Communications of the ACM (to appear 1997)
Also Preprint ANL/MCS-P655-0397

Programs That Offer Fast, Flawless, Logical Reasoning*

Larry Wos

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439-4801
e-mail: wos@mcs.anl.gov

Sherlock Holmes and Mr. Spock of *Star Trek* could reason logically and flawlessly, always. Some people you know have that ability, sometimes. Unfortunately, without perfect reasoning, diverse problems arise:

- Bugs in computer programs. If a sort program places Sun before Intel, more than disappointment is experienced.
- Flaws in chip design. One Pentium chip became famous because of a flaw.
- Errors in mathematical proofs. Papers having a title of the form ‘‘On an Error by MacLane’’ are well remembered, but not with pleasure (at least by MacLane).

How can the likelihood of the various cited disasters be reduced? One answer is *automated reasoning*. (For further information on automated reasoning at Argonne National Laboratory, see http://www.mcs.anl.gov/home/mccune/ar/, which will give you all of the needed pointers for obtaining a versatile automated reasoning program, for new results, for various neat proofs, and for puzzles.)

1. Automated Reasoning and the Eureka Experience

The focus of *automated reasoning* is the design and implementation of computer programs that flawlessly apply logical reasoning to reach the objective, whether the area of interest is circuit design, program verification, theorem proving, puzzle solving, or more. Featured in this article is a versatile automated reasoning program—McCune’s OTTER [3]—that you can use for attacking problems in each of these areas and, so important, for sharply reducing the likelihood of disaster.

Fortunately, you can easily obtain your own personal copy of OTTER. For example, a relatively recent version is included on diskette in a book that gives you an introduction to automated reasoning, details various applications, and assumes no background in computing or in logic [9]. Alternatively, you may wish to read about the *process* by which significant discoveries are made using an automated reasoning program; a new book [11] presents just such an approach and includes a very recent version of OTTER on diskette. You can also obtain a copy of the OTTER source from the Internet; see the Web page cited earlier. Finally, for those of you who wish a small taste of using OTTER immediately, you can try your hand using the Son of BirdBrain (hot-linked on the cited Web page).

Before you decide whether access to an automated reasoning program will sharply increase your likelihood of success, you might naturally wonder how its reasoning possibly differs from yours and from that of others you know. You might also be curious about how its strategic attack possibly differs from that a person typically takes. I believe—although some of my colleagues do not entirely share my views—that precious little is known about how people consciously reason, especially when attacking a deep problem.

*This work was 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.

Of course, the eureka experience is well documented: Poincaŕe, after failing to solve a problem on which he had worked for months, suddenly knew its solution. I have had a similar experience, waking at three in the morning, instantly aware of what was needed to solve a problem in abstract algebra. The subconscious mind is a marvel! A computer program (of the type under discussion), of course, has no subconscious mind, nor intuition, nor, for that matter, experience on which to draw.

Also in sharp contrast to people is an automated reasoning program's use of specific and well-defined inference rules for drawing conclusions. People, I have discovered, seldom use specific rules for drawing conclusions, even when the subject is mathematics. In addition, people often wish to reason from pairs of statements, whereas a reasoning program such as OTTER offers various means to draw conclusions from statements taken three at a time, or more.

Compared with reasoning, even less is known about how a person selects appropriate data from a huge amount of information—specifically, which techniques (strategies) are used to restrict the reasoning and which are used to direct it. Indeed, how does the disciplined mind escape drowning in useless conclusions, and how does such a mind avoid getting lost? If you question researchers, you will find that *explicit* strategy is seldom if ever used, in contrast to playing chess or playing poker. If a reasoning program is to serve well as an assistant, however, the use of explicit strategies (and I use the plural deliberately) is an absolute necessity.

Strategies for restricting and strategies for directing a program's reasoning have intrigued me for decades. The sidebar on Strategies discusses some of the more powerful strategies that my colleagues and I have formulated. Through such strategies, a reasoning program can avoid drowning in new conclusions and can avoid getting lost. Also, by using such strategies, OTTER can attack a wide variety of problems from diverse areas (as illustrated in Section 3). (You might find it interesting to note that I believe one of my most important contributions was the introduction, in 1963, of the use of strategy by automated reasoning programs. I am almost equally proud of having introduced the term ''automated reasoning'' in 1980, for it captures far better than the traditional term ''automated theorem proving'' the remarkable diversity of these computer programs.) And OTTER has been remarkably successful in such attacks, not the least of which is the answering of questions that had resisted mathematicians for more than sixty years; see the sidebar on Robbins algebra. Section 4 details some of the outstanding achievements of various automated reasoning programs. (For evidence of how well automated reasoning is doing, see the Progress sidebar.)

