Larry Wos

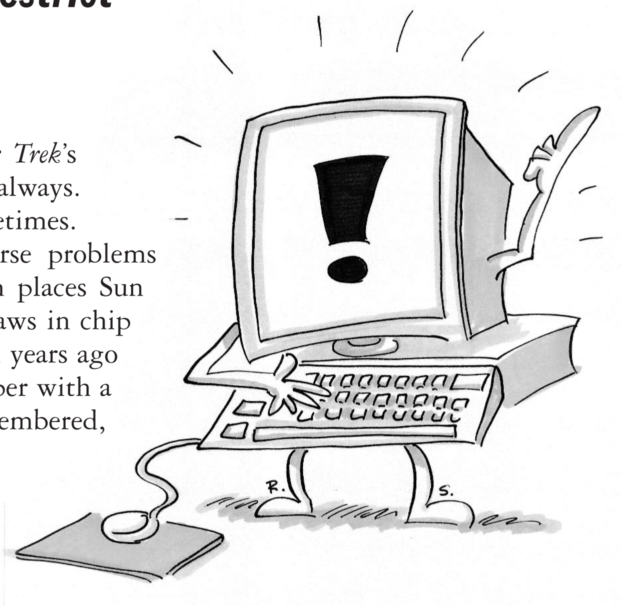
PROGRAMS THAT OFFER FAST, FLAWLESS, LOGICAL REASONING

Confused by too many reasonable conclusions? Sort them out through automated reasoning using special strategies that logically restrict and direct the search for the answer.

The English detective Sherlock Holmes and *Star Trek*'s Mr. Spock could reason logically and flawlessly, always. Some people you know have that ability, sometimes. Unfortunately, without perfect reasoning, diverse problems arise, like bugs in computer programs (if a sort program places Sun before Intel, more than disappointment is experienced); flaws in chip design (one type of Pentium chip became infamous several years ago because of a flaw); and errors in mathematical proofs (a paper with a title of the form "On an Error by MacLane" is easily remembered, but not with pleasure—at least by MacLane).

How can the likelihood of such disasters be reduced? One answer is *automated reasoning*.

The focus of automated reasoning is the design and implementation of computer programs that flawlessly apply logical reasoning to reach the objective,



RICK STROMOSKI

whether the area of interest is circuit design, program verification, theorem proving, puzzle solving, or something else. Featured in this article is the versatile program OTTER—developed by researcher William McCune [3]—that you can use to attack problems in each of these areas, especially to sharply reduce the likelihood of disaster. (For lots more on automated reasoning at Argonne National Laboratory in Illinois, including pointers to obtaining an automated reasoning program called OTTER, as well as new results, neat proofs, and puzzles, see www.mcs.anl.gov/ home/mccune/ar/).

Fortunately, you can easily obtain your own personal copy of OTTER, which is included on a diskette in [9]. That book also shows you the *process* by which significant discoveries are made using an automated reasoning program. You can obtain a copy of the OTTER source code through the Argonne Web page. Finally, for those who want a taste of using OTTER immediately, try the Son of BirdBrain hot-link on the same Web page.

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

The eureka experience is well documented. Jules Henri Poincaré, the French mathematician, 1854–1912, 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 3 A.M., 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) has no subconscious mind, nor intuition, nor, for that matter, experience on which to draw.

Also in sharp contrast to the way people reason 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 like OTTER offers various means of drawing conclusions from statements taken three or more at a time.

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 an *explicit* strategy is seldom used, in contrast to the case in playing chess or poker. If a reasoning program is to serve well as an assistant,

Unit Preference (Wos): Directs a program's reasoning by preferring nonempty statements free of logical **or** as parents for drawing conclusions.

Set of Support (Wos): Restricts a program's reasoning by preventing it from drawing conclusions whose parents are all among statements the user (in effect) designates as general information.

Weighting (Overbeek): Directs a program's reasoning by giving priority to terms the user selects and restricting a program's reasoning by discarding new conclusions whose complexity exceeds a user-chosen upper bound.

