Controlling Redundancy in Large Search Spaces: Argonne-Style Theorem Proving Through the Years*

Ewing L. Lusk

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439, U.S.A.
lusk@mcs.anl.gov

Abstract. Argonne National Laboratory has been a center of research in automated theorem proving for nearly thirty years. In that time Argonne scientists and collaborators have developed a long series of theorem-proving systems with which to explore this fascinating field. Since the first systems of the mid-sixties, many features have changed of course, but others, perhaps the most fundamental, have remained the same. We survey here some of the history of those theorem-proving programs from the earliest to the most current, focusing in particular on the fundamental approach that has remained basically unchanged as a framework for the implementation of new ideas as they have come along. We will also try to identify the strengths and weaknesses of each of the systems by describing some of the problems that each could solve and could not solve, thus motivating the ideas for the next system.

1 Introduction

Automated theorem proving started at Argonne in the early nineteen-sixties and remains a very active area of research there today. The theorem-proving systems developed during that last nearly thirty years have, in each generation, been among the most capable systems in the world, as new ideas have been incorporated into whatever the current program has been. Periodically programs have been replaced with entirely new ones, as the weight of accumulated additions and changes in computer science and computing as a whole have made completely new systems desirable.

Throughout all this change, however, certain ideas have remained constant. Some of them have been philosophical, such as the use of specific problems to motivate progress and the maintenance of a single, powerful current system, incorporating all of the features developed up to that point. In this paper we would like to focus on one central technical aspect that has remained exactly the same through multiple generations of systems and has perhaps been overlooked as a source of the success of the Argonne systems. We will call this feature *the closure approach* for reasons that will become apparent below. It is at the heart of all of the Argonne programs.

In this paper we will first explain in detail exactly what the closure approach is, and then illustrate how it has been able to continue to serve as a framework for a

* This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

very wide variety of new ideas in automated reasoning. We will do the illustrating with a historical review of the major families of system that have been built and used at Argonne over the years. We will conclude by describing some current work that continues to be based on the same theorem-proving paradigm.

2 Large Search Spaces and the Problem of Redundancy Control

All those who have worked on theorem proving problems in nontrivial domains are familiar with the combinatorial explosion of the size of the search space. Indeed the inevitability of this combinatorial explosion is periodically cited by theorists as the reason why automated theorem proving is a doomed area of research. In most significant systems the search space explodes in two different ways which it is important to distinguish:

1. The number of distinct derivable formulas not only is infinite, but increases faster with the depth.
2. The number of ways in which each of these formulas can be derived from the initial axioms increases along with the proof depth.

The basic idea behind the closure approach is that the second problem is far worse that the first, and that if it can be successfully attacked, then the first problem can be dealt with.

We define the closure approach to be the fundamental paradigm in which formulas that are derived from the initial axioms *are kept*, and thus are available for use in controlling the growth in the size of the search space. This approach has advantages and disadvantages, corresponding to the fact that keeping clauses both creates and solves the problem of redundancy control.

The principal advantage is that once a formula is derived, it is available for use thereafter whenever needed, and need not be derived again when it appears in another subproof. The two most powerful techniques are subsumption and demodulation, each of which have forward and backward variations. The principal disadvantage is that the actual derivation process cannot be made as fast as it can, for example, by the use of Prolog-style compilation techniques. In addition, some overhead is incurred in keeping the newly-derived formulas and indexing them for later use. The rest of this paper is an historical explanation of how the advantages of the closure approach have outweighed the disadvantages. In the process, we will reveal another important advantage: that the closure approach to theorem proving is extraordinarily flexible in that it can accommodate a very wide variety of variations that have been contrived for it over the years.

3 Early Argonne Provers 1963–1970

The first phase of Argonne theorem proving started in 1963. Over the following seven years, two major systems were constructed. These systems supplied the tools for the very first experiments with the new resolution principle, and set the tradition of exploring the applicability of theorem proving to problems in mathematics and logic.