But, to answer a pressing question, OTTER will never replace people (see Section 6). Rather, OTTER is intended to *complement* the approach taken by a person, not to emulate it.

2. The Program OTTER

Through the sponsorship of the Department of Energy and its predecessors, researchers at Argonne National Laboratory have been designing and implementing automated reasoning programs for more than thirty years. William McCune worked for several years on earlier reasoning programs at Argonne before producing—in approximately four months—the first version of OTTER in late 1987. The 1997 version of OTTER consists of more than 24,000 lines of C and can be used on a workstation, a personal computer, and a Macintosh.

OTTER (Organized Theorem-proving Techniques for Effective Research) derives its name in part from its original intended use and in part from the delight of using it—the delight one can experience when watching the animal of the same name. The program was designed and implemented for a number of reasons, including giving a person access to a team member offering (1) a wide variety of automated means for drawing conclusions that follow inevitably from the assumptions that are supplied and (2) diverse powerful strategies for controlling the program's reasoning.

3. Five Example Problems Solved with Logical Reasoning

The following five example problems, each involving logical reasoning, provide a tiny taste of what a reasoning program (such as OTTER) can do for you. They also suggest charming choices you can make for (1) the inference rules the program uses to draw conclusions, (2) the strategies for restricting and directing its reasoning, and (3) features such as that for rewriting information into a canonical form. The examples also hint at the unpleasant feature of using an automated reasoning program: the input language, the need to be explicit about trivial information, and the dangers of implicit information and irrelevant information, both of which affect the program's chances of succeeding. Perhaps as compensation for your labor, the program protects you in various ways. For example, a person can easily think of the brother of Bob's aunt as Bob's uncle—which is not necessarily the case—where an automated reasoning program would not make such a possibly incorrect translation.

The five problems may enable you to find new applications for automated reasoning, professional and recreational. You will discover that a program such as OTTER does not imitate the approach a person would ordinarily take, which is why the term *artificial intelligence*—especially in view of AI's original intent in the 1960s—is not appropriate.

Example 1: Databases. The first example is vaguely reminiscent of a simple database problem. You have some knowledge and, by drawing some conclusions, hope to determine whether an assertion is true. Here is the problem.

You are told that Joy, who is married and not male, is the managing editor of the *Journal of Automated Reasoning*, and you are told that the only person who left the parking lot before 5:00 in the evening was not a female. You are then told that Joy left the parking lot before 5:00 in the evening. Finally, you are asked to prove that this last assertion is false. Elementary reasoning suffices. Indeed, you first conclude that Joy is female, and then you use that conclusion to deduce that Joy did not leave the parking lot before 5:00 in the evening. Therefore, a contradiction is obtained—you have completed a proof by contradiction and, in particular, have solved the problem by showing that the assertion is false. If you look closely, you see that you ignored two items of information (that Joy is a managing editor and that Joy is married) and at the same time used one item of information that was not given explicitly (that people are female or male).

What would happen if the same example were given to an automated reasoning program such as OTTER? So that you can briefly experience what can be less pleasant, let us give some of the possibly pertinent information in clause form [9] (one of the language representations OTTER recognizes), where logical **not** is denoted by ''-''.

Joy is not male: -MALE(Joy).
Joy is managing editor of JAR: EDITOR(Joy).
and the like.

Without more, the program would fail to draw any conclusions. Why? First, it does not ''know'' that each person is female or male; it must be told such, for example, with the following clause, where logical **or** is denoted by '' | ''.

FEMALE(x) | MALE(x).

(Your conclusion that *x* is a variable is indeed correct, implicitly ranging over all people.) For that matter, the program does not even ''understand'' the concept of female or of male. Indeed, without the item of information that every person is female or male (which you used implicitly), the program would be helpless.