Ratio (McCune): Directs a program's reasoning by focusing on a combination of least-complex information and first-come, first-served information, where the combination is determined by the user.

Resonance (Wos): Directs a program's reasoning based on the inclusion of formulas and equations the user conjectures merit high priority.

Hints (Veroff): Directs a program's reasoning based on the inclusion of formulas and equations the user conjectures merit proving.

Hot List (Wos): Rearranges conclusion-drawing by visiting and revisiting as parents statements the user conjectures to be especially significant.

From Variable (Wos): Restricts a program's reasoning in the context of equality-oriented reasoning (paramodulation) by preventing it from substituting from a variable.

Into Variable (Wos): Restricts a program's reasoning in the context of equality-orientied reasoning (paramodulation) by preventing it from substituting into a variable.

Figure 1. Examples of strategy

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.

Figure 2. The Robbins problem

however, the use of explicit strategies (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. Figure 1 lists some of the more powerful ones my colleagues and I have formulated. Through such strategies, a reasoning program can avoid drowning in new conclusions and avoid getting lost. Also, by using such strategies, OTTER can attack a wide variety of problems from diverse areas. (You might find it interesting to note that one of my most important contributions to the field 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; it captures far better than the traditional term "automated theorem proving" the remarkable diversity of these computer programs.) OTTER has been remarkably successful in such attacks, not the least of which is its answering of various questions that had resisted mathematicians for decades. (For such a success with a related program, see Figure 2. For evidence of how well automated reasoning is doing, see Figure 3.)

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

The Program OTTER

Through the sponsorship of the U.S. Department of Energy and its predecessors, researchers at Argonne have been designing and implementing automated reasoning programs for more than 30 years. William McCune worked for several years on earlier reasoning programs before producing—in approximately four months—the first version of OTTER in 1987. The 1998 version of OTTER consists of more than 24,000 lines of C and can be used on powerful workstations, PCs, and Macintoshes.

OTTER (Organized Theorem-proving Techniques for Effective Research) derives its name in part from

