101	(63)	194	(210)	33	(94)	3	(17)	(M)	(—)
all-1-giv	27	(60)	218	(276)	37	(118)	6	(24)	32	(106)
all-4-giv	2	(26)	92	(320)	14	(76)	3	(65)	18	(129)
all-8-giv	5	(46)	63	(225)	12	(127)	2	(22)	26	(75)
all-n-giv	8	(151)	99	(258)	10	(63)	2	(23)	(M)	(—)
all-1-pair	(M)	(—)	188	(259)	12	(74)	3	(12)	5	(62)
all-4-pair	14	(33)	135	(253)	9	(96)	1	(12)	24	(59)
all-8-pair	10	(42)	64	(287)	11	(110)	1	(18)	17	(66)
all-n-pair	21	(74)	86	(182)	11	(97)	2	(17)	(M)	(—)

Table 4.3
Non-AC: Group B

Strategy	LT-2 time	LT-2 (len.)	LT-3e time	LT-3e (len.)	LT-4 time	LT-4 (len.)	LT-5 time	LT-5 (len.)	LT-6 time	LT-6 (len.)
start-1-giv	(T)	(—)	51	(87)	1	(8)	46	(86)	109	(219)
start-4-giv	924	(66)	39	(56)	2	(14)	(M)	(—)	46	(110)
start-8-giv	(T)	(—)	30	(65)	4	(27)	(M)	(—)	30	(114)
start-n-giv	(T)	(—)	27	(62)	7	(31)	(M)	(—)	(M)	(—)
start-1-pair	(T)	(—)	48	(59)	2	(8)	53	(100)	40	(163)
start-4-pair	(T)	(—)	69	(67)	1	(16)	45	(83)	9	(82)
start-8-pair	(T)	(—)	136	(54)	2	(22)	42	(79)	6	(88)
start-n-pair	(T)	(—)	72	(56)	2	(15)	(M)	(—)	(M)	(—)
block-1-giv	(T)	(—)	65	(87)	1	(8)	52	(85)	114	(250)
block-4-giv	1223	(66)	47	(55)	2	(14)	(M)	(—)	50	(95)
block-8-giv	(T)	(—)	37	(65)	5	(27)	(M)	(—)	32	(114)
block-n-giv	(T)	(—)	34	(62)	8	(28)	(M)	(—)	(M)	(—)
block-1-pair	(T)	(—)	55	(59)	2	(8)	58	(100)	40	(158)
block-4-pair	(T)	(—)	82	(67)	2	(16)	51	(83)	9	(83)
block-8-pair	(T)	(—)	159	(54)	2	(22)	46	(79)	6	(88)
block-n-pair	(T)	(—)	88	(56)	2	(15)	(M)	(—)	(M)	(—)
basic-1-giv	189	(25)	(T)	(—)	0	(8)	26	(85)	73	(131)
basic-4-giv	855	(41)	20	(73)	1	(13)	6	(57)	43	(66)
basic-8-giv	(T)	(—)	14	(101)	2	(20)	(M)	(—)	30	(107)
basic-n-giv	(T)	(—)	25	(58)	3	(17)	(M)	(—)	(M)	(—)
basic-1-pair	(T)	(—)	18	(83)	0	(8)	15	(81)	34	(262)
basic-4-pair	(T)	(—)	22	(77)	1	(17)	(M)	(—)	44	(135)
basic-8-pair	(T)	(—)	46	(70)	1	(17)	24	(70)	18	(126)
basic-n-pair	(T)	(—)	108	(58)	1	(14)	(M)	(—)	(M)	(—)
ord-1-giv	(T)	(—)	70	(82)	1	(8)	44	(77)	108	(282)
ord-4-giv	(T)	(—)	43	(65)	2	(14)	(M)	(—)	48	(109)
ord-8-giv	(T)	(—)	34	(68)	5	(27)	(M)	(—)	30	(107)
ord-n-giv	(T)	(—)	38	(96)	7	(18)	(M)	(—)	(M)	(—)
ord-1-pair	(T)	(—)	48	(59)	2	(8)	53	(100)	40	(163)
ord-4-pair	(T)	(—)	68	(67)	1	(16)	44	(83)	9	(82)
ord-8-pair	(T)	(—)	134	(54)	2	(22)	41	(79)	6	(88)
ord-n-pair	(T)	(—)	71	(56)	2	(15)	(M)	(—)	(M)	(—)
all-1-giv	(T)	(—)	229	(134)	1	(8)	36	(67)	72	(244)
all-4-giv	(T)	(—)	23	(112)	1	(13)	6	(68)	39	(73)
all-8-giv	(T)	(—)	10	(83)	3	(18)	(M)	(—)	30	(102)
all-n-giv	(T)	(—)	44	(79)	4	(8)	(M)	(—)	(M)	(—)
all-1-pair	(T)	(—)	38	(68)	1	(8)	15	(74)	40	(197)
all-4-pair	(T)	(—)	34	(56)	1	(22)	27	(99)	49	(90)
all-8-pair	(T)	(—)	43	(75)	1	(22)	24	(91)	41	(237)
all-n-pair	(T)	(—)	106	(58)	1	(14)	(M)	(—)	(M)	(—)

Table 4.4
Non-AC: Group C

	LT-8		LT-9b		LT-10a		CS-2		CS-6a	
Strategy	time	(len.)	time	(len.)	time	(len.)	time	(len.)	time	(len.)
start-1-giv	(T)	(—)	203	(46)	32	(16)	19	(6)	1	(5)
start-4-giv	(T)	(—)	287	(61)	3	(15)	25	(6)	1	(7)
start-8-giv	(T)	(—)	296	(51)	1	(14)	22	(6)	1	(8)
start-n-giv	(T)	(—)	(M)	(—)	1	(14)	22	(6)	4	(7)
start-1-pair	31	(18)	68	(40)	4	(18)	12	(7)	0	(3)
start-4-pair	16	(21)	245	(37)	1	(15)	34	(7)	0	(6)
start-8-pair	14	(21)	393	(45)	1	(13)	33	(8)	0	(6)
start-n-pair	11	(21)	270	(40)	0	(16)	34	(10)	6	(7)
block-1-giv	(T)	(—)	243	(46)	35	(16)	20	(7)	1	(5)
block-4-giv	(T)	(—)	353	(61)	3	(15)	27	(7)	1	(7)
block-8-giv	(T)	(—)	361	(51)	2	(14)	23	(7)	1	(8)
block-n-giv	(T)	(—)	(M)	(—)	1	(14)	23	(7)	4	(7)
block-1-pair	37	(18)	82	(40)	4	(18)	13	(7)	0	(3)
block-4-pair	20	(21)	291	(37)	1	(15)	36	(7)	1	(6)
block-8-pair	17	(21)	479	(45)	1	(13)	35	(8)	1	(6)
block-n-pair	14	(21)	339	(40)	0	(16)	36	(10)	7	(7)
basic-1-giv	(T)	(—)	71	(68)	3	(26)	17	(6)	1	(5)
basic-4-giv	(T)	(—)	28	(70)	1	(36)	21	(6)	1	(7)
basic-8-giv	(T)	(—)	37	(80)	1	(49)	18	(6)	1	(8)
basic-n-giv	(T)	(—)	111	(83)	1	(40)	18	(6)	4	(7)
basic-1-pair	(T)	(—)	19	(43)	1	(36)	10	(7)	0	(3)
basic-4-pair	(T)	(—)	256	(50)	1	(41)	27	(7)	1	(6)
basic-8-pair	(T)	(—)	148	(47)	0	(41)	27	(8)	1	(6)
basic-n-pair	17	(15)	98	(50)	0	(40)	29	(10)	6	(7)
ord-1-giv	(T)	(—)	192	(48)	30	(16)	20	(6)	1	(8)
ord-4-giv	(T)	(—)	295	(64)	3	(15)	27	(6)	0	(3)
ord-8-giv	(T)	(—)	302	(85)	1	(14)	23	(6)	1	(3)
ord-n-giv	(T)	(—)	(M)	(—)	1	(14)	23	(6)	4	(7)
ord-1-pair	31	(18)	67	(40)	4	(18)	12	(7)	0	(3)
ord-4-pair	16	(21)	243	(37)	1	(15)	34	(7)	1	(6)
ord-8-pair	14	(21)	388	(45)	1	(13)	33	(8)	1	(6)
ord-n-pair	11	(21)	269	(40)	0	(16)	34	(10)	6	(7)
all-1-giv	(T)	(—)	89	(64)	4	(26)	17	(7)	1	(8)
all-4-giv	(T)	(—)	38	(76)	1	(37)	14	(7)	0	(3)
all-8-giv	(T)	(—)	49	(87)	1	(41)	11	(7)	1	(3)
all-n-giv	(T)	(—)	135	(134)	1	(41)	11	(7)	4	(7)
all-1-pair	(T)	(—)	23	(43)	1	(36)	8	(7)	0	(3)
all-4-pair	(T)	(—)	224	(50)	1	(41)	24	(7)	1	(6)
all-8-pair	(T)	(—)	172	(47)	1	(41)	46	(7)	1	(6)
all-n-pair	17	(15)	111	(50)	0	(40)	55	(11)	7	(7)