Even with that information, the automated reasoning program would require additional help. You would have to choose some specific inference rule, a rule of reasoning with which to draw conclusions. OTTER (and many other such programs) could use a rule such as *UR-resolution*, which works in the following way. The program takes the clause that asserts that Joy is not male and *unifies* it with one of the literals in the clause asserting that each person is female or male. In particular, all occurrences of the variable *x* in the female-or-male clause are replaced with the term Joy. Then, the -MALE(Joy) literal cancels the (just-obtained) MALE(Joy) literal (because the two literals are identical but opposite in sign), and the useful conclusion is reached and stored in the following clause.

FEMALE(Joy).

(For those who are curious about other ways OTTER reasons, and especially about how its reasoning sharply differs from that expected of a person, I discuss these topics in Section 5.)

Can OTTER succeed now? Perhaps, but unless you add the use of *strategy*, the odds are high that the program would get lost, at least for problems of greater depth. One of the key strategies used by OTTER to restrict its reasoning is the *set of support strategy* [9]. You tell the program which statements that present the problem are (in effect) general information. This strategy restricts the program from applying an inference rule to a set of items all of which are among the general-information statements. The program is therefore prevented from exploring the underlying theory from which the problem is taken, which almost always sharply increases its efficiency.

Strategy—explicit strategy—is vital to OTTER and vital to any reasoning program attacking deep questions; for a discussion of various strategies used by OTTER, see Section 5, [9], and the Strategy sidebar. In contrast, seldom if ever does a person use explicit strategy. In no way does this fact mean that people are not brilliant at problem solving; indeed, they are. What it does mean is that OTTER is not designed to emulate a person's reasoning. Perhaps this fact explains why OTTER makes such a valuable team member—by *complementing* the approach a person might take.

Example 2: Circuit Design. In language, the **and** of two statements is true if and only if each is true; otherwise, the **and** is false. In circuit design, (in effect) the same holds: An AND gate takes two inputs and outputs 1 if and only if each input is 1; otherwise, the AND gate outputs 0. In the second example, you are asked to design a number of different circuits, but you are constrained to avoiding the use of AND gates because their cost has risen sharply. Rather than taking a complicated approach, you decide to use OR gates and NOT gates. You rely on one single simplification or canonicalization rule: The **and** of A and B = **not**(**not**(A) **or** **not**(B)). OTTER can apply such a rule, as well as other such rules, without error, at the rate of 5000 per CPU second. (In contrast to most concepts, OTTER ''understands'' equality, as seen in this and the next example.)

Perhaps you are puzzled; after all, applying such a simplification rule is hardly difficult and not particularly prone to error. But now imagine being presented with 17,000 simplification rules (called *demodulators* [9] in automated reasoning) and an expression that was rewritten into its final form only after 1766 demodulators had been applied. For an additional measure of the chore that simplification and canonicalization can present, I note that in one of the greatest successes with a reasoning program—the answering of an open question concerning Robbins algebra [5]—more than 500,000 CPU-seconds were spent on demodulation, approximately 75% of the total run time; also see [2], and, especially for a statement of the Robbins problem that had resisted mathematicians for sixty years, see the Robbins algebra sidebar. So you see how what might seem innocent on the surface, namely, simplification, can in fact be a nightmare if attacked by hand rather than by computer.

Example 3: Arithmetic and Mathematics. You are given the following two well-known properties of arithmetic (and of group theory)

$$x + -x = 0$$
$$y + (-y + z) = z$$

and are asked to deduce a third well-known, useful, and powerful property:

$$y = -(-y).$$

OTTER offers an inference rule, *paramodulation* [9], that does almost all of the work needed to draw this conclusion immediately. (Paramodulation, illustrated in Section 5, is more general than equality substitution.) With paramodulation, the program deduces

$$y + 0 = -(-y),$$

which, assuming that (as is usually the case) the following simplification rule is applied, is rewritten to the desired result.

$$w + 0 = w$$

With paramodulation, the automated reasoning program finds conclusions of (in a sense) maximum generality. A person, on the other hand, typically seeks a conclusion that mirrors earlier experience and intuition, and that conclusion may lack the needed generality in turn preventing one from finding the desired information, for example, completing the sought-after proof. Unification indeed saves the day, by finding the most general replacement for variables that, when applied to the two expressions of concern, yields two identical expressions. When the needed most general replacement for variables is overlooked—which would be easy if done by hand—chaos can result. For but one example, a specific conclusion might be drawn rather than a needed general conclusion. An automated reasoning program is not subject to such misfortune.