3.1 The First Argonne Theorem Prover

Shortly after the formulation of the resolution principle, Dan Carson and Larry Wos developed a resolution theorem prover they called P1 (for "Program 1"). It was based on the closure approach, with forward subsumption only, and used binary resolution with factoring. It also had unit preference[27], an ordering strategy that is one of the few features not carried on into subsequent systems. The set-of-support strategy[29] and demodulation were developed during experimentation with this program. Thus from the very beginning the closure approach was not applied blindly, but in conjunction with a strategy for exploring the more relevant parts of the search space earlier than the less relevant parts. As we will see when we examine it in detail, the closure approach itself makes the application of such strategies quite natural while still maintaining completeness.

It was implemented in IBM 704 assembly language. It could prove, once the set of support strategy was added to the program, that groups in which every element has order 2 are commutative, that $x \cdot 0 = 0$ in a ring, and that $(-x)(-y) = xy$ in a ring, although for this last problem it needed the lemma that $x \cdot 0 = 0$.

3.2 Paramodulation

It was experimentation with P1 that led to Wos's invention of the paramodulation inference rule[22]. Wos began collaboration with George Robinson, who was the principal architect of the next system, called RW1 ("Robinson-Wos 1"). This system was used in the late sixties to experiment with the new paramodulation inference rule as well as demodulation[22]. (This work preceded, but apparently did not influence, the work of Knuth and Bendix, who independently formulated paramodulation and demodulation, and then applied them to the notion of a complete set of reductions in their 1970 paper[5]). RW1 could do the equality versions of the problems P1 could do, as well as showing the dependence of the fourth and fifth axioms in a ternary boolean algebra on the first three axioms. RW1 also had all the capabilities of P1.

One problem that was too hard for RW1 was the "the commutator problem", which states that in a group in which $x^3 = e$, $[x, y], y] = e$, where $[x, y]$ represents the commutator of x and y. Wos had proofs of this theorem, one with binary resolution (138 steps), and another using paramodulation (46 steps), but the problem could not be done automatically using the available technology. One reason was the lack of back demodulation.

4 The Early Overbeek Systems

4.1 Thesis

Meanwhile, and independently, Ross Overbeek was completing his dissertation at Penn State under the direction of Gene Singletary. Part of that work involved the implementation of a high-performance theorem prover, and a proof of the commutator problem was his goal. An interesting feature of this prover[20] was that although complete for first-order logic, it did not use functions. It was called FTP (for Functionless Theorem Prover), and introduced a number of features for the first time that have remained important components of the Argonne family of theorem-proving systems. It had the first efficient implementation of hyperresolution, FPA indexing, forward and backward subsumption, and weighting. It used the following algorithm for its basic loop, which remained exactly the same through several generations of systems to come.

We assume here that we start with a set of clauses divided into two sets, which we call the "axioms" and the "set of support". The following is what we refer to throughout this paper as the *closure algorithm*.

```
While (the null clause has not been produced and the set of support is not empty)
        Choose a clause from the set of support, call it the given clause, and move it
                to the axioms
        Generate all clauses that can be deduced from the given clause and
                other clauses in the axiom set
        For each new generated clause
                Process it (rewrite to canonical form, merge literals, etc.)
                Test whether it is subsumed by any existing clause
                Perform any other desired filtering (e.g. weighting)
                If the new clause survives
                        Add it to the set of support
                end if
                Perform back subsumption with new clause
                Rewrite existing clauses with it if it is a demodulator
        end for
end while
```

This algorithm, which at the time was thought of as just one implementation of the set of support strategy, owes its long life to its (perhaps unforeseen) great adaptability. Nearly all the additions, modifications, and refinements of the next twenty years have fit into the framework of this simple algorithm. Even the parallel algorithms described in Section 8 can be seen as small variations on this theme.

The closure algorithm computes the closure of a set of clauses under the operation of a set of inference rules, keeping an eye out for the empty clause. It is different from various other theorem-proving algorithms in two important ways.

- It differs from the "natural deduction" or "agenda" approaches in that at each step it generates all consequences of the given clause.
- It differs from the "Prolog technology" family of approaches in that (processed) new clauses are stored in an ever-growing database. It is considered that a lot of work went into computing them, and this work should not have to be repeated. The algorithm stops after each step to decide what to work on next.

There are many places for user control and the operation of heuristics: in the choice of the initial set of support, the choice of inference rules, the choice of given clause at the beginning of each iteration, and the choice of what to keep during post-processing.