Table 4.5
Non-AC: Group D

Strategy	QLT-1 time	(len.)	QLT-2 time	(len.)	QLT-3 time	(len.)	QLT-4 time	(len.)	QLT-5 time	(len.)
start-1-giv	4	(23)	6	(25)	(M)	(—)	10	(66)	(M)	(—)
start-4-giv	3	(45)	4	(27)	(M)	(—)	5	(55)	(M)	(—)
start-8-giv	4	(42)	3	(27)	(M)	(—)	7	(53)	(M)	(—)
start-n-giv	5	(55)	17	(64)	(M)	(—)	45	(46)	(M)	(—)
start-1-pair	6	(39)	5	(23)	(M)	(—)	2	(23)	168	(122)
start-4-pair	3	(36)	3	(17)	(M)	(—)	4	(23)	100	(52)
start-8-pair	3	(50)	7	(39)	(M)	(—)	3	(21)	97	(62)
start-n-pair	13	(44)	13	(49)	(M)	(—)	6	(30)	(M)	(—)
block-1-giv	5	(23)	6	(25)	(M)	(—)	11	(66)	(M)	(—)
block-4-giv	3	(45)	5	(27)	(M)	(—)	6	(55)	(M)	(—)
block-8-giv	4	(42)	4	(27)	(M)	(—)	7	(53)	(M)	(—)
block-n-giv	6	(55)	18	(64)	(M)	(—)	51	(46)	(M)	(—)
block-1-pair	6	(39)	5	(23)	(M)	(—)	2	(23)	178	(122)
block-4-pair	3	(36)	4	(17)	(M)	(—)	5	(23)	105	(52)
block-8-pair	3	(50)	7	(39)	(M)	(—)	4	(21)	104	(62)
block-n-pair	15	(44)	15	(49)	(M)	(—)	6	(30)	(M)	(—)
basic-1-giv	2	(24)	1	(29)	(M)	(—)	10	(56)	149	(52)
basic-4-giv	2	(45)	2	(28)	(M)	(—)	6	(52)	(M)	(—)
basic-8-giv	3	(45)	2	(29)	(M)	(—)	5	(51)	(M)	(—)
basic-n-giv	3	(51)	12	(53)	(M)	(—)	30	(46)	(M)	(—)
basic-1-pair	4	(36)	3	(18)	(M)	(—)	2	(22)	150	(142)
basic-4-pair	3	(39)	3	(17)	(M)	(—)	3	(24)	91	(110)
basic-8-pair	3	(52)	6	(16)	(M)	(—)	2	(21)	89	(104)
basic-n-pair	9	(20)	19	(114)	(M)	(—)	4	(30)	(M)	(—)
ord-1-giv	4	(23)	7	(23)	(M)	(—)	13	(92)	(M)	(—)
ord-4-giv	3	(34)	5	(28)	(M)	(—)	6	(79)	(M)	(—)
ord-8-giv	4	(32)	4	(18)	(M)	(—)	7	(115)	(M)	(—)
ord-n-giv	5	(38)	19	(50)	(M)	(—)	48	(96)	(M)	(—)
ord-1-pair	6	(39)	5	(23)	(M)	(—)	2	(23)	169	(122)
ord-4-pair	3	(36)	3	(17)	(M)	(—)	4	(23)	100	(52)
ord-8-pair	3	(50)	6	(39)	(M)	(—)	3	(21)	96	(62)
ord-n-pair	13	(44)	13	(49)	(M)	(—)	6	(30)	(M)	(—)
all-1-giv	2	(30)	2	(36)	(M)	(—)	8	(69)	(M)	(—)
all-4-giv	3	(60)	2	(32)	(M)	(—)	6	(64)	(M)	(—)
all-8-giv	3	(66)	2	(28)	(M)	(—)	5	(80)	(M)	(—)
all-n-giv	10	(51)	6	(49)	(M)	(—)	15	(81)	(M)	(—)
all-1-pair	3	(33)	3	(18)	(M)	(—)	1	(22)	71	(44)
all-4-pair	2	(33)	3	(17)	(M)	(—)	2	(24)	119	(71)
all-8-pair	3	(53)	5	(16)	(M)	(—)	2	(20)	58	(109)
all-n-pair	9	(20)	14	(127)	(M)	(—)	4	(39)	317	(101)

Table 4.6
Non-AC: Group E

	QLT-6		MFL-1		MFL-2		MFL-3		MFL-7	
Strategy	time	(len.)	time	(len.)	time	(len.)	time	(len.)	time	(len.)
start-1-giv	(M)	(—)	2	(23)	3	(29)	8	(28)	24	(57)
start-4-giv	(M)	(—)	4	(30)	[1]	(34)	7	(30)	15	(51)
start-8-giv	(M)	(—)	10	(32)	4	(34)	8	(40)	13	(55)
start-n-giv	(M)	(—)	9	(46)	10	(77)	8	(60)	9	(51)
start-1-pair	146	(41)	5	(38)	4	(28)	4	(20)	8	(40)
start-4-pair	116	(46)	3	(52)	2	(32)	3	(30)	5	(44)
start-8-pair	100	(57)	3	(37)	3	(30)	2	(30)	5	(42)
start-n-pair	133	(368)	3	(43)	3	(33)	3	(81)	4	(57)
block-1-giv	(M)	(—)	2	(23)	14	(33)	8	(28)	18	(58)
block-4-giv	186	(113)	9	(28)	7	(34)	8	(31)	14	(51)
block-8-giv	(M)	(—)	13	(31)	12	(35)	8	(38)	13	(55)
block-n-giv	(M)	(—)	12	(37)	11	(77)	9	(60)	10	(51)
block-1-pair	154	(41)	5	(38)	5	(28)	4	(20)	8	(40)
block-4-pair	120	(46)	3	(49)	3	(32)	3	(33)	5	(47)
block-8-pair	103	(57)	3	(35)	3	(30)	2	(30)	5	(44)
block-n-pair	133	(328)	3	(46)	3	(33)	4	(83)	4	(59)
basic-1-giv	152	(72)	[1]	(38)	8	(37)	4	(27)	15	(58)
basic-4-giv	(M)	(—)	7	(42)	5	(31)	3	(37)	8	(58)
basic-8-giv	(M)	(—)	6	(33)	5	(52)	11	(55)	22	(75)
basic-n-giv	(M)	(—)	7	(38)	8	(109)	6	(78)	20	(72)
basic-1-pair	179	(55)	4	(34)	3	(31)	2	(27)	7	(53)
basic-4-pair	131	(52)	3	(30)	2	(31)	2	(28)	5	(42)
basic-8-pair	120	(56)	2	(43)	2	(39)	[1]	(33)	4	(42)
basic-n-pair	168	(263)	2	(72)	2	(67)	[1]	(69)	[3]	(42)
ord-1-giv	(M)	(—)	2	(23)	3	(29)	8	(28)	25	(57)
ord-4-giv	(M)	(—)	4	(30)	[1]	(34)	7	(30)	15	(51)
ord-8-giv	(M)	(—)	10	(32)	4	(34)	8	(40)	13	(55)
ord-n-giv	(M)	(—)	8	(42)	11	(78)	8	(60)	9	(51)
ord-1-pair	146	(41)	4	(38)	4	(28)	4	(20)	8	(40)
ord-4-pair	116	(46)	3	(52)	3	(32)	3	(30)	5	(44)
ord-8-pair	100	(57)	2	(37)	2	(30)	2	(30)	5	(42)
ord-n-pair	132	(368)	3	(43)	3	(33)	4	(81)	4	(57)
all-1-giv	138	(113)	6	(34)	7	(37)	5	(26)	27	(47)
all-4-giv	(M)	(—)	10	(51)	5	(33)	9	(46)	11	(50)
all-8-giv	(M)	(—)	6	(33)	7	(34)	4	(49)	25	(102)
all-n-giv	(M)	(—)	9	(33)	7	(100)	9	(109)	25	(77)
all-1-pair	145	(55)	4	(34)	3	(31)	2	(30)	6	(53)
all-4-pair	93	(190)	3	(31)	2	(31)	2	(31)	6	(45)
all-8-pair	[88]	(75)	2	(44)	2	(39)	2	(33)	5	(45)
all-n-pair	89	(58)	2	(72)	2	(68)	2	(69)	[3]	(45)

Table 4.7
Non-AC: Group F

Strategy	RBA-2 time	RBA-2 (len.)	TBA-1-e1 time	TBA-1-e1 (len.)	TBA-1-e2d time	TBA-1-e2d (len.)	WAL-1a time	WAL-1a (len.)	WAL-2 time	WAL-2 (len.)
start-1-giv	53	(53)	168	(50)	5	(42)	140	(33)	7	(17)
start-4-giv	19	(42)	126	(39)	2	(29)	137	(31)	7	(16)
start-8-giv	13	(44)	177	(37)	2	(32)	187	(29)	7	(17)
start-n-giv	104	(74)	(M)	(—)	4	(42)	1343	(60)	6	(17)
start-1-pair	43	(42)	103	(42)	1	(31)	330	(35)	(S)	(—)
start-4-pair	19	(41)	32	(35)	1	(28)	233	(33)	(S)	(—)
start-8-pair	15	(45)	14	(35)	1	(37)	281	(33)	(S)	(—)
start-n-pair	10	(38)	(M)	(—)	2	(38)	227	(40)	(S)	(—)
block-1-giv	62	(53)	187	(43)	5	(33)	178	(33)	9	(17)
block-4-giv	23	(42)	133	(39)	4	(46)	183	(31)	9	(16)
block-8-giv	16	(44)	168	(37)	4	(43)	251	(29)	9	(17)
block-n-giv	122	(74)	(M)	(—)	4	(42)	(T)	(—)	8	(17)
block-1-pair	49	(42)	110	(42)	1	(42)	409	(35)	(S)	(—)
block-4-pair	22	(41)	35	(35)	1	(44)	300	(33)	(S)	(—)
block-8-pair	16	(45)	16	(35)	1	(47)	367	(33)	(S)	(—)
block-n-pair	12	(38)	(M)	(—)	5	(43)	299	(40)	(S)	(—)
basic-1-giv	23	(49)	(M)	(—)	37	(81)	69	(62)	(S)	(—)
basic-4-giv	31	(48)	(M)	(—)	17	(74)	33	(62)	1	(30)
basic-8-giv	26	(43)	127	(39)	2	(38)	52	(70)	1	(36)
basic-n-giv	247	(69)	(M)	(—)	2	(39)	38	(81)	1	(30)
basic-1-pair	(M)	(—)	54	(40)	1	(33)	68	(88)	2	(26)
basic-4-pair	609	(140)	21	(32)	1	(32)	71	(68)	(S)	(—)
basic-8-pair	523	(142)	17	(24)	1	(31)	182	(77)	(S)	(—)
basic-n-pair	350	(143)	(M)	(—)	1	(62)	152	(64)	(S)	(—)
ord-1-giv	46	(61)	193	(52)	5	(42)	135	(43)	7	(17)
ord-4-giv	20	(45)	127	(39)	3	(32)	147	(31)	7	(16)
ord-8-giv	29	(42)	177	(37)	2	(35)	224	(36)	7	(17)
ord-n-giv	107	(59)	(M)	(—)	4	(42)	1370	(54)	6	(17)
ord-1-pair	43	(42)	104	(42)	1	(31)	325	(35)	(S)	(—)
ord-4-pair	19	(41)	31	(35)	1	(28)	233	(33)	(S)	(—)
ord-8-pair	14	(45)	14	(35)	1	(37)	278	(33)	(S)	(—)
ord-n-pair	10	(38)	(M)	(—)	2	(38)	226	(40)	(S)	(—)
all-1-giv	14	(35)	(M)	(—)	6	(34)	104	(58)	(S)	(—)
all-4-giv	23	(32)	(M)	(—)	3	(36)	44	(60)	(S)	(—)
all-8-giv	34	(32)	149	(45)	2	(36)	67	(65)	2	(36)
all-n-giv	122	(58)	(M)	(—)	2	(54)	50	(85)	2	(30)
all-1-pair	641	(142)	55	(40)	1	(28)	215	(89)	2	(26)
all-4-pair	491	(148)	24	(32)	1	(27)	190	(75)	(S)	(—)
all-8-pair	496	(124)	198	(45)	1	(28)	209	(78)	(S)	(—)
all-n-pair	301	(128)	(M)	(—)	3	(63)	184	(64)	(S)	(—)