Example 4: A Hard Puzzle. Like Lewis Carroll and Raymond Smullyan, many people are fascinated by puzzles. You might find the following billiard ball puzzle stimulating and useful, by offering it to colleagues, friends, loved ones, and strangers.

> There are twelve billiard balls, eleven of which are identical in weight. The remaining ball—the odd one—has a different weight. You are not told whether it is heavier or lighter. You have a balance scale for weighing the balls. Can you find which ball is the odd ball in three weighings, and can you also find out whether it is lighter or heavier than the others?

Of course, the heart of the problem concerns the constraint of the three weighings. (I shall give you a hint about solving the puzzle just after mentioning the number of solutions that were found; stop reading there if you wish to solve the puzzle by yourself.) Without such a constraint, you could choose two billiard balls and weigh one against the other with the scale. If they balanced, you would mark each as standard, as one of the eleven with identical weight. If they did not balance, which is even better, you would learn something crucial, putting them aside with a marking of undecided. You would then know that the remaining ten were of identical weight, and it would be a trivial matter to ascertain which was the odd ball and whether it was lighter or heavier. But life and the billiard ball puzzle are not that easy, and you may be surprised to learn that OTTER found more than 40 nontrivially distinct solutions [9] for the billiard ball puzzle—while various people were still working on 1. *Hint*: Begin by weighing four balls against four balls.

Example 5: Circuit Design and the Two-Inverter Problem. The following problem was included on various exams given to Ph.D. candidates. The challenge to the circuit designer is twofold: (1) to meet all the restrictions and (2) to ensure that the circuit behaves as desired. A person might follow one unrewarding line of reasoning after another, getting lost in the myriad of paths that lead nowhere. An automated reasoning program such as OTTER succeeds easily [9]. Can you solve the problem?

> Using as many AND and OR gates as you like, but using only two NOT gates, can you design a circuit according to the following specification? There are three inputs, i1, i2, and i3, and three outputs, o1, o2, and o3. The outputs are related to the inputs in the following simple way:
>
> o1 = not(i1)   o2 = not(i2)   o3 = not(i3).
>
> Remember, you can use only two NOT gates!

(Once you have tried your hand, you might enjoy glancing at the pictorial result translated from OTTER's success, given as a sidebar.)

4. A Grand Return on the Investment

You now see how complex reasoning often is, for a person and for a computer program. You also have some feeling for the value of using an automated reasoning program, especially for attacking a hard problem. How comforting it is to know that none of the reasoning that occurs contains an error, that the program does not tire, and that it explores paths of reasoning that would simply be too exhausting for a person! Further, some of the problems whose solution is now known would still be in the dark were it not for the role played by a program such as OTTER.

Of course, the five example problems present no problem for OTTER. (Otherwise, ''better'' examples would have been chosen.) But OTTER, as well as other automated reasoning programs, has answered numerous and far more challenging questions, some of which had defied experts for many years [4, 5, 11].

Nevertheless, a miracle is not at hand. OTTER's principal successes have been mainly in mathematics and in logic, for example, finding answers to open questions concerning combinatory logic, lattice theory, and algebraic geometry. The reason rests in large part with the fact that these areas offer a concise description of the underlying theory. Access to such a concise description (a set of axioms, for example) is what is so often missing when one contemplates a new application of automated reasoning.

Consider what would happen if you decided to try to determine whether certain properties held in field theory (a branch of physics), relying on the assistance of an automated reasoning assistant. First, you would have to formulate the precise concepts that are needed. Then, you would have to find a means to map the salient concepts into the language acceptable to whatever automated reasoning program you intended to use. For a second example, if you were studying astronomy, the concepts of planet, star, galaxy, universe, and the like might be tough nuts to crack.

For a problem taken directly from industry (a problem on which I and a colleague worked), imagine that an assembly line is producing cars—some painted blue, some green, some with air conditioning, some with power windows, and the like—and you know that the efficiency is severely reduced if the cars are sequenced randomly. For example, flushing the painter to accept alternately blue, green, blue, green is not ordinarily efficient; similarly, since installing air conditioning can be time consuming, a sequence of cars all requiring this feature might not be the best. Use of an automated reasoning program might indeed aid you in finding a good car-sequencing algorithm, but you would need to convey the crucial information; for a discussion of the approach taken to this job scheduling problem, see [6]. (The car-sequencing problem is representative of a class of optimization problems typically studied in the field of operations research.)