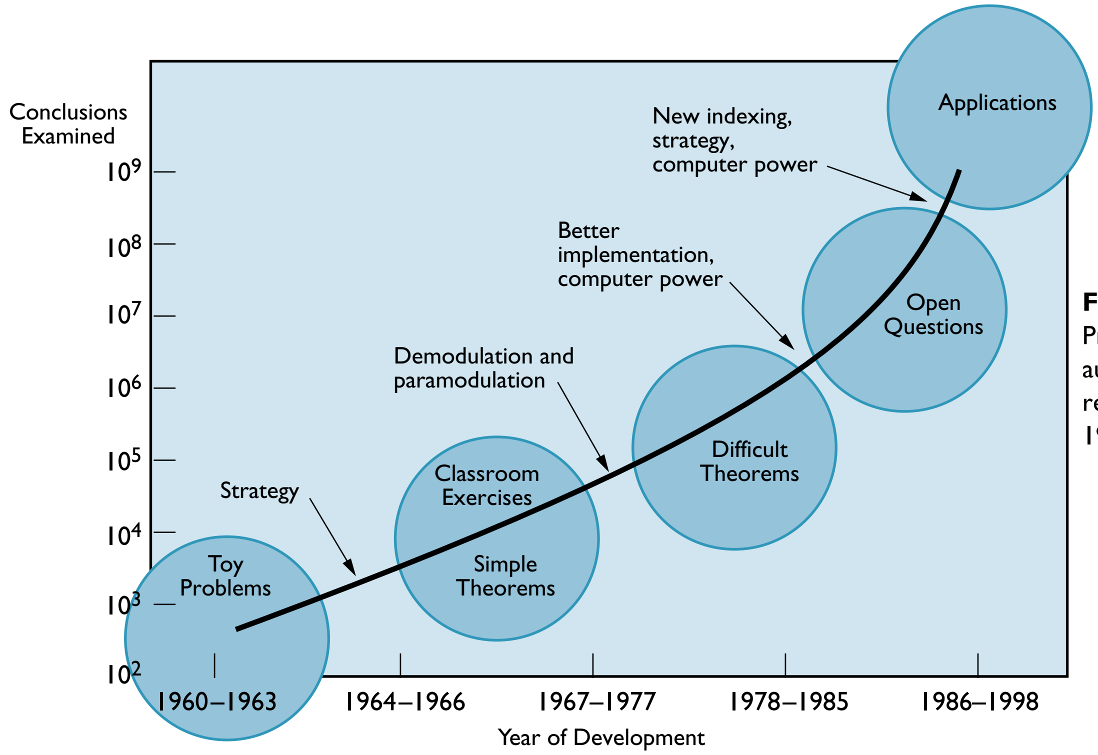


Figure 3. Progress in automated reasoning 1960–1998

HOW DOES THE DISCIPLINED MIND ESCAPE DROWNING IN USELESS CONCLUSIONS, AND HOW DOES SUCH A MIND AVOID GETTING LOST?

its original intended use and in part from the delight of using it—in the same way one delights in watching the aquatic mammal of the same name. The program was designed and implemented for a number of reasons, including to give a person using it access to a tireless team member, offering (1) a wide variety of automated means of drawing conclusions that follow inevitably from the assumptions being supplied and (2) diverse powerful strategies for controlling the program's reasoning.

Problems Solved with Logical Reasoning

The following five example problems, each involving logical reasoning, provide a taste of what a reasoning program (such as OTTER) can do for you. They also suggest charming choices you can make for the inference rules the program uses to draw conclusions; the strategies for restricting and directing its reasoning; and features, such as those for rewriting information into a canonical form. The examples also hint at the unpleasant features of using automated reasoning programs: 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 success. Perhaps as compensation for your labor, however, the program protects you in various ways. For example, a person might easily think of the brother of Bob's aunt as Bob's uncle—which is not necessarily the case. An automated reasoning program would not make such a possibly incorrect translation.

The five problems may help you find new applications, professional and recreational, for automated reasoning. You will discover, among other things, that a program like OTTER does not imitate the approach a person ordinarily takes, which is why the term *artificial intelligence*—especially in view of AI's original intent in the 1960s—does not apply.

Databases. This 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 that the only person who left the parking lot before 5 P.M. was not female. You are then told that Joy left the parking lot before 5 P.M. Finally, you are asked to prove this last assertion is false. Elementary reasoning suffices.

You first conclude that Joy is female; you then use that conclusion to deduce that Joy did not leave the parking lot before 5 P.M. A contradiction is obtained—you have completed a proof by contradiction and, in particular, solved the problem by showing that the last 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? So you can briefly experience what can be less than pleasant, I give some of the possibly pertinent information in clause form [12] (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)`

Without more information, 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 so with, for example, 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 male. Indeed, without the information that every person is female or male (which you used implicitly), the program would be helpless.

Even with that information, the 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 *unit-resulting-resolution*, which works in the following way: The program takes the clause asserting 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 curious about other ways OTTER reasons, and especially about how its reasoning differs sharply from that expected of a person, see the circuit design and the two-inverter problem later in the article.)

Does OTTER have enough data to succeed now? Perhaps, but unless you add *strategy*, the odds are high 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 [12]. You tell the program which statements presenting 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, thereby almost always sharply increasing its efficiency.

Strategy—explicit strategy—is vital to OTTER and to any reasoning program attacking deep questions (see [12] and Figure 1). In contrast, a person seldom uses explicit strategy. In no way does this fact mean that people are not brilliant at problem solving; indeed, we are. It means that OTTER is not designed to emulate a person's reasoning, perhaps explaining why OTTER makes such a valuable team member—*complementing* the approach a person might take.

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 this example, you are asked to design a number of different circuits, but you are constrained to avoid 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 5,000 rules per CPU second. (In contrast to its lack of understanding of most concepts, OTTER "understands" equality, as demonstrated in this and the next example.)