Table 4.8
Non-AC: Group G

Strategy	SD-2a time	(len.)	SD-3-e1 time	(len.)	SD-3-e2 time	(len.)
start-1-giv	[2]	(34)	4	(32)	[2]	(23)
start-4-giv	[2]	(52)	1	(15)	6	(27)
start-8-giv	[2]	(52)	1	(15)	30	(29)
start-n-giv	6	(63)	1	(15)	50	(41)
start-1-pair	5	(33)	52	(89)	4	(19)
start-4-pair	4	(63)	5	(51)	10	(24)
start-8-pair	4	(44)	2	(43)	13	(37)
start-n-pair	23	(53)	1	(17)	13	(37)
block-1-giv	[2]	(34)	5	(32)	3	(23)
block-4-giv	[2]	(52)	1	(15)	8	(27)
block-8-giv	[2]	(52)	1	(15)	36	(29)
block-n-giv	7	(63)	1	(15)	58	(41)
block-1-pair	6	(33)	53	(89)	4	(19)
block-4-pair	5	(63)	5	(51)	12	(24)
block-8-pair	5	(44)	3	(43)	15	(37)
block-n-pair	26	(53)	2	(17)	16	(37)
basic-1-giv	5	(55)	12	(38)	15	(27)
basic-4-giv	3	(73)	2	(23)	3	(26)
basic-8-giv	[2]	(39)	1	(30)	5	(29)
basic-n-giv	6	(68)	1	(30)	42	(32)
basic-1-pair	8	(80)	12	(60)	[2]	(21)
basic-4-pair	6	(83)	6	(74)	5	(24)
basic-8-pair	6	(83)	3	(61)	8	(31)
basic-n-pair	15	(54)	2	(35)	9	(40)
ord-1-giv	11	(74)	7	(34)	45	(39)
ord-4-giv	4	(76)	1	(15)	30	(32)
ord-8-giv	3	(63)	1	(15)	33	(29)
ord-n-giv	6	(68)	[0]	(15)	54	(40)
ord-1-pair	5	(33)	52	(89)	4	(19)
ord-4-pair	4	(63)	5	(51)	10	(24)
ord-8-pair	4	(44)	2	(43)	13	(37)
ord-n-pair	23	(53)	1	(17)	13	(37)
all-1-giv	26	(93)	11	(33)	14	(32)
all-4-giv	12	(81)	2	(25)	4	(26)
all-8-giv	3	(65)	1	(31)	6	(29)
all-n-giv	14	(51)	4	(23)	47	(32)
all-1-pair	5	(70)	16	(43)	[2]	(21)
all-4-pair	3	(48)	2	(49)	6	(24)
all-8-pair	4	(91)	2	(33)	9	(31)
all-n-pair	17	(46)	1	(33)	11	(40)

Table 4.9
AC: Group A

Strategy	LT-2 time	(len.)	LT-5 time	(len.)	LT-6 time	(len.)	LT-8 time	(len.)
start-1-giv	202	(10)	13	(10)	615	(5)	816	(6)
start-4-giv	165	(9)	90	(10)	(T)	(—)	164	(6)
start-8-giv	384	(9)	834	(20)	(T)	(—)	51	(6)
start-n-giv	960	(9)	(T)	(—)	(T)	(—)	21	(6)
start-1-pair	803	(10)	6	(10)	112	(5)	602	(5)
start-4-pair	748	(8)	18	(15)	438	(14)	137	(5)
start-8-pair	896	(8)	31	(19)	788	(14)	114	(5)
start-n-pair	1463	(7)	(T)	(—)	(T)	(—)	20	(5)
block-1-giv	308	(10)	10	(10)	529	(5)	1392	(6)
block-4-giv	255	(9)	181	(10)	(T)	(—)	282	(6)
block-8-giv	549	(9)	(T)	(—)	(T)	(—)	85	(6)
block-n-giv	1314	(9)	(T)	(—)	(T)	(—)	25	(6)
block-1-pair	1218	(10)	6	(10)	85	(5)	1056	(5)
block-4-pair	1033	(8)	10	(15)	225	(14)	239	(5)
block-8-pair	1197	(8)	28	(19)	582	(14)	223	(5)
block-n-pair	1659	(7)	(T)	(—)	(T)	(—)	22	(5)
basic-1-giv	172	(10)	—	(—)	598	(5)	416	(6)
basic-4-giv	136	(9)	192	(11)	1821	(19)	91	(6)
basic-8-giv	312	(9)	157	(13)	(T)	(—)	33	(6)
basic-n-giv	696	(9)	(T)	(—)	(T)	(—)	10	(6)
basic-1-pair	662	(10)	(T)	(—)	112	(5)	477	(6)
basic-4-pair	606	(8)	36	(30)	295	(14)	141	(6)
basic-8-pair	683	(8)	25	(18)	510	(14)	118	(6)
basic-n-pair	1086	(7)	(T)	(—)	(T)	(—)	39	(6)
super0-1-giv	45	(10)	16	(15)	32	(12)	35	(6)
super0-4-giv	22	(9)	26	(21)	164	(20)	8	(6)
super0-8-giv	61	(9)	75	(21)	1672	(24)	6	(6)
super0-n-giv	354	(9)	(T)	(—)	(T)	(—)	4	(6)
super0-1-pair	278	(10)	4	(15)	11	(9)	26	(5)
super0-4-pair	280	(8)	3	(18)	25	(15)	7	(5)
super0-8-pair	311	(8)	7	(16)	48	(14)	6	(5)
super0-n-pair	709	(7)	1421	(43)	(T)	(—)	4	(5)
all-1-giv	44	(10)	23	(14)	29	(12)	12	(6)
all-4-giv	19	(9)	3	(5)	147	(20)	4	(6)
all-8-giv	54	(9)	18	(8)	719	(13)	2	(6)
all-n-giv	299	(9)	(T)	(—)	(T)	(—)	2	(6)
all-1-pair	290	(10)	4	(15)	11	(9)	27	(6)
all-4-pair	269	(8)	3	(18)	21	(15)	5	(6)
all-8-pair	269	(8)	6	(15)	37	(14)	4	(6)
all-n-pair	607	(7)	737	(66)	1482	(29)	2	(6)

Table 4.10
AC: Group B

	QLT-1		QLT-2		QLT-3		QLT-4	
Strategy	time	(len.)	time	(len.)	time	(len.)	time	(len.)
start-1-giv	3	(11)	5	(12)	1296	(68)	17	(6)
start-4-giv	134	(11)	62	(13)	165	(51)	19	(9)
start-8-giv	134	(11)	63	(13)	193	(62)	19	(9)
start-n-giv	55	(11)	57	(12)	(T)	(—)	19	(9)
start-1-pair	6	(11)	19	(11)	854	(52)	3	(6)
start-4-pair	18	(11)	35	(13)	(T)	(—)	4	(6)
start-8-pair	23	(11)	56	(18)	(T)	(—)	10	(6)
start-n-pair	207	(23)	54	(18)	(T)	(—)	77	(20)
block-1-giv	2	(11)	4	(12)	392	(83)	7	(6)
block-4-giv	66	(12)	67	(15)	228	(52)	8	(9)
block-8-giv	6	(12)	66	(17)	219	(60)	8	(9)
block-n-giv	83	(12)	61	(17)	(T)	(—)	8	(9)
block-1-pair	2	(12)	6	(11)	484	(52)	1	(6)
block-4-pair	4	(12)	7	(13)	900	(90)	2	(6)
block-8-pair	4	(12)	11	(21)	1511	(120)	2	(6)
block-n-pair	97	(24)	10	(21)	(T)	(—)	18	(16)
basic-1-giv	3	(11)	5	(12)	1224	(68)	17	(6)
basic-4-giv	131	(11)	62	(13)	160	(51)	19	(9)
basic-8-giv	132	(11)	63	(13)	185	(62)	19	(9)
basic-n-giv	54	(11)	57	(12)	(T)	(—)	19	(9)
basic-1-pair	(T)	(—)	20	(11)	816	(52)	4	(6)
basic-4-pair	18	(11)	35	(13)	(T)	(—)	5	(6)
basic-8-pair	23	(11)	56	(18)	(T)	(—)	10	(6)
basic-n-pair	209	(23)	54	(18)	(T)	(—)	77	(20)
super0-1-giv	1	(11)	11	(14)	162	(83)	2	(6)
super0-4-giv	4	(12)	6	(14)	43	(75)	2	(9)
super0-8-giv	2	(12)	6	(14)	41	(63)	2	(9)
super0-n-giv	7	(12)	4	(14)	(T)	(—)	2	(9)
super0-1-pair	4	(7)	7	(9)	72	(65)	1	(6)
super0-4-pair	2	(7)	5	(17)	120	(55)	1	(6)
super0-8-pair	2	(7)	4	(17)	149	(89)	1	(6)
super0-n-pair	13	(21)	4	(17)	(T)	(—)	7	(12)
all-1-giv	1	(11)	11	(9)	122	(73)	2	(6)
all-4-giv	4	(12)	6	(16)	23	(41)	2	(9)
all-8-giv	2	(12)	6	(16)	37	(58)	2	(9)
all-n-giv	6	(12)	4	(16)	(T)	(—)	2	(9)
all-1-pair	15	(3)	7	(9)	52	(59)	1	(6)
all-4-pair	12	(9)	5	(36)	83	(55)	1	(6)
all-8-pair	11	(9)	4	(26)	139	(90)	1	(6)
all-n-pair	7	(21)	4	(26)	(T)	(—)	6	(12)