Both tasks—that of axiomatization and that of representation in the program's language—can be trying, even formidable.

Nevertheless, once all is in place, the return on such an investment can be astounding.

Automated reasoning programs (such as OTTER and the program of Boyer and Moore [1]) have, for example, been used in research [4, 11] and in applications [10] to

1. produce a fully automated proof of the correctness of a division circuit that implements the floating-point IEEE standard;
2. produce correctness proofs of security systems;
3. verify commercial-size adder and multiplier circuits;
4. achieve new mathematics, for example, new results in quasigroups and in non-Euclidean geometry;
5. settle open questions, such as finding minimal axiom sets; and
6. confirm conjectures, for example, that of Higmann and the paradox of Gerard.

Reasoning programs have also been used by educators, for example, in undergraduate logic courses and in upper-level courses in interactive theorem proving. Finally, for a recreational example, automated reasoning programs have turned out to be excellent assistants in puzzle solving. (For both easy and hard puzzles, including variations on the well-known checkerboard and dominoes puzzle, see [9] and the earlier-cited Web address.)

5. Computer-oriented Reasoning versus Person-oriented Reasoning

Does automated reasoning mirror the classical approach to artificial intelligence? Does OTTER, for example, reason somewhat the way a person does? As the following shows, the answer to both questions is a resounding ''No!'', if one adheres to the strictest definitions.

Unification, as you probably surmised, is at the heart of automated reasoning. You have probably already conjectured, correctly, that unifying expressions is *not* typical of what a person does. Reasoning focusing on equality offers another fine example of what OTTER does well and, at the same time, what a person sometimes finds arduous.

For evidence, with the following three equations, let us return to Example 3 and the use of paramodulation. Specifically, by a well-chosen replacement of terms for variables in each of the first two equations, followed by an equals-for-equals substitution *from* the lefthand argument of the modified first equation *into* a proper subterm of the lefthand argument of the modified second equation, you obtain the third equation.

$x + -x = 0$
$y + (-y + z) = z$

$y + 0 = -(-y)$

Indeed, if all occurrences of x in the first equation are replaced by $-y$ and all occurrences of z in the second by $-(-y)$, then you are ready to apply the final equality-substitution step to obtain the third equation. Paramodulation combines the variable-replacement step and the equality-substitution step; it finds the most general replacement that yields identical subexpressions (through unification); and it produces the conclusion without an intermediate so-to-speak subconclusion—and in one step. I suspect that you do not need a more complicated illustration of paramodulation to be persuaded that a person might not always find the appropriate variable replacement and might, therefore, miss drawing a key conclusion.

On the other hand, if you wish an example of what a person does well and OTTER does not even attempt, you need only turn to what is called *instantiation*. Although perhaps not with the formal name of instantiation, you at some point have witnessed its use by some lecturer or author when a formula or equation is presented with, say, variables x, y, and z, and a possibly mysterious conclusion is presented next by replacing (instantiating) z by $uv+vv$ with u and v variables, y by $a(b+c)$ for the constants a, b, and c, and x is replaced by the familiar and well-known number 2. The mysteriousness (if present) results from the lack of explanation for this particular instantiation (choice of instances) from among the infinite possibilities that could have been chosen. For that very reason, OTTER (and similar programs) do not offer as an inference rule instantiation; I know of no effective strategy for wisely choosing from among the infinite set of instances that are usually available.

In that an automated reasoning program does not rely on instantiation to draw conclusions, you might wonder what inference rules it does use, in addition to UR-resolution (used in Example 1) and paramodulation (used in Example 3). You are correct if you conjectured that UR-resolution requires the conclusion to be nonempty and free of logical **or**. OTTER also offers an inference rule, *hyperresolution*, that yields conclusions free of logical **not**. Both inference rules are frequently applied to a set of statements containing more than two items, which is not the way a person usually reasons. The program also offers you the use of *binary resolution*, a rule that always focuses on pairs of statements without any constraint on the conclusion that is drawn, which is reminiscent of a person's reasoning.