The algorithm has thus been able to be used in a wide variety of contexts: showing that a set of clauses is unsatisfiable, executing the Knuth-Bendix completion procedure, computing the sizes of certain algebraic structures[12], designing circuits, and searching for fixed-point combinators[19]. What this says is that many problems can be usefully expressed in terms of computing (part of) the logical closure of a set of clauses.

FTP could do the commutator problem, thus fulfilling its motivating goal. It used hyperresolution rather than paramodulation, and made extensive use of weighting. It could also prove that subgroups of index 2 are normal and that Boolean rings are commutative, again without using paramodulation, although demodulation played an important role. These problems remain difficult today for many systems.

4.2 The First NIU System

In 1971, Overbeek came to Northern Illinois University, not far from Argonne, met Wos, and redesigned FTP in order to merge the two lines of development represented by RW1 and FTP. The new system was called WOS1. It contained all the features of both of its predecessors, thus starting another tradition of the Argonne family, that (for the most part) each new system would not be a small system designed to illustrate or validate a single idea, but rather the union of all that had gone before, in order to accumulate a really powerful system. Thus WOS1 had function symbols and paramodulation, hyperresolution and weighting, and was based on the closure algorithm. Its implementation, particularly the sophisticated indexing method (FPA lists) that made the speed of the unification algorithm largely irrelevant, is described in [21].

This system was used for an extensive set of experiments, on all the problems that the group could assemble, and the results published, along with the precise statements of the problems[13]. This paper has served as a model for subsequent collections of problems. It represents a snapshot of the capabilities of the Argonne system of the early seventies. The paper was presented at the Argonne Workshop on Automated Theorem Proving, which was retroactively named CADE-1. The system ran on the supercomputer of its day, the IBM 360/195, and was written in 40,000 lines of assembly language. Its capabilities and limitations are precisely documented in [13].

It was thought by some that the power of this system came primarily from its implementation in assembly language. This was not true. Assembly language provided the most convenient method of the time to implement sophisticated data structures, which provided the "secret" of the system's performance. High performance without redundancy control just makes the combinatorial explosion occur faster. It was the use of the closure algorithm and data structures to support fast subsumption and resolution that underlay the capabilities of that system.

5 The NIU Systems: a Bilingual Approach

The group at Northern Illinois University expanded to include John McCharen and Steve Winker, and eventually Ewing Lusk. In order to facilitate the increased diversity of experiments that the larger group wanted to undertake, it as decided to restructure WOS1 into a toolkit of assembly-language subroutines, and rewrite the upper levels, including the closure algorithm, in PL/1. (IBM machines were still the primary computers, both at Argonne and NIU.) This led to a series of systems that lasted nearly seven years, named NIUTP1–NIUTP7.

The notion of a toolkit with which one could build theorem provers was introduced at this time and became another theme for later Argonne development. In this case the approach was used to build a series of systems incorporating ever more complex variations on the closure algorithm without changing the underlying data structures and inference functions. The ultimate system (NIUTP7) provided a set of user-definable theorem-proving "environments", each running a version of the closure algorithm with different controlling parameters, and a meta-language for controlling their interactions[30]. There was enough control, and there were enough built-in predicates, that it became possible to "program" the theorem prover to perform a number of symbolic computational tasks. With these systems Winker and Wos began the systematic attack on open problems[25, 26, 32, 31] and the work on qualification and locking was done[10]. The systems of this era were all built on the closure approach, although the PL/1 layer made it possible to rapidly customize it in order to apply it in unorthodox ways.

6 Portability, Interactivity, and Distribution: LMA

By the late 1970's, computing had changed in fundamental ways, and it was felt that the theorem-proving effort would benefit from a complete redesign of the system that would make it more robust. For one thing, the rise of the science of software engineering had contributed new knowledge about how to structure large systems. For another, portability and ease of use were becoming standard requirements for software, and the Argonne-NIU group wanted to be able to share their prover with other researchers in the area. Thirdly, computer systems were becoming more interactive.