Table 4.11
AC: Group C

Strategy	QLT-5 time	(len.)	QLT-6 time	(len.)	RBA-2 time	(len.)	D-BA-5a time	(len.)	D-BA-5c time	(len.)
start-1-giv	206	(11)	293	(8)	44	(21)	24	(7)	61	(6)
start-4-giv	237	(31)	177	(31)	16	(22)	59	(8)	59	(11)
start-8-giv	387	(11)	178	(31)	792	(25)	59	(8)	59	(11)
start-n-giv	(T)	(—)	169	(31)	17	(25)	60	(8)	60	(11)
start-1-pair	119	(21)	431	(5)	19	(19)	5	(7)	13	(18)
start-4-pair	79	(22)	713	(40)	8	(17)	7	(7)	9	(14)
start-8-pair	76	(20)	833	(50)	5	(16)	11	(7)	12	(14)
start-n-pair	(T)	(—)	(T)	(—)	4	(16)	9	(7)	10	(10)
block-1-giv	253	(11)	356	(8)	40	(21)	197	(7)	235	(6)
block-4-giv	360	(11)	212	(31)	15	(22)	118	(8)	119	(11)
block-8-giv	415	(11)	221	(31)	15	(25)	118	(8)	119	(11)
block-n-giv	(T)	(—)	208	(31)	7	(25)	118	(8)	119	(11)
block-1-pair	255	(21)	171	(56)	18	(19)	6	(7)	13	(18)
block-4-pair	33	(22)	274	(42)	7	(17)	5	(7)	6	(14)
block-8-pair	27	(20)	293	(33)	4	(16)	6	(7)	7	(14)
block-n-pair	1519	(89)	548	(14)	3	(16)	5	(7)	6	(10)
basic-1-giv	201	(11)	269	(8)	11	(22)	19	(7)	1056	(5)
basic-4-giv	236	(32)	173	(31)	7	(22)	26	(8)	27	(11)
basic-8-giv	243	(32)	173	(31)	778	(25)	26	(8)	27	(11)
basic-n-giv	(T)	(—)	164	(31)	10	(25)	26	(8)	27	(11)
basic-1-pair	240	(28)	411	(5)	6	(16)	4	(7)	10	(14)
basic-4-pair	78	(22)	701	(42)	5	(16)	7	(7)	9	(14)
basic-8-pair	75	(20)	(T)	(—)	4	(16)	11	(7)	12	(14)
basic-n-pair	(T)	(—)	(T)	(—)	4	(16)	9	(7)	10	(10)
super0-1-giv	25	(11)	37	(8)	30	(23)	6	(7)	6	(5)
super0-4-giv	58	(11)	26	(7)	14	(21)	35	(22)	2	(6)
super0-8-giv	69	(11)	26	(7)	12	(21)	21	(22)	2	(6)
super0-n-giv	(T)	(—)	26	(7)	7	(24)	15	(22)	2	(6)
super0-1-pair	36	(7)	32	(5)	13	(18)	114	(14)	51	(27)
super0-4-pair	89	(7)	62	(6)	5	(17)	40	(14)	5	(7)
super0-8-pair	82	(23)	77	(23)	3	(16)	29	(14)	3	(7)
super0-n-pair	107	(58)	89	(95)	2	(16)	21	(23)	2	(7)
all-1-giv	23	(11)	29	(8)	6	(22)	2	(7)	2	(5)
all-4-giv	53	(11)	24	(7)	6	(21)	45	(7)	5	(6)
all-8-giv	63	(11)	24	(7)	5	(21)	111	(7)	5	(7)
all-n-giv	(T)	(—)	25	(7)	3	(21)	164	(58)	5	(7)
all-1-pair	29	(7)	26	(5)	4	(16)	58	(14)	39	(7)
all-4-pair	56	(25)	53	(5)	2	(16)	12	(22)	7	(7)
all-8-pair	60	(23)	56	(22)	2	(16)	27	(9)	5	(7)
all-n-pair	89	(65)	53	(8)	1	(16)	16	(15)	3	(7)

4.7 Strategy Comparisons

The data in this section are based on the ten tables (4.2–4.11) in the preceding section. The non-AC and AC experiments are considered separately, and for most comparisons, the given and pair algorithms are considered separately. For each comparison, the strategy with the lowest proof time wins for that problem. An n-way tie counts as $1/n$ win. The numbers of wins for each strategy are listed in the tables. Although this is a crude measure (the wins are not weighted by difficulty of problem or magnitude of the win), it gives some indication of the more useful strategies.

Given vs. Pair Algorithm. Table 4.12 is a comparison of the given clause algorithm and the pair algorithm. The non-AC and AC strategies were run with both algorithms, and the number of wins is listed. Before this work, we believed that for problems without AC unification, the given clause algorithm is nearly always better and that for problems with AC unification, the pair algorithm is nearly always better. This belief was based on intuition (see 4.5.1, p. 8), and a few (unpublished) experiments with earlier versions of EQP. However, the table indicates that the pair algorithm is somewhat better, on average, for both types of problem.

Table 4.12
Winning Selection Method

	Non-AC	AC
given	259.5 (39%)	105.5 (40%)
pair	400.5 (61%)	154.5 (60%)

Best-first : Breadth-first Ratio. For each paramodulation strategy and selection algorithm, we have data for ratios n:1, for n= 1, 4, 8, and ∞. In Table 4.13, we show the number of wins, separated into non-AC and AC cases, for each value of the ratio. Note that for a given ratio and unification type, there appears to be a correlation between the given and pair algorithms. This is surprising to us because the given and pair algorithms are quite different in ordering the search. We expected more of a correlation among the ratios for a given unification type and selection algorithm.

For many years we used pure best-first search ($n = \infty$), and lately we have used $n = 4$ with good results for non-AC problems. Therefore the good performance

Table 4.13
Winning Ratios

	Non-AC		AC	
Ratio	given	pair	given	pair
1	40.3 (24%)	43.4 (26%)	29.5 (45%)	30.7 (47%)
4	48.8 (30%)	32.9 (20%)	14.2 (22%)	11.7 (18%)
8	49.3 (30%)	52.4 (32%)	6.7 (10%)	7.2 (11%)
∞	26.5 (16%)	36.3 (22%)	14.7 (23%)	15.5 (24%)

of smaller ratios surprises us. This is especially so for the AC problems, where we assumed that focusing on small clauses would be better because of the prolific nature of AC inference.

AC Axioms vs. AC Unification. Table 4.14 compares AC axioms with AC unification for the 13 problems that have AC function symbols. Strategy "ord" is excluded from the AC axioms searches, and strategy "super0" is excluded from the AC unification searches, but the "all" strategies (which include "ord" for AC axioms and "super0" for AC unification) are compared. Thus there are 208 (= 13 [problems] $\times$ 4 [ratios] $\times$ 4 [paramodulation strategies]) comparisons.

Table 4.14
Winning AC Strategy

	given	pair
AC axioms	87.5 (42%)	100.0 (48%)
AC unification	120.5 (58%)	108.0 (52%)

Functional Subsumption. We ran all 368 (=(33 [non-AC] + 13 [AC]) $\times$ 4 [ratio] $\times$ 2 [selection]) experiments with functional subsumption, but we have omitted the results, because of lack of space, and because the results are less interesting than the results of the other strategies. In summary, functional subsumption, when added to the starting strategy, had very little effect on the proof times for the AC problems. For the non-AC problems, it had, on average, a slightly positive effect; for several problems (QLT-5, QLT-6, RBA-2, WAL-1a), the effect was very positive, and for one (TBA-1-e1) the effect was very negative.

For the paramodulation strategies named "all", we included functional subsumption as well as the other paramodulation strategies.

Starting, Basic, Blocked, Ordered-Instance, and Super-0 Strategies. Table 4.15 shows one of the most important results of this work. The table lists the number of wins for each paramodulation strategy. It does not include the "all"

Table 4.15
Winning Paramodulation Strategies

Non-AC		
	given	pair
start	24.8 (19%)	20.0 (15%)
block	10.9 (8%)	8.0 (6%)
basic	81.9 (62%)	79.5 (60%)
ord	14.4 (11%)	24.5 (19%)

AC		
	given	pair
start	1.0 (2%)	0.5 (1%)
block	3.5 (7%)	8.0 (15%)
basic	4.0 (8%)	4.0 (8%)
super0	43.5 (83%)	39.5(76%)

paramodulation strategies because we wish to compare the individual paramodulation strategies with the starting strategy. The most obvious indications from the table are that basic paramodulation is widely useful for non-AC problems and that the superset limit, although incomplete, is widely useful for AC problems.

Overall Winning Strategies. Table 4.16 uses the proof times in Tables 4.2 through 4.11 to rank the 40 strategies by number of problems on which each strategy won. Here we have included the "all" paramodulation strategies. As one might expect, the "all" strategy frequently performed better than the best individual strategy. A scan of Tables 4.2 through 4.11 shows also that the "all" strategy sometimes inherits the bad properties of the worst individual strategy. For example, in problem RBA-2 in Table 4.7, basic paramodulation with the pair algorithm seems to delay proofs, and in problem D-BA-5a in Table 4.11, blocked paramodulation with the given clause algorithm seems to delay proofs.

Other Comments on the Experiments. We discuss here several properties that are not apparent in the preceding tables of winning strategies.