Perhaps you are puzzled. Application of 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* [12] in automated reasoning) and an expression rewritten into its final form only after 1,766 demodulators have 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 [4]—more than 500,000 CPU seconds were spent on demodulation, approximately 75% of the total run time; also see [2] and Figure 2 for a statement of the Robbins problem, which had resisted mathematicians for 60 years. Now you see how what might seem innocent on the surface, namely, simplification, can be a nightmare if attacked by hand rather than by computer.

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, called *paramodulation* [12], that does almost all of the work needed to draw this conclusion immediately. (Paramodulation, explored later, 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, an automated reasoning program finds conclusions of (in a sense) maximum generality. People, on the other hand, typically seek a conclusion that mirrors earlier experience and intuition, although that conclusion may lack the needed generality, in turn preventing them from finding the desired information by, say, 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 can happen easily if done by hand—chaos can result. For but one example, a specific conclusion might be drawn,

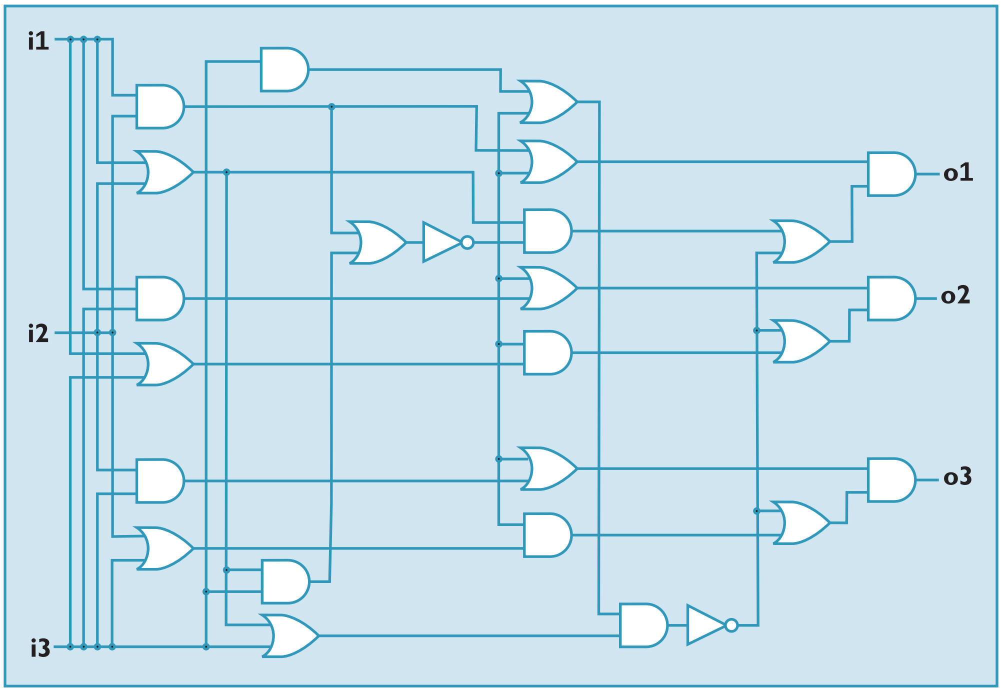


Figure 4. Solution to the two-inverter puzzle, which asked for the design of a circuit in which there are three inputs and three outputs and only two NOT gates

rather than a needed general conclusion. An automated reasoning program is not subject to such misfortune.

A difficult puzzle. Like Lewis Carroll, the 19th-century English mathematician and author of *Alice in Wonderland*, and Raymond Smullyan, the logician and professor of philosophy at Indiana University, many people are fascinated by puzzles. You might find the following billiard ball puzzle stimulating and useful, and want to offer it to colleagues, friends, loved ones, and strangers.

> There are 12 billiard balls, 11 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 whether it is lighter or heavier than the others?

The heart of the problem concerns the constraint of the three weighings. (I'll give you a hint about solving the puzzle after mentioning the number of solutions that were found; stop reading here if you wish to solve the puzzle independently.) 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 11 with identical weight. If they did not balance—even better—you would learn something crucial, putting them aside and marking them as undecided. You would then know that the remaining 10 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 the billiard ball puzzle, like life itself, is not that easy, and you may be surprised to learn that OTTER found more than 40 nontrivially distinct solutions [12] for this puzzle—while some people were still working on one. (Hint: Begin by weighing four balls against four balls.)