Even with such diverse rules for drawing conclusions, all would still be lost were it not for strategy, for the program's reasoning must be restricted and must also be directed; see, for example, Section 3 and the Strategy sidebar.

Further, to be most effective, some means is needed for the program to benefit from what you know and what you do well. Indeed, ideally, you might wish to combine a person's experience, intuition, and reasoning with the reasoning power of a program such as OTTER. In a sense, OTTER lets you do just that—through the use of strategy.

- You have some experience that suggests one subexpression merits preference over another, such as that involving sum over that involving product. You can use the *weighting strategy* [9] to guide the program accordingly. For another example, you might wish to design circuits with minimal, but perhaps some, use of OR gates. With weighting, you can instruct the program to follow your preference.
- Your intuition—or perhaps a wild guess—tells you that the steps of a proof you have in hand could be profitably used to find a proof only distantly related. Two strategies can be used for your intent, that of *hints* [7], and that of *resonance* [11].
- You conjecture that certain statements in the problem description, as parents, merit immediate visiting and even immediate revisiting when conclusions are to be drawn. You can have your way by employing the *hot list strategy* [12].

Using these and other strategies, and guided by your own reasoning, experience, and intuition, OTTER can assist you in solving a wide variety of problems.

6. A Replacement for People?

Especially in this era of downsizing, you might still wonder whether OTTER will replace people. Rest assured: OTTER is *not* a replacement for a person. As noted earlier, it cannot draw on intuition or experience, nor can it formulate new concepts. Such a program does not learn, nor is it self-analytical in the following sense. When you choose an approach to attacking a problem, you often monitor (in some fashion) your progress and, based on that monitoring, modify the approach as you go along. Some day, but not now, an automated reasoning program will have that self-analytical capacity (see [8]). An automated reasoning program lacks common sense, cannot judge the accuracy of your statements, and cannot test for omissions in the problem description: It trusts you.

In many other ways, however, a program such as OTTER provides excellent incentives for relying on it instead of relying exclusively on a person. First, OTTER is remarkably powerful, able to draw thousands of conclusions per second. And it does not tire—it continues to draw conclusions at virtually the same rate even after millions of conclusions have been drawn.

Second, OTTER does not leave huge gaps in its reasoning, forcing you to guess the ‘‘not-always-obvious’’ omitted steps. It explicitly presents its work in an output file, including the history of each conclusion it draws. If your initial attack on a problem fails, you can examine what OTTER has done and modify your approach or correct the input. For but one example, you might find that the problem description is inconsistent.

Third, the program can have multiple personalities (without the attendant psychological problems). In particular, if you have access to a number of computers, you can apply a multifaceted attack, having the program try different approaches on different computers. Such a multifaceted attack might enable you to explore radically different approaches that might ordinarily be rejected because of the effort involved.

OTTER was not designed to replace people, or even to imitate people. Will it, or any other automated reasoning program, replace scientists and engineers and the like? *Never*. Unquestionably, automated reasoning has made great strides in the past few years; see the graph. But at most, I envision that automated reasoning programs will enable people to devote their energy and time to bigger pictures, having reasoning progras attack smaller problems quickly and flawlessly and, occasionally, without any assistance answer a deep question.

Tempting?

If any of the preceding tempts you to experiment or simply dabble with the use of an automated reasoning program, I recommend that you obtain your own copy of OTTER, whether for a workstation, personal computer, or Macintosh. Two paths to this end are via the earlier cited books, [9] and [11]. Finally, you might also wish to read the OTTER Web page: http://www.mcs.anl.gov/home/mccune/ar/otter/.

I conjecture that once you have used OTTER, you will more than like it. OTTER makes a marvelous automated reasoning assistant.

References

1. Boyer, R. S., and Moore, J S. *A Computational Logic Handbook.* New York: Academic Press, 1988 (also Web information http://www.cli.com/software/nqthm/obtaining.html).

2. Kolata, G., ‘‘With Major Math Proof, Brute Computers Show Flash of Reasoning Power’’, *New York Times*, Science Times, 10 December 1996, B5ff.

3. McCune, W., *OTTER 3.0 Reference Manual and Guide,* Technical Report ANL-94/6, Argonne National Laboratory, Argonne, Illinois, 1994.