- Ordered-instance paramodulation had little effect on the searches. This property can be seen by comparing the "start" searches with the "ord" searches in Tables 4.2 through 4.8. We believe the reason is that with our 33 test problems most of the input and derived equations are orientable. Also, this leads us to believe that ordered-instance demodulation (using nonorientable equations as demodulators when the instance for the rewrite is orientable, also called lex-dependent demodulation) would have a small effect on the searches.

Table 4.16
Overall Winning Strategies

Non-AC				AC	
basic-1-giv	3.7 (11%)	all-4-giv	0.6	all-4-giv	2.8 (22%)
all-1-pair	3.0	start-1-giv	0.4	all-1-giv	2.7
basic-1-pair	2.5	block-8-pair	0.3	all-n-pair	1.4
basic-4-giv	2.4	basic-4-pair	0.2	all-8-giv	0.8
all-8-pair	2.2	all-1-giv	0.2	super0-1-pair	0.6
basic-8-giv	1.8	start-1-pair	0.1	all-1-pair	0.6
basic-n-pair	1.4	start-4-pair	0.1	super0-1-giv	0.5
basic-n-giv	1.3	ord-1-pair	0.1	all-n-giv	0.4
start-n-pair	1.1	block-1-pair	0.1	super0-4-pair	0.4
ord-n-pair	1.1	block-n-pair	0.1	all-4-pair	0.4
start-8-pair	1.0	start-8-giv	0.1	super0-n-pair	0.3
ord-n-giv	1.0	block-1-giv	0.1	super0-n-giv	0.3
all-8-giv	1.0	block-4-giv	0.1	super0-8-pair	0.2
basic-8-pair	0.9	block-8-giv	0.1	all-8-pair	0.2
ord-8-pair	0.8	ord-4-pair	0.1	super0-4-giv	0.2
all-4-pair	0.8	block-4-pair	0.1	super0-8-giv	0.2
all-n-pair	0.6	*all others*	0.0	block-1-pair	0.1
start-4-giv	0.6			block-1-giv	0.1
ord-4-giv	0.6			*all others*	0.0

- Blocked paramodulation usually had little effect for the non-AC problems.[10] In many cases the blocked search was slightly slower than the corresponding starting search, probably because of the time required for checking substitutions for reducibility. For the AC problems, blocked paramodulation had greater effects, both positive and negative, when compared with the starting strategy. These results were unexpected because Kapur and Zhang [7] report excellent results for blocked AC-paramodulation on ring problems.

4.8 Robbins Algebra

In this section we present two results in Robbins algebra obtained with EQP's AC unification and matching, then present a small case study on the easier result and one experiment on the more difficult result. The Robbins problem, whether every

[10] A similar observation was made by Kapur, Musser, and Narendran in [5].

associative-commutative algebra satisfying the Robbins axiom is also a Boolean algebra, was posed in the 1930s and is still open. In working on the problem, Steve Winker proved by hand two difficult lemmas [18] suggested by Larry Wos. The first is that a Robbins algebra satisfying $\exists C \exists D, C + D = C$ must be Boolean, and the second is that a Robbins algebra satisfying $\exists C \exists D, n(C + D) = n(C)$ must be Boolean. (Problem RBA-2, from the main body of experiments, is the much simpler lemma that a Robbins algebra satisfying $\exists C, C + C = C$, must be Boolean.) In November 1992, an early version of EQP proved the first lemma (in a few hours), and in February 1996, EQP proved the second lemma (in 12 days). Winker's proofs are higher order in the sense that they use induction, so EQP's proofs are seen as the first equational proofs of the lemmas. As far as we know, no other program has proved either of these lemmas. The first lemma was proved by refuting the set

$$\left\{ \begin{array}{l} n(n(n(x) + y) + n(x + y)) = y \\ C + D = C \\ x + x \neq x \end{array} \right\} \quad \text{(denial of first Robbins lemma)}$$

and the second by refuting the set

$$\left\{ \begin{array}{l} n(n(n(x) + y) + n(x + y)) = y \\ n(C + D) = n(C) \\ x + y \neq x \end{array} \right\} \quad \text{(denial of second Robbins lemma).}$$

In both cases, + is an AC function symbol, and C and D are constants. Our proofs of these two lemmas have not been previously published, and we present them in Appendix B, starting on page 40. The two Robbins lemmas are much more difficult than any of the 33 test problems, so they were not included in the main body of experiments.

Experiments with the First Robbins Lemma. For the first Robbins lemma, we specified a time limit of 4 hours and max-weight=30 (the same as in the first successful search); otherwise, the strategies are the same as in the other AC experiments. Table 4.17 contains the results (in seconds on an RS/6000 processor). How closely does this problem follow the trends apparent in the main body of experiments on the 33 test problems? Answer: *not closely.*

1. The main body of experiments indicates that the "super0" and "all" strategies are best for AC problems, but they are clearly the worst for the first Robbins lemma. We don't have an explanation for this disparity. The reason is not the incompleteness of the "super0" strategy (which is part of the "all" strategy) because it does not prevent all proofs in this case.

Table 4.17
First Robbins Lemma Results

Given		Pair	
start-1-giv	(T)	start-1-pair	(T)
start-4-giv	(T)	start-4-pair	8437
start-8-giv	(T)	start-8-pair	6675
start-n-giv	(T)	start-n-pair	4400
block-1-giv	(T)	block-1-pair	(T)
block-4-giv	(T)	block-4-pair	7625
block-8-giv	(T)	block-8-pair	6130
block-n-giv	(T)	block-n-pair	4311
basic-1-giv	3052	basic-1-pair	8054
basic-4-giv	(T)	basic-4-pair	3123
basic-8-giv	11397	basic-8-pair	2427
basic-n-giv	6593	basic-n-pair	1902
super0-1-giv	(T)	super0-1-pair	(T)
super0-4-giv	(T)	super0-4-pair	(T)
super0-8-giv	(T)	super0-8-pair	9325
super0-n-giv	(T)	super0-n-pair	(T)
all-1-giv	(T)	all-1-pair	(T)
all-4-giv	(T)	all-4-pair	(T)
all-8-giv	9484	all-8-pair	14193
all-n-giv	4972	all-n-pair	12888

2. The main body of experiments indicates that the "basic" strategy is not particularly powerful for AC problems, but it is clearly the winner for the first Robbins lemma.
3. The main body of experiments indicates that lower ratios (more breadth-first) are generally better for AC problems, but higher ratios, especially pure best-first, are clearly best for the first Robbins lemma.
4. As in the main body of experiments, the pair algorithm is clearly better than the given clause algorithm.
5. Many previous (unpublished) experiments on the first Robbins lemma by the author and by Larry Wos show that proofs are much more difficult to find with AC axioms than with AC unification. These results are consistent with the main body of experiments.

Experiments with the Second Robbins Lemma. Our only previous proof of the second Robbins lemma was with an earlier version of EQP and required 12.5 days on a 486DX2/66 processor (estimated 7.5 days on an RS/6000) with a strategy similar to "start-n-pair" with max-weight=34. Our computing environment was not stable enough to run many multi-day jobs during the period we had to run new experiments on this lemma, but we were able to obtain one meaningful comparison. The lemma was run on an RS/6000 with strategy "start-n-pair", max-weight=34, and a proof was found in 7.04 days. Then the winning strategy for the *first* Robbins lemma, "basic-n-pair", was run on the second lemma; the job ran for about 7 days without finding a proof, then the computer had to be shut down. Thus we conclude that, although "basic-n-pair" is twice as good as "start-n-pair" for the first lemma, it is no better (and possibly much worse) for the second lemma.

4.9 Conclusion

The 33 test problems come from a study in a few areas of equational logic and are mostly about lattice-like algebras and group-like algebras. The statements of the theorems are characterized mostly by small sets of small equations. We don't know how well our results and conclusions will apply to other areas such as ring-like algebras, nonequational theories, and theorems involving defined concepts.

The following results on the 33 test problems are consistent with our previous experience and beliefs about paramodulation strategies for equational logic.

- No winning strategies exist. The strategies that win in most cases are sometimes the worst strategy, and each strategy wins in some cases. This supports our long-held position that users of automated deduction systems need a variety of strategies with which to experiment.
- Basic paramodulation is widely useful for non-AC problems and is also powerful for the first Robbins lemma with AC unification.
- Although incomplete, the superset limit on AC unifiers (strategy "super0") is widely useful for AC problems.
- The tables in Section 4.6 include data on proof length. We found substantial variations in proof length, but we have not been able to draw any conclusions about which strategies lead to short proofs. We advise users seeking short proofs to consult Larry Wos's recent work [20, 19] and to experiment with a variety of strategies.

The following results are surprising to us.

- For selecting the next unit of work, we had previously assumed that the given clause algorithm is much better for non-AC problems and that the pair algorithm is much better for AC problems. But for this set of problems, paramodulation strategies, and selection ratios, the pair algorithm wins about 60% of the contests for both non-AC and AC problems. The good performance of the given clause algorithm on the AC problems was just as surprising as the very good performance of the pair algorithm on the non-AC problems.
- For the 13 problems with AC function symbols, AC axioms perform nearly as well as AC unification. However, we now believe that these 13 problems may be biased toward AC axioms because they all are about lattice-like structures and because they are not particularly difficult problems. Our experience with difficult Robbins algebra theorems (e.g., the two Robbins lemmas in the preceding section) and the experience of others with ring problems indicate that AC unification is much better than AC axioms for those areas.
- The lower selection ratios (which are closer to equal parts of best-first and breadth-first search) perform very well, especially for AC problems, where 1:1 is the winner. This observation leads us to believe that we should experiment with ratios that emphasize breadth-first search. (Experiments with pure breadth-first search have shown it to be useful in very few cases.)
- The set of results on the main body of experiments is not a good indicator for the performance of the various strategies on the first Robbins lemma, and the results on the first Robbins lemma are not a good indicator for the second Robbins lemma.
- The superset restriction for AC unifiers (strategy "super0") is incomplete, but it did not block all proofs for any of the problems to which it was applied. Although this behavior surprises us, it supports our long-held position that incomplete restrictions are extremely valuable in automated deduction.