Circuit design and the two-inverter problem. The following problem has been included on various exams given to Ph.D. candidates in engineering. The challenge to the circuit designer is twofold: meet all the restrictions and 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 leading nowhere. An auto-

STRATEGY—EXPLICIT STRATEGY—IS VITAL TO OTTER AND TO ANY REASONING PROGRAM ATTACKING DEEP QUESTIONS.

mated reasoning program succeeds easily [12]. Can you solve the following problem?

> Using as many AND and OR gates as you like, but using only two NOT gates, 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 want to glance at the pictorial result in Figure 4, translated from OTTER's success.)

A Grand Return on the Investment

These problems give some idea how complex reasoning often is for a person—and for a computer program. You should also have some feeling for the value of using an automated reasoning program, especially for attacking difficult problems. How comforting it is to know that none of the reasoning contains an error, that the program does not tire, and that it explores paths of reasoning that would be too exhausting for a person. Further, we would still be in the dark about some of the problems whose solutions are now known were it not for the role played by programs like OTTER.

The five example problems are no problem for OTTER; otherwise I would have used "better" examples. But OTTER, as well as other automated reasoning programs, has answered numerous and far more challenging questions, some of which defied human experts for years [4, 5, 9].

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

Consider what would happen if you wanted 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 way to map them into the language accepted by whatever automated reasoning program you intended to use. In another 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 a colleague and I 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 line's efficiency is severely reduced if the cars are sequenced randomly. For example, flushing the painter to refill it with blue and green alternately 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. 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 in 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—"axiomatization" and 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 one developed by Boyer and Moore [1]) have, for example, been used in research [5, 9] and applications [11] to:

- Produce a fully automated proof of the correctness of a division circuit implementing the floating-point IEEE standard.
- Produce correctness proofs of security systems.
- Verify commercial-size adder and multiplier circuits.
- Achieve new mathematics, such as new results in

quasigroups and in non-Euclidean geometry.
- Settle open questions, such as finding minimal axiom sets.
- Confirm conjectures, such as that of Higmann, and the paradox of Gerard.

Reasoning programs have also been used by educators in undergraduate logic courses and in graduate-level courses in interactive theorem proving. And as a recreational example, automated reasoning programs have turned out to be excellent assistants in puzzle solving. (For easy and difficult puzzles, including variations on the well-known checkerboard-and-dominoes puzzle, see [12] and the OTTER Web site.)

Computer-oriented Reasoning vs. Person-oriented Reasoning

Does automated reasoning mirror the classic approach to artificial intelligence? Does OTTER reason somewhat the way a person does? The answer to both questions is a resounding *no*, if one adheres to the strictest definitions.

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

As evidence, focusing on the following three equations, return to the puzzle problem and the use of paramodulation. You obtain the third equation by way of 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:

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

Indeed, if you replace all occurrences of x in the first equation with $-y$ and all occurrences of z in the second with $-(-y)$, 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 yielding identical subexpressions (through unification); and it produces the conclusion without an intermediate (so to speak) subconclusion—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 want an example of what a person does well and what OTTER does not even attempt, you need to turn to what is called *instantiation*. Although perhaps without actual use of the formal name "instantiation," you may have seen this inference rule being used 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 with $uv+vv$ (with u and v being variables); y by $a(b+c)$ for the constants a, b, and c; and x 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 reason, OTTER (and similar programs) do not offer instantiation as an inference rule; I know of no effective strategy for wisely choosing from among the infinite set of instances usually available.

Aware 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 the databases problem example) and paramodulation (used in the arithmetic and mathematics problem example). 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—called *hyperresolution*—that yields conclusions free of logical `not`. The program frequently applies the inference rules to a set of statements containing more than two items—which is not the way a person usually reasons. The program also offers you *binary resolution*, a rule that always focuses on pairs of statements without any constraint on the conclusion that is drawn—which is like a person's reasoning.