Lusk and Overbeek decided to re-implement the system from scratch, with an eye to all of these concerns. A layered library approach was chosen, with Pascal as the implementation language. (A strange choice, perhaps, but one needs to recall that at that time, Lisp was not yet fast, C was not yet public, and Ada looked promising as a new standard portable language that would include multiprocessing facilities. Lusk and Overbeek thought that they would probably eventually translate the system into Ada, and the syntactically similar Pascal would make it easier to do this automatically.)

The new system, written by Lusk, McCune, and Overbeek, was called LMA (Logic Machine Architecture) and had three distinct layers. The lowest layer[8] supplied a set of objects that eliminated the limits that had been encountered in extending the NIUTP systems into new domains. LMA would have no built-in limits on the number of arguments per term or literals per clause. The first layer also supplied indexing and unification services. The second layer[7] supplied the inference rules and demodulation and subsumption functions. The third layer[11] contained the basic closure algorithm, which could be controlled interactively.

LMA exhibited the classic "second system" effect, in which a system contains every desirable feature and thus becomes cumbersome. It had Lisp and Prolog interfaces (including a built-in Prolog system callable in the middle of an inference), floating-point numbers, and an interactive help system. It was portable to Unix, VMS, and CMS operating systems, although it was most used on a VAX 11/780. The combination of its VAX home and its many data-hiding features made it slower than the earlier systems, but it could still do all of the old problems because of its extensive facilities for redundancy control. It was used for exploring areas outside mathematics and logic, such as puzzles, circuit design, etc. It could be used interactively to construct proofs that it could not find unaided.

LMA started another tradition. It was the first system to be widely distributed to others, along with documentation. Over 150 copies were given to outside users. John Kalman wrote a tutorial and workbook[4] to help the novice user.

7 Back to Basic Speed

7.1 OTTER

After Bill McCune joined Argonne in 1984, he undertook a complete rewrite of the system during 1987-1988 in order to recapture the speed of the early systems and provide a tool that could absorb new theory and new implementation techniques he wanted to experiment with. This system became OTTER[14], still the system most used at Argonne, and also in use by many others around the world.

OTTER discarded many of the LMA features designed for flexibility in order to be a small, fast system. It is currently a batch-only system, and it imposes limits on term sizes, etc., although it is easy to change these at compile time. It is written in C. Two notable features are indexing via discrimination trees for forward subsumption (See [16] for a full discussion of indexing issues.) and the use of lexical recursive path ordering for rewrite rule manipulation.

OTTER has been used in a number of investigations that have led to the solutions of open problems, particularly in the areas of combinatory logic[19] and axiom systems for groups[17, 15]. It was also used by Overbeek in the solution of open questions posed in Smullyan's *To Mock a Mockingbird*. In fact, for a time OTTER was installed in such a way that it could answer email that asked it to solve certain problems about combinatory logic. OTTER's speed has been put to good use as many of the single axiom problems required submitting thousands of OTTER runs in an automated way. OTTER is very widely distributed, being available both by anonymous `ftp` and on a diskette bound into the book [28]. It runs on IBM-compatible personal computers and Macintoshes as well as on most Unix machines. Very recent enhancements include a graphics-based tool for manipulating formulas[2] and a customizable Prolog-style operator input language extension[18].

7.2 FDB

OTTER is not particularly conveniently structured as a library. While Overbeek was visiting the University of Texas in 1989, he wrote a library of formula management routines called FDB (Formula DataBase), in order to make some of the new indexing techniques available in library form. He and Ralph Butler turned this into a theorem prover that was competitive with OTTER in speed[1]. It is a good place to start for someone who wants to build his own system without re-implementing basic indexing routines and inference rules.

8 Parallelism

The late nineteen-eighties saw the impact of parallel computing on almost all aspects of computer science. Automated theorem proving offers the potential for doing a great deal of its computations in parallel, although the maintenance of a coherent state of knowledge dictated by the closure approach makes the problem non-trivial, and the problem of redundancy control is accentuated. Indeed, parallelism in symbolic fields such as theorem proving can be considerably more challenging than in many traditional numerical areas, where regular patterns of uniform parallelism are common. Two quite different approaches to parallelizing the closure approach have been tried so far, one for each of the two major computational models for MIMD computation. In each case, a modification of the closure algorithm is needed, but not a fundamental change.