We have several World Wide Web pages associated with this work. They are accessible from the page

`http://www.mcs.anl.gov/home/mccune/ar/33-basic-test-problems/`

and include (1) the source code of the version of EQP used for these experiments, (2) all of the input files for the 33 test problems in both Otter and EQP format, (3) the database of search results (in Prolog format) from which all of the tables in this chapter were generated, and (4) proofs of the first and second Robbins algebra lemmas.

Appendix A: Denials of the 33 Theorems

We present here the denials of the 33 theorems in clause form. A short description is given for each theorem; see [11] for further details. Variables are distinguished from constants by starting with a member of $\{u, v, w, x, y, z\}$. If "max-weight" is listed, it was used for all experiments; otherwise no limit was used in any experiment. If a symbol order is listed, it was used for all experiments; otherwise the default order was used.

CS-2. Support for Padmanabhan's conjecture on cancellative semigroups.
Max-weight=31.

$$\left\{ \begin{array}{l} (x*y)*z = x*(y*z) \\ x*(y*y) = y*(y*x) \\ A*B*A*B*A*B*A*B \neq A*A*A*A*B*B*B*B \end{array} \right\}$$

CS-6a. Support for Padmanabhan's conjecture on cancellative semigroups.
Max-weight=31.

$$\left\{ \begin{array}{l} (x*y)*z = x*(y*z) \\ x*y*z*u*v = y*z*u*v*x \\ A*B*A*B*A*B*A*B \neq A*A*A*A*B*B*B*B \end{array} \right\}$$

D-BA-1b. An independent self-dual 6-basis for Boolean algebra.
(Name abbreviated from DUAL-BA-1b.)

$$\left\{ \begin{array}{ll} (x+y)*y = y, & (x*y)+y = y \\ x*(y+z) = (y*x)+(z*x), & x+(y*z) = (y+x)*(z+x) \\ x+x' = 1, & x*x' = 0 \\ (A*B)*C \neq A*(B*C) & \end{array} \right\}$$

D-BA-2a. A basis for Boolean algebra.
(Name abbreviated from DUAL-BA-2a.) Symbol order: $1 \prec A \prec B \prec C \prec * \prec + \prec ' \prec p$.
Max-weight=28.

$$\left\{ \begin{array}{ll} (x+y)*y = y, & p(x,x,y) = y \\ x*(y+z) = (x*y)+(x*z), & p(x,y,y) = x \\ x+x' = 1, & p(x,y,x) = x \\ p(x,y,z) = (x*y')+((x*z)+(y'*z)) & \\ A+(B*C) \neq (A+B)*(A+C) & \end{array} \right\}$$

D-BA-5a. A self-dual 2-basis for Boolean algebra.
(Name abbreviated from DUAL-BA-5a.) Max-weight=23.

$$\left\{\begin{array}{ll} y+(x*(y*z))=y, & y*(x+(y+z))=y \\ ((x*y)+(y*z))+y=y, & ((x+y)*(y+z))*y=y \\ (x+y)*(x+y')=x, & (x*y)+(x*y')=x \\ x+y=y+x, & x*y=y*x \\ (x+y)+z=x+(y+z), & (x*y)*z=x*(y*z) \\ (A*B)+(A*C)\neq A*(B+C) & \end{array}\right\}$$

D-BA-5c. A self-dual 2-basis for Boolean algebra.
(Name abbreviated from DUAL-BA-5c.) Max-weight=23.

$$\left\{\begin{array}{ll} y+(x*(y*z))=y, & y*(x+(y+z))=y \\ ((x*y)+(y*z))+y=y, & ((x+y)*(y+z))*y=y \\ (x+y)*(x+y')=x, & (x*y)+(x*y')=x \\ x+y=y+x, & x*y=y*x \\ (x+y)+z=x+(y+z), & (x*y)*z=x*(y*z) \\ B+B'\neq A+A' & \end{array}\right\}$$

D-BA-8a. A self-dual 3-basis for Boolean algebra.
(Name abbreviated from DUAL-BA-8a.) Max-weight=23.

$$\left\{\begin{array}{ll} y+(x*(y*z))=y, & y*(x+(y+z))=y \\ ((x*y)+(y*z))+y=y, & ((x+y)*(y+z))*y=y \\ (x+x')*y=y, & (x*x')+y=y \\ x+x'=1, & x*x'=0 \\ (x+y)+z=x+(y+z), & (x*y)*z=x*(y*z) \\ (x*y)+((y*z)+(z*x))=(x+y)*((y+z)*(z+x)) & \\ (A*B)+(A*C)\neq A*(B+C) & \end{array}\right\}$$

LT-10a. An absorption 3-basis for lattice theory.

$$\left\{\begin{array}{l} y\wedge(x\vee(y\vee z))=y \\ ((x\wedge y)\vee(y\wedge z))\vee y=y \\ ((y\vee x)\wedge(y\vee z))\wedge y=y \\ B\vee(A\wedge(B\wedge C))\neq B \end{array}\right\}$$

LT-2. An equational version of SAM's lemma.
Max-weight=15.

$$\left\{\begin{array}{ll}
x \wedge x = x, & x \vee x = x \\
x \wedge y = y \wedge x, & x \vee y = y \vee x \\
(x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\
x \wedge (x \vee y) = x, & x \vee (x \wedge y) = x \\
0 \wedge x = 0, & 0 \vee x = x \\
1 \wedge x = x, & 1 \vee x = 1 \\
x \wedge x = x, & x \vee x = x \\
(x \wedge y) \vee (x \wedge z) = x \wedge (y \vee (x \wedge z)) & \\
C1 \vee (A \vee B) = 1, & C2 \vee (A \wedge B) = 1 \\
C1 \wedge (A \vee B) = 0, & C2 \wedge (A \wedge B) = 0 \\
(C1 \vee (A \wedge C2)) \wedge (C1 \vee (B \wedge C2)) \neq C1 &
\end{array}\right\}$$

LT-3e. Sholander's basis for distributive lattices.
Max-weight=19.

$$\left\{\begin{array}{l}
x \wedge (x \vee y) = x \\
x \wedge (y \vee z) = (z \wedge x) \vee (y \wedge x) \\
(A \vee B) \vee C \neq A \vee (B \vee C)
\end{array}\right\}$$

LT-4. The distributive law implies its dual in lattice theory.
Max-weight=17.

$$\left\{\begin{array}{ll}
x \wedge x = x, & x \vee x = x \\
x \wedge y = y \wedge x, & x \vee y = y \vee x \\
(x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\
x \wedge (x \vee y) = x, & x \vee (x \wedge y) = x \\
x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) & \\
A \vee (B \wedge C) \neq (A \vee B) \wedge (A \vee C) &
\end{array}\right\}$$

LT-5. A new self-dual form of distributivity for lattice theory.
Max-weight=23.

$$\left\{\begin{array}{ll}
x \wedge x = x, & x \vee x = x \\
x \wedge y = y \wedge x, & x \vee y = y \vee x \\
(x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\
x \wedge (x \vee y) = x, & x \vee (x \wedge y) = x \\
(((x \wedge y) \vee z) \wedge y) \vee (z \wedge x) = (((x \vee y) \wedge z) \vee y) \wedge (z \vee x) & \\
A \wedge (B \vee C) \neq (A \wedge B) \vee (A \wedge C) &
\end{array}\right\}$$

LT-6. McKenzie's basis for the lattice variety generated by N_5.
Max-weight=35.

$$\left\{\begin{array}{ll}
x \wedge x = x, & x \vee x = x \\
x \wedge y = y \wedge x, & x \vee y = y \vee x \\
(x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\
x \wedge (x \vee y) = x, & x \vee (x \wedge y) = x \\
x \wedge (y \vee (z \wedge (x \vee u))) = (x \wedge (y \vee (x \wedge z))) \vee (x \wedge ((x \wedge y) \vee (z \wedge u))) & \\
x \vee (y \wedge (z \vee (x \wedge u))) = (x \vee (y \wedge (x \vee z))) \wedge (x \vee ((x \vee y) \wedge (z \vee u))) & \\
(x \vee (y \wedge z)) \wedge (z \vee (x \wedge y)) = (z \wedge ((x \vee (y \wedge z)))) \vee (x \wedge (y \vee z)) & \\
A \wedge ((B \vee C) \wedge (B \vee D)) \neq & \\
(A \wedge ((B \vee C) \wedge (B \vee D))) \wedge ((A \wedge (B \vee (C \wedge D))) \vee ((A \wedge C) \vee (A \wedge D))) &
\end{array}\right\}$$

LT-8. Uniqueness of the meet operation in lattice theory.
Max-weight=17.

$$\left\{\begin{array}{ll}
x \wedge x = x, & x \vee x = x \\
x \wedge y = y \wedge x, & x \vee y = y \vee x \\
(x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\
x \wedge (x \vee y) = x, & x \vee (x \wedge y) = x \\
x * x = x & \\
x * y = y * x & \\
(x * y) * z = x * (y * z) & \\
x * (x \vee y) = x & \\
x \vee (x * y) = x & \\
A \wedge B \neq A * B &
\end{array}\right\}$$

LT-9b. McKenzie's absorption basis for lattice theory.
Max-weight=21.

$$\left\{\begin{array}{l}
y \vee (x \wedge (y \wedge z)) = y \\
y \wedge (x \vee (y \vee z)) = y \\
((x \wedge y) \vee (y \wedge z)) \vee y = y \\
((x \vee y) \wedge (y \vee z)) \wedge y = y \\
(A \wedge B) \wedge C \neq A \wedge (B \wedge C)
\end{array}\right\}$$

MFL-1. Moufang-1 $\Rightarrow$ Moufang-2 in loops.
Symbol order: $A \prec B \prec C \prec 1 \prec * \prec R \prec L \prec / \prec \backslash$. Max-weight=17.

$$\left\{\begin{array}{ll}
1 * x = x, & x * 1 = x \\
x * (x\backslash y) = y, & x\backslash(x * y) = y \\
(x/y) * y = x, & (x * y)/y = x \\
x * R(x) = 1, & L(x) * x = 1 \\
(x * (y * z)) * x = (x * y) * (z * x) & \\
((A * B) * C) * B \neq A * (B * (C * B)) &
\end{array}\right\}$$