4. McCune, W., and Padmanabhan, R., *Automated Deduction in Equational Logic and Cubic Curves,* Lecture Notes in Computer Science, Vol. 1095, Springer-Verlag: Heidelberg, 1996. See http://www.mcs.anl.gov/home/mccune/ar/monograph/ for additional information.

5. McCune, W., ‘‘Solution of the Robbins Problem’’, *J. Automated Reasoning,* accepted for publication, 1997.

6. Parrello, B., Kabat, W., and Wos, L., ‘‘Job-Shop Scheduling Using Automated Reasoning: A Case Study of the Car-Sequencing Problem’’, *J. Automated Reasoning* **2**, no. 1, 1-42 (1986).

7. Veroff, R., ‘‘Using Hints to Increase the Effectiveness of an Automated Reasoning Program: Case Studies’’, *J. Automated Reasoning* **16**, no. 3, 223-239 (1996)

8. Wos, L., *Automated Reasoning: 33 Basic Research Problems,* Prentice-Hall, Englewood Cliffs, New Jersey (1987).

9. Wos, L., Overbeek, R., Lusk, E., and Boyle, J., *Automated Reasoning: Introduction and Applications,* 2nd ed., McGraw-Hill, New York (1992).

10. Wos, L., and Pieper, G. W., eds. Special issue on automated reasoning. *Computers and Mathematics with Applications* **29**, no. 2, 133-178 (February 1995).

11. Wos, L., *The Automation of Reasoning: An Experimenter’s Notebook with OTTER Tutorial,* Academic Press: New York, 1996.

12. Wos, L., ‘‘OTTER and the Moufang Identity Problem’’, *J. Automated Reasoning* **17,** no. 2, 259-289 (1996).

Larry Wos is a senior mathematician in the Mathematics and Computer Science Division at Argonne National Laboratory. He received his M.S. in Mathematics from the University of Chicago in 1954 and his Ph.D. in mathematics from the University of Illinois-Urbana in 1957. Dr. Wos is president of the Association for Automated Reasoning and a member of the editorial board of the *Journal of Automated Reasoning* (which he founded in 1983). He is author of four books and approximately 70 refereed articles. He was the first to receive the Herbrand Award in Automated Deduction, presented in 1992.

Examples of Strategy

1. *Unit Preference* (Wos): directs a program's reasoning by preferring nonempty statements free of logical **or** as parents for drawing conclusions
2. *Set of Support* (Wos): restricts a program's reasoning by preventing it from drawing conclusions all of whose parents are among statements the user (in effect) designates as general information
3. *Weighting* (Overbeek): directs a program's reasoning by giving priority to terms the user selects, and to restrict a program's reasoning by discarding new conclusions whose complexity exceeds a user chosen upper bound
4. *Ratio* (McCune): directs a program's reasoning by focusing on a combination of least-complex information and first-come first-serve information, where the combination is determined by the user
5. *Resonance* (Wos): directs a program's reasoning based on the inclusion of formulas and equations that the user conjectures merit high priority
6. *Hints* (Veroff): directs a program's reasoning based on the inclusion of formulas and equations that the user conjectures merit proving
7. *Hot List* (Wos): rearranges conclusion drawing by visiting and revisiting as parents statements that the user conjectures to be especially significant
8. *From Variable* (Wos): restricts a program's reasoning in the context of equality-oriented reasoning (paramodulation) by preventing it from substituting from a variable
9. *Into Variable* (Wos): restricts a program's reasoning in the context of equality-oriented reasoning (paramodulation) by preventing it from substituting into a variable

The Robbins Problem

As background for the Robbins problem, note that a Boolean algebra is the mathematical abstraction of the study of logical **and**, **or**, and **not**. The Robbins problem asks if the following three axioms completely characterize Boolean algebra, where + can be thought of as logical **or**, and the function *n* can be thought of as logical **not**.

```
x + y = y + x.                 % Commutativity of +
(x + y) + z = x + (y + z).     % Associativity of +
n(n(n(x) + y) + n(x + y)) = y. % Robbins axiom
```

From a set-theoretic perspective, + can be thought of as union and the function *n* as complement.

Note: For figures (Progress in Automated Reasoning 1960-1966 and circuit diagram), contact Judy Beumer at beumer@mcs.anl.gov and request the figures from ANL/MCS-P655-0397.