8.1 A Parallel Theorem Prover for Distributed-Memory

Anita Jindal's dissertation[3] included a parallel implementation of OTTER based on the distributed memory computational model. Ralph Butler and Ross Overbeek also did a version based on FDB. The idea is to specialize the work to be done by separate processes that communicate via messages. Thus some processes derive new clauses and send them to subsumption processes, who filter the derived clauses and send the survivors on to a master process which then distributes new work back to the generating processes. Each process maintains a copy of the database of clauses that has been used as given clauses. The separate copies of the database can become slightly different from one another, but by by having the master process perform a final subsumption test prior to updating his copy of the database, consistency and redundancy control are maintained. The system has obtained good speedups on large problems.

8.2 Shared-Memory

In [24] Slaney and Lusk focused on parallelizing the closure algorithm in a completely general setting. The application of this parallel algorithm to OTTER yielded ROO[9], a shared-memory parallel theorem prover completely compatible with OTTER, including back subsumption and back demodulation. In ROO, there is only one copy of the formula database, and the parallelism comes from parallelizing the outer loop of the closure algorithm. The complicated part is ensuring that separate processes do not add the same newly derived clause to the database. That is, once more the basic issue is redundancy control. For details of how this problem is solved, see [9].

An extensive set of experiments, together with clauses for all the problems, appears in [6]. ROO, running on a Sequent symmetry, is currently the fastest system in use at Argonne. A planned port to the Alliant FX/2800 will make it substantially faster.

9 Work in Progress

In [23], Slaney presents a system for generating small finite models, called FINDER. The system is very fast, using sophisticated data structures and intelligent backtracking. We (Slaney, Lusk, and McCune) are currently incorporating FINDER into OTTER. Finite models fit neatly into the closure algorithm as a filter on allowable inferences. FINDER is so fast that it can generate new models during the OTTER run and test every newly derived formula to ascertain whether or not it is true in the current model. Preliminary experiments show that dynamic model generation and testing allow proofs to be obtained with far fewer generated clauses than OTTER on its own would generate, although more work is needed before this can be translated into a more than a modest gain in speed. It also remains unclear just how to apply this idea in the parallel setting. We call this system SCOTT (for Semantically Constrained OTTER).

10 Future Work

Automated theorem proving remains an active area of research at Argonne. In addition to continuing application of our existing systems to open problems in various domains, a number of research projects involve enhancements to OTTER and ROO. The first is AC unification, which is orthogonal to the closure approach. The next is induction, which will involve some modifications to the closure algorithm. We expect to (re)introduce interaction, in order to direct the system by hand when it is appropriate. And finally, we intend to address problems that require reasoning at multiple levels (the definition-unfolding problem). It remains to be seen just exactly how the closure algorithm itself will be modified, but we intend to preserve the overall approach.

11 Summary

In this history of Argonne-style theorem proving we have focused on the common theme that has run through all the systems in the extensive family: the use of the closure approach to theorem proving. It is this approach that provides the redundancy control necessary for successful navigation of the large search spaces that characterize significant problems.

References

1. R. Butler and R. Overbeek. A tutorial on the construction of high-performance resolution/paramodulation systems. Tech. Report ANL-90/30, Argonne National Laboratory, Argonne, IL, 1990.
2. T. Henry and W. McCune. FormEd: An X Window System application for managing first-order formulas. Tech. Memo ANL/MCS-TM-141, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, October 1990.