MFL-2. Moufang-2 ⇒ Moufang-3 in loops.
Symbol order: $A \prec B \prec C \prec 1 \prec * \prec R \prec L \prec / \prec \backslash$. Max-weight=17.

$$\left\{ \begin{array}{ll} 1 * x = x, & x * 1 = x \\ x * (x\backslash y) = y, & x\backslash(x * y) = y \\ (x/y) * y = x, & (x * y)/y = x \\ x * R(x) = 1, & L(x) * x = 1 \\ ((x * y) * z) * y = x * (y * (z * y)) & \\ ((A * B) * A) * C \neq A * (B * (A * C)) & \end{array} \right\}$$

MFL-3. Moufang-3 ⇒ Moufang-1 in loops.
Symbol order: $A \prec B \prec C \prec 1 \prec * \prec R \prec L \prec / \prec \backslash$. Max-weight=17.

$$\left\{ \begin{array}{ll} 1 * x = x, & x * 1 = x \\ x * (x\backslash y) = y, & x\backslash(x * y) = y \\ (x/y) * y = x, & (x * y)/y = x \\ x * R(x) = 1, & L(x) * x = 1 \\ ((x * y) * x) * z = x * (y * (x * z)) & \\ (A * (B * C)) * A \neq (A * B) * (C * A) & \end{array} \right\}$$

MFL-7. Simple basis with Moufang-3.
Max-weight=21.

$$\left\{ \begin{array}{l} 1 * x = x \\ x' * x = 1 \\ ((x * y) * x) * z = x * (y * (x * z)) \\ ((A * B) * C) * B \neq A * (B * (C * B)) \end{array} \right\}$$

QLT-1. A form of distributivity for quasilattices.
Max-weight=19.

$$\left\{ \begin{array}{ll} x \wedge x = x, & x \vee x = x \\ x \wedge y = y \wedge x, & x \vee y = y \vee x \\ (x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\ (x \wedge (y \vee z)) \vee (x \wedge y) = x \wedge (y \vee z), & (x \vee (y \wedge z)) \wedge (x \vee y) = x \vee (y \wedge z) \\ x \wedge (y \vee (x \wedge z)) = x \wedge (y \vee z) & \\ A \wedge (B \vee C) \neq (A \wedge B) \vee (A \wedge C) & \end{array} \right\}$$

QLT-2. The distributive law implies its dual in quasilattices.
Max-weight=19.

$$\left\{\begin{array}{ll} x \wedge x = x, & x \vee x = x \\ x \wedge y = y \wedge x, & x \vee y = y \vee x \\ (x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\ (x \wedge (y \vee z)) \vee (x \wedge y) = x \wedge (y \vee z), & (x \vee (y \wedge z)) \wedge (x \vee y) = x \vee (y \wedge z) \\ x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) & \\ A \vee (B \wedge C) \neq (A \vee B) \wedge (A \vee C) & \end{array}\right\}$$

QLT-3. A self-dual form of distributivity for quasilattices.
Max-weight=24.

$$\left\{\begin{array}{ll} x \wedge x = x, & x \vee x = x \\ x \wedge y = y \wedge x, & x \vee y = y \vee x \\ (x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\ (x \wedge (y \vee z)) \vee (x \wedge y) = x \wedge (y \vee z), & (x \vee (y \wedge z)) \wedge (x \vee y) = x \vee (y \wedge z) \\ (((x \wedge y) \vee z) \wedge y) \vee (z \wedge x) = (((x \vee y) \wedge z) \vee y) \wedge (z \vee x) & \\ A \wedge (B \vee C) \neq (A \wedge B) \vee (A \wedge C) & \end{array}\right\}$$

QLT-4. Bowden's inequality gives distributivity in quasilattices.
Max-weight=19.

$$\left\{\begin{array}{ll} x \wedge x = x, & x \vee x = x \\ x \wedge y = y \wedge x, & x \vee y = y \vee x \\ (x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\ (x \wedge (y \vee z)) \vee (x \wedge y) = x \wedge (y \vee z), & (x \vee (y \wedge z)) \wedge (x \vee y) = x \vee (y \wedge z) \\ (x \vee (y \wedge z)) \vee ((x \vee y) \wedge z) = x \vee (y \wedge z) & \\ A \wedge (B \vee C) \neq (A \wedge B) \vee (A \wedge C) & \end{array}\right\}$$

QLT-5. Self-dual modularity axiom for quasilattices.
Max-weight=23.

$$\left\{\begin{array}{ll} x \wedge x = x, & x \vee x = x \\ x \wedge y = y \wedge x, & x \vee y = y \vee x \\ (x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\ (x \wedge (y \vee z)) \vee (x \wedge y) = x \wedge (y \vee z), & (x \vee (y \wedge z)) \wedge (x \vee y) = x \vee (y \wedge z) \\ (x \wedge y) \vee (z \wedge (x \vee y)) = (x \vee y) \wedge (z \vee (x \wedge y)) & \\ A \wedge (B \vee (A \wedge C)) \neq (A \wedge B) \vee (A \wedge C) & \end{array}\right\}$$

QLT-6. A modularity axiom for quasilattices.
Max-weight=23.

$$\left\{\begin{array}{ll} x \wedge x = x, & x \vee x = x \\ x \wedge y = y \wedge x, & x \vee y = y \vee x \\ (x \wedge y) \wedge z = x \wedge (y \wedge z), & (x \vee y) \vee z = x \vee (y \vee z) \\ (x \wedge (y \vee z)) \vee (x \wedge y) = x \wedge (y \vee z), & (x \vee (y \wedge z)) \wedge (x \vee y) = x \vee (y \wedge z) \\ ((x \vee y) \wedge z) \vee y = ((z \vee y) \wedge x) \vee y & \\ A \wedge (B \vee (A \wedge C)) \neq (A \wedge B) \vee (A \wedge C) & \end{array}\right\}$$

RBA-2. A Robbins algebra with an idempotent element is Boolean.
Max-weight=21.

$$\left\{\begin{array}{l} x + y = y + x \\ (x + y) + z = x + (y + z) \\ n(n(n(x) + y) + n(x + y)) = y \\ c + c = c \\ n(A + n(B)) + n(n(A) + n(B)) \neq B \end{array}\right\}$$

SD-2a. Intersection in terms of set difference.
Symbol order: $A \prec B \prec C \prec - \prec *$. Max-weight=23.

$$\left\{\begin{array}{l} x - (y - x) = x \\ x - (x - y) = y - (y - x) \\ (x - y) - z = (x - z) - (y - z) \\ x * y = x - (x - y) \\ (A * B) * C \neq A * (B * C) \end{array}\right\}$$

SD-3-e1. A simple basis for set difference.

$$\left\{\begin{array}{l} x - (y - x) = x \\ x - (x - y) = y - (y - x) \\ (x - y) - z = (x - z) - (y - z) \\ (A - C) - B \neq (A - B) - C \end{array}\right\}$$

SD-3-e2. A simple basis for set difference.
Max-weight=23.

$$\left\{\begin{array}{l} x - (y - x) = x \\ x - (x - y) = y - (y - x) \\ (x - y) - z = (x - z) - y \\ (A - B) - C \neq (A - C) - (B - C) \end{array}\right\}$$

TBA-1-e1. A short single axiom for ternary Boolean algebra.
Max-weight=26.

$$\left\{ \begin{array}{l} f(f(v,w,x),y,f(v,w,z)) = f(v,w,f(x,y,z)) \\ f(y,x,x) = x \\ f(x,y,g(y)) = x \\ f(x,x,y) = x \\ f(g(y),y,x) = x \\ f(f(A,g(A),B),g(f(f(C,D,E),F,f(C,D,G))),f(D,f(G,F,E),C)) \neq B \end{array} \right\}$$

TBA-1-e2d. A short single axiom for ternary Boolean algebra.
Max-weight=50.

$$\left\{ \begin{array}{l} f(f(x,g(x),y),g(f(f(z,u,v),w,f(z,u,v6))),f(u,f(v6,w,v),z)) = y \\ f(A,A,B) \neq A \end{array} \right\}$$

WAL-1a. A relationship between lattices and weakly associative lattices.
Max-weight=17.

$$\left\{ \begin{array}{ll} x \wedge x = x, & x \vee x = x \\ x \wedge y = y \wedge x, & x \vee y = y \vee x \\ ((x \vee z) \wedge (y \vee z)) \wedge z = z, & ((x \wedge z) \vee (y \wedge z)) \vee z = z \\ x \wedge (y \vee (x \vee z)) = x & \\ (A \wedge B) \wedge C \neq A \wedge (B \wedge C) & \end{array} \right\}$$

WAL-2. Uniqueness of the meet operation in weakly associative lattices.
Max-weight=17.

$$\left\{ \begin{array}{ll} x \wedge x = x, & x \vee x = x \\ x \wedge y = y \wedge x, & x \vee y = y \vee x \\ ((x \vee z) \wedge (y \vee z)) \wedge z = z, & ((x \wedge z) \vee (y \wedge z)) \vee z = z \\ x * x = x & \\ x * y = y * x & \\ ((x \vee z) * (y \vee z)) * z = z & \\ ((x * z) \vee (y * z)) \vee z = z & \\ A \wedge B \neq A * B & \end{array} \right\}$$

Appendix B: Proofs of the Two Robbins Lemmas

We present here proofs of the two Robbins algebra lemmas discussed in Section 4.8.

The first lemma was proved in 1992 with an early version of EQP on a SPARC 2 with a strategy similar to our starting strategy, with the pair algorithm, pure

best-first pair selection, and max-weight=30; the proof was found in about 2 hours and used 8 megabytes of memory.[11] The second lemma was proved with a similar strategy (max-weight=34) in 1996 with a recent version of EQP on a 486/DX2-66 (Linux); the proof was found in about 12.5 *days* and used 6 megabytes.[12]

Each derived equation in the proofs below has a justification. The notation "$m \to n$" indicates paramodulation from the left side of m into the left side of n, "$: i, j, k, \cdots$" indicates rewriting with the demodulators $i, j, k, \cdots$, and "n'" indicates the extension of n.