Even with such diverse rules for drawing conclusions, all would still be lost were it not for strategy. The program's reasoning must be restricted and must also be directed; see, for example, the arithmetic and mathematics problem and Figure 4.

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

- You have some experience that suggests one subexpression merits preference over another,

such as one involving sum over one involving product. You can use the weighting strategy [12] to guide the program accordingly. In another example, you might want 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—even a wild guess—tells you that the steps of a proof you have in hand could be used profitably to find a proof only distantly related. Either of two strategies—hints [7] or resonance [9]—can be used to achieve your intent.
- 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 using the hot list strategy [10].

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

A Replacement for People?

OTTER is not a replacement for a person. As noted, 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 that approach as you go along. Someday, 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 like OTTER provides excellent incentives for you to rely on it instead of relying exclusively on a person. First, OTTER is remarkably powerful, being able to draw thousands of conclusions per second. And it does not tire, continuing to draw conclusions at virtually the same rate even after millions of conclusions have been drawn.

OTTER does not leave huge gaps in its reasoning—gaps that would force you to guess the not-always-obvious omitted steps. It explicitly presents its work in an output file, including the history of each of its conclusions. If your initial attack on a problem fails, you can examine what OTTER has done and modify your approach or correct the input. You might, for example, find that the problem description is inconsistent.

Moreover, OTTER can have multiple personalities (without the 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 machines. Such a multifaceted attack might enable you to explore radically different approaches that might ordinarily be rejected because of the required effort.

OTTER was not designed to replace people, or even to imitate people. Will it or will any other automated reasoning program replace scientists and engineers? Never. Unquestionably, automated reasoning has made great strides in the past few years. But at most, I expect automated reasoning programs to enable people to devote their energy and time to bigger pictures, having the programs attack smaller problems quickly and flawlessly and, occasionally, without assistance, answering deep questions. C

References

1. Boyer, R., and Moore, J. *A Computational Logic Handbook*. Academic Press, New York, 1988 (see also www.cli.com/software/nqthm/obtaining.html).
2. Kolata, G. With major math proof, brute computers show flash of reasoning power. *New York Times,* Dec. 10, 1996, C1.
3. McCune, W. OTTER 3.0 reference manual and guide. Tech. Rep. ANL-94/6, Argonne National Laboratory, Argonne, Ill., 1994.
4. McCune, W. Solution of the Robbins problem. *J. Autom. Reasoning 19,* 3 (Dec. 1997), 277–318.
5. McCune, W., and Padmanabhan, R. *Automated Deduction in Equational Logic and Cubic Curves. Lecture Notes in Computer Science 1095*. Springer-Verlag, Heidelberg, Germany, 1996; see also www.mcs.anl.gov/home/mccune/ar/monograph/.
6. Parrello, B., Kabat, W., and Wos, L. Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *J. Autom. Reasoning 2,* 1 (1986), 1–42.
7. Veroff, R. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *J. Autom. Reasoning 16,* 3 (June 1996), 223–239.
8. Wos, L. *Automated Reasoning: 33 Basic Research Problems*. Prentice-Hall, Englewood Cliffs, N.J., 1987.
9. Wos, L. *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial*. Academic Press, New York, 1996.
10. Wos, L. OTTER and the Moufang identity problem. *J. Autom. Reasoning 17,* 2 (Oct. 1996), 259–289.
11. Wos, L., and Pieper, G., eds. Special issue on automated reasoning. *Comput. Math. Appl. 29,* 2 (Feb. 1995), 133–178.
12. Wos, L., Overbeek, R., Lusk, E., and Boyle, J. *Automated Reasoning: Introduction and Applications*. 2d ed. McGraw-Hill, New York, 1992.

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.

Larry Wos (wos@mcs.anl.gov) is a senior mathematician in the Mathematics and Computer Science Division of Argonne National Laboratory in Argonne, Ill. He is president of the Association for Automated Reasoning, a member of the editorial board of the *Journal of Automated Reasoning* (which he founded in 1983), and the first recipient of the Herbrand Award in Automated Deduction, presented in 1992.