3. A. Jindal, R. Overbeek, and W. Cabot. Exploitation of parallel processing for implementing high-performance deduction systems. *Journal of Automated Reasoning*, 8(1):23–38, 1992.
4. John Kalman. An itp workbook. Technical Report ANL–86–56, Argonne National Laboratory, December 1986.
5. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebras.* Pergamon Press, 1970.
6. E. Lusk and W. McCune. Experiments with Roo, a parallel automated deduction system. Preprint MCS-P254-0791, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1991. To appear in Springer-Verlag Lecture Notes in Artificial Intelligence.
7. E. Lusk, W. McCune, and R. Overbeek. Logic Machine Architecture: Inference mechanisms. In D. Loveland, editor, *Proceedings of the 6th Conference on Automated Deduction, Lecture Notes in Computer Science, Vol. 138*, pages 85–108, New York, 1982. Springer-Verlag.
8. E. Lusk, W. McCune, and R. Overbeek. Logic Machine Architecture: Kernel functions. In D. Loveland, editor, *Proceedings of the 6th Conference on Automated Deduction, Lecture Notes in Computer Science, Vol. 138*, pages 70–84, New York, 1982. Springer-Verlag.
9. E. Lusk, W. McCune, and J. Slaney. Roo—a parallel theorem prover. Tech. Memo MCS-TM-149, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1991.
10. E. Lusk and R. Overbeek. Experiments with resolution-based theorem-proving algorithms. *Computers and Mathematics with Application*, 8(3):141–152, 1982.
11. E. Lusk and R. Overbeek. The automated reasoning system ITP. Tech. Report ANL-84/27, Argonne National Laboratory, Argonne, IL, April 1984.
12. E. Lusk and R. McFadden. Using automated reasoning tools: A study of the semigroup f2b2. *Semigroup Forum*, 36(1):75–88, 1987.
13. J. McCharen, R. Overbeek, and L. Wos. Problems and experiments for and with automated theorem-proving programs. *IEEE Transactions on Computers*, C-25(8):773–782, August 1976.
14. W. McCune. Otter 2.0 Users Guide. Tech. Report ANL-90/9, Argonne National Laboratory, Argonne, IL, March 1990.
15. W. McCune. Automated discovery of new axiomatizations of the left group and right group calculi. Preprint MCS-P220-0391, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1991. To appear in *J. Automated Reasoning*.
16. W. McCune. Experiments with discrimination tree indexing and path indexing for term retrieval. Preprint MCS-P191-1190, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1991. Invited paper, to appear in *J. of Automated Reasoning*.
17. W. McCune. Single axioms for the left group and right group calculi. Preprint MCS-P219-0391, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1991.
18. W. McCune. What's New in Otter 2.2. Tech. Memo ANL/MCS-TM-153, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, July 1991.
19. W. McCune and L. Wos. The absence and the presence of fixed point combinators. *Theoretical Computer Science*, 87:221–228, 1991.
20. R. Overbeek. A new class of automated theorem-proving algorithms. *Journal of the ACM*, 21:191–200, 1974.

21. R. Overbeek. An implementation of hyper-resolution. *Computers and Mathematics with Applications*, 1:201–214, 1975.
22. 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. Edinburg U. Press, 1969.
23. J. Slaney. FINDER, finite domain enumerator: Version 1.0 notes and guide. Tech. Report TR-ARP-10/91, Automated Reasoning Project, Australian National University, Canberra, Australia, 1991.
24. J. Slaney and E. Lusk. Parallelizing the closure computation in automated deduction. In M. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 449*, pages 28–39, New York, July 1990. Springer-Verlag.
25. S. Winker. Generation and verification of finite models and counterexamples using an automated theorem prover answering two open questions. *J. ACM*, 29:273–284, 1982.
26. S. Winker, L. Wos, and E. Lusk. Semigroups, antiautomorphisms, and involutions: A computer solution to an open problem, I. *Math. of Comp.*, 37:533–545, 1981.
27. L. Wos, D. Carson, and G. Robinson. The unit preference strategy in theorem proving. In *Proceedings of the Fall Joint Computer Conferenc*, pages 615–62, New York, 1964. Thompson Book Compan.
28. L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications*. McGraw–Hill, second edition, 1992.
29. L. Wos, G. Robinson, and D. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, 1965.
30. L. Wos, S. Winker, and E. Lusk. An automated reasoning system. In *Proceedings of the AFIPS National Computer Conference*, pages 697–702, 1981.
31. L. Wos, S. Winker, W. McCune, R. Overbeek, E. Lusk, R. Stevens, and R. Butler. Automated reasoning contributes to mathematics and logic. In M. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 449*, pages 485–499, New York, July 1990. Springer-Verlag.
32. L. Wos, S. Winker, B. Smith, R. Veroff, and L. Henschen. A new use of an automated reasoning assistant: Open questions in equivalential calculus and the study of infinite domains. *Artificial Intelligence*, 22:303–356, 1984.