First Robbins Lemma

$$\left\{ \begin{array}{l} x + y = y + x \\ (x + y) + z = x + (y + z) \\ n(n(n(x) + y) + n(x + y)) = y \\ \exists C \exists D \ (C + D = C) \end{array} \right\} \Rightarrow \exists x (x + x = x)$$

Proof (found on cosmo.mcs.anl.gov at 7801.45 seconds, Nov. 17, 1992).

1	$C + D = C$	
2	$n(n(n(x) + y) + n(x + y)) = y$	
4	$x + x \neq x$	
11	$n(n(C) + n(D + n(C))) = D$	$[1 \to 2]$
17	$n(n(C + x + y) + n(D + n(C + x) + y)) = D + y$	$[1' \to 2]$
29	$n(D + n(C + n(D + n(C)))) = n(D + n(C))$	$[11 \to 2]$
36	$n(n(n(n(x) + y) + n(x + y) + z) + n(z + y)) = z$	$[2 \to 2]$
37	$n(n(n(n(x) + y) + x + y) + y) = n(n(x) + y)$	$[2 \to 2]$
76	$n(n(C) + n(D + n(C + n(x)) + n(C + x))) = D$	$[1 \to 36]$
160	$n(n(D + n(C + n(D + n(C))) + x) + n(n(D + n(C)) + x)) = x$	$[29 \to 2]$
161	$n(n(C + n(D + n(C))) + n(D + n(C))) = D$	$[29 \to 2 : 1]$
203	$n(n(C + n(C + n(D + n(C)))) + n(C + n(D + n(C)))) = C$	$[1' \to 160 : 1,1,1]$
206	$n(D + n(D + n(C) + n(C + n(D + n(C))))) = n(C + n(D + n(C)))$	$[161 \to 2]$
302	$n(D + n(n(C) + n(n(D + n(C)) + n(x)) + n(n(D + n(C)) + x))) = n(C)$	$[11 \to 36]$
304	$n(n(n(n(n(x) + y) + x + y) + z + y) + n(n(n(x) + y) + z)) = z$	$[37 \to 2]$
465	$n(C + n(D + n(C))) = n(C)$	$[1' \to 302 : 1,11,206]$
466	$n(n(C) + n(C + n(C))) = C$	$[203 : 465,465]$
470	$n(n(C + x) + n(n(C) + n(C + n(C)) + x)) = x$	$[466 \to 2]$
516	$n(n(C + C + n(C + n(C))) + n(C + n(C))) = C$	$[466 \to 37 : 466]$
585	$n(C + n(C + n(C) + n(C + C + n(C + n(C))))) = n(C + C + n(C + n(C)))$	

[11] The SPARC 2 runs EQP about 0.34 as fast as the RS/6000 processors used for the main body of experiments.

[12] The 486 runs EQP about 0.57 as fast as the RS/6000 processors.

		$[516 \to 2]$
1432	$n(C + C + n(C + n(C))) = n(C)$	$[466 \to 304 :585]$
1452	$D + n(C + n(C)) = D$	$[1432 \to 17 :76]$
1472	$C + n(C + n(C)) = C$	$[1452' \to 1' :1]$
1531	$n(C + n(C)) + x = x$	$[1472' \to 2 :470]$
1532	$\square$	[1531,4]

Second Robbins Lemma

$$\left\{ \begin{array}{l} x + y = y + x \\ (x + y) + z = x + (y + z) \\ n(n(n(x) + y) + n(x + y)) = y \\ \exists C \exists D \ (n(D + C) = n(C)) \end{array} \right\} \Rightarrow \exists x \exists y (x + y = y)$$

Proof (found on gyro.mcs.anl.gov at 1081324.76 seconds, Feb. 18, 1996).

2	$x + y \neq y$	
4	$n(D + C) = n(C)$	
5	$n(n(n(x) + y) + n(x + y)) = y$	
7	$n(n(D + C + x) + n(n(C) + x)) = x$	$[4 \to 5]$
8	$n(n(C) + n(D + n(C))) = D$	$[4 \to 5]$
10	$n(n(D + x) + n(n(C) + n(D + n(C)) + x)) = x$	$[8 \to 5]$
12	$n(D + n(C + n(D + n(C)))) = n(D + n(C))$	$[8 \to 5]$
21	$n(n(n(n(x) + y) + x + y) + y) = n(n(x) + y)$	$[5 \to 5]$
28	$n(n(D + C + n(n(C) + x) + x) + x) = n(n(C) + x)$	$[4 \to 21 :4]$
38	$n(n(n(D + C + x) + n(n(C) + x) + y) + n(x + y)) = y$	$[7 \to 5]$
67	$n(n(D + n(C)) + n(D + C + n(D + n(C)))) = D$	$[12 \to 5]$
144	$n(n(n(n(x) + y) + x + y + y) + n(n(x) + y)) = y$	$[21 \to 5]$
148	$n(n(n(n(x) + y) + n(x + y) + n(y + z) + z) + z) = n(y + z)$	$[5 \to 21 :5]$
173	$n(D + n(D + n(C) + n(D + C + n(D + n(C))))) = n(D + C + n(D + n(C)))$	$[67 \to 5]$
290	$n(n(n(n(n(x) + y) + x + y + y) + n(n(x) + y) + z) + n(y + z)) = z$	$[144 \to 5]$
301	$n(D + C + n(D + n(C))) = n(C)$	$[8 \to 38 :8,173]$
829	$n(n(D + n(C)) + n(D + n(D + n(C)) + n(n(C) + n(x)) + n(n(C) + x))) = D$	$[8 \to 148 :8]$
1237	$n(n(n(n(n(x) + y) + n(x + y) + n(y + z) + z) + z + u) + n(n(y + z) + u)) = u$	$[148 \to 5]$
1478	$n(D + D + C + n(D + n(C))) = n(C)$	$[28 \to 10 :290]$
1501	$n(n(D + D + C + n(D + n(C)) + n(n(C) + x) + x) + x) = n(n(C) + x)$	$[1478 \to 21 :1478]$
3321	$n(D + n(D + n(C) + n(D + n(D + n(C)) + n(n(C) + n(x)) + n(n(C) + x)))) = n(C)$	$[8 \to 1237]$
3710	$n(D + n(D + n(C)) + n(n(C) + n(x)) + n(n(C) + x)) = n(C)$	

$[829 \rightarrow 38 : 301,8,3321]$

3711 $n(D + n(D + n(C)) + n(D + C + n(C)) + n(n(C) + n(C))) = n(C)$ $[4 \rightarrow 3710]$

3736 $D + n(D + n(C)) + n(n(C) + n(C)) = n(n(C) + n(C))$ $[3711 \rightarrow 5 : 1501]$

3737 $\square$ [3736,2]

References

[1] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 607*, pages 462–476. Springer-Verlag, 1992.

[2] M. Bonacina and W. McCune. Distributed theorem proving by Peers. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 814*, pages 841–845. Springer-Verlag, 1994. Extended abstract.

[3] J. Hsiang and M. Rusinowitch. On word problems in equational theories. In T. Ottmann, editor, *Proceedings of 14th ICALP, Lecture Notes in Computer Science, Vol. 267*, pages 54–71. Springer-Verlag, 1987.

[4] G. Huet. An algorithm to generate the basis of solutions to homogeneous linear Diophantine equations. *Information Processing Letters*, 7:144–147, 1978.

[5] D. Kapur, D. Musser, and P Narendran. Only prime superpositions need be considered in the Knuth-Bendix completion procedure. *J. Symbolic Computation*, 6:19–36, 1988.

[6] D. Kapur and H. Zhang. RRL: Rewrite rule laboratory user's manual. Technical Report 89-03, Department of Computer Science, University of Iowa, 1989.

[7] D. Kapur and H. Zhang. A case study of the completion procedure: Proving ring commutativity problems. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, chapter 10, pages 360–394. MIT Press, 1991.

[8] D. Lankford. Canonical inference. Tech. Report ATP-32, Dept. of Mathematics, University of Texas at Austin, 1975.

[9] W. McCune. Experiments with discrimination tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992. Invited paper.

[10] W. McCune. OTTER 3.0 Reference Manual and Guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.

[11] W. McCune and R. Padmanabhan. *Automated Deduction in Equational Logic and Cubic Curves*, volume 1095 of *Lecture Notes in Computer Science (AI subseries)*. Springer-Verlag, Berlin, 1996.

[12] W. McCune and L. Wos. Experiments in automated deduction with condensed detachment. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 607*, pages 209–223, Berlin, 1992. Springer-Verlag.

[13] R. Niewenhuis and A. Rubio. Basic superposition is complete. In B. Krieg-Brückner, editor, *Proceedings of the European Symposium on Programming, Lecture Notes in Computer Science, Vol. 582*, pages 371–389, Berlin, 1992. Springer-Verlag.

[14] R. Overbeek. *A new class of automated theorem-proving algorithms.* PhD thesis, Pennsylvania State University, 1971.

[15] G. E. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM J. Computing*, 12(1):82–100, 1983.

[16] G. Robinson and L. Wos. Paramodulation and theorem-proving in first-order theories with equality. In D. Michie and R. Meltzer, editors, *Machine Intelligence, Vol. IV*, pages 135–150. Edinburgh University Press, 1969.

[17] M. Stickel. A unification algorithm for associative-commutative functions. *J. ACM*, 28(3):423–434, 1981.

[18] S. Winker. Robbins algebra: Conditions that make a near-Boolean algebra Boolean. *Journal of Automated Reasoning*, 6(4):465–489, 1990.

[19] L. Wos. Automating the search for elegant proofs. *J. Automated Reasoning*, 1996. To Appear.

[20] L. Wos. *The Automation of Reasoning: An Experimenter's Notebook with Otter Tutorial.* Academic Press, New York, 1996.

[21] L. Wos, D. Carson, and G. Robinson. The unit preference strategy in theorem proving. In *AFIPS Proceedings 26*, pages 615–621. Spartan Books, 1964.

[22] L. Wos, G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem proving. *J. ACM*, 14(4):698–709, 1967.

Index