Automating the Search for Answers to Open Questions*

Larry Wos[1] and Branden Fitelson[1,2]

[1] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4801,
wos@mcs.anl.gov
[2] University of Wisconsin, Department of Philosophy, Madison, WI 53706,
fitelson@facstaff.wisc.edu

Abstract. This article provides evidence for the arrival of automated reasoning. Indeed, one of its primary goals of the early 1960s has been reached: The use of an automated reasoning program frequently leads to significant contributions to mathematics and to logic. In addition, although not clearly an original objective, the use of such a program now plays an important role for chip design and for program verification. That importance can be sharply increased; indeed, in this article we discuss the possible value of automated reasoning to finding better designs of chips, circuits, and computer code. We also provide insight into the mechanisms—in particular, strategy—that have led to numerous successes. To complement the evidence we present and to encourage further research, we offer challenges and open questions for consideration. We include a glimpse of the future and some commentary on the possibly unexpected benefits of automating the search for answers to open questions.

1 An Unlikely but Realized Dream

In the late 1940s, researchers (including the logician Luukasiewicz) considered the possibility of *mechanically checking a proof* to be within reach. At the other end of the spectrum, some thought *mechanical proof finding* to be out of the question. For many, that view was still extant in the early 1960s, when an audacious effort seriously commenced whose objective was the design of a computer program that could make significant contributions to mathematics and to logic. In other words, some researchers (brave or foolish) embarked on a journey whose destination was proof finding of interesting theorems—and, if gold was found, the automated answering of open questions.

The search for answers to open questions is frequently thought to be the dominion of mathematics and logic. However, closely related are questions posed by the designers of circuits and chips and by the writers of computer programs.

* This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Such questions take various forms. Does a given design of a chip or circuit meet its specifications—is it free of bugs? Given a bit of computer code (or a given design), can one find an alternative that offers far more efficiency? In this article, we make a case for using a program to find better designs, where the methodologies are taken from our recent successful research in finding shorter proofs; see the Appendix, which takes the form of a whitepaper.

As for mathematics and logic, as the following list illustrates, open questions come in an even greater variety of flavors.

1. Is every Robbins algebra a Boolean algebra?
2. Is the formula XHN a single axiom for equivalential calculus?
3. Does there exist a circle of pure proofs for the Moufang identities?
4. Does a particular identity that holds in orthomodular lattices also hold in ortholattices?
5. Does the fragment of combinatory logic with basis consisting of B and M satisfy the strong fixed point property?
6. Is the formula XCB a single axiom for equivalential calculus?

Answers to the cited questions—and to many, many more of diverse types—can often be found by relying heavily on an automated reasoning assistant. Indeed, the first four have already been answered with the help of Argonne's powerful automated reasoning programs [3, 7, 6, 1]; for a discussion of the two open questions, 5 and 6, see Sect. 3. A delightful bonus: To enlist the aid of a reasoning program in the search for answers, one need not be an expert. Today, various mathematicians only vaguely familiar with automated reasoning are using William McCune's reasoning program OTTER [2]—the program featured in this article—in their research. (If guidance is desired in the use of OTTER, if a fuller understanding of the elements of automated reasoning is the objective, or if one seeks open questions to attack, each is easily within reach through consulting the new book *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning* [7]; its included CD-ROM is a gold mine. If one wishes to browse in a dense forest of once-open questions answered with the use of OTTER, the monograph by McCune and Padmanabhan [4] is the choice.)

Complementing the use by mathematicians of automated reasoning and also contributing substantially to the realization of the dream of profitably using a reasoning program is the use by firms that include AMD and Intel. In particular, the cited firms each employ people whose assignment is to prove theorems in the context of correctness of various designs. The chief weapon is recourse to a reasoning program. Was part of the motivation the remarkable achievement of Boyer, Moore, and colleagues in their design and verification of a chip and language [5], a chip that was eventually manufactured and used?

1.1 The Source of Power

Although for many years OTTER was the fastest reasoning program, programs now exist that run faster. But (in our view), CPU speed is not the key. Indeed, an

increase in CPU power of a factor of 4 (we conjecture) brings very few theorems in range that were previously out of range. The obstacle rests with the incredible size of the space of deducible conclusions.

And here is where the power resides that has led to so many recent successes: OTTER offers a variety of powerful strategies, some to restrict its reasoning, some to direct its reasoning, and some to permit the program to emphasize the role of certain designated information. We strongly conjecture that without access to various types of strategy, the vast majority of the successes of the past few years would not have been reached. For a glimpse of how all has changed, we note that, in contrast to two decades ago, our submission to OTTER of problems taken from various areas of logic is almost always met with a proof. (We intend to present many new proofs in print and on an included CD-ROM in a planned book entitled *Automated Reasoning and the Finding of Missing and Elegant Proofs in Formal Logic*.)

Rather than formal definitions, we give the following examples of strategy in terms of their objective.

1. The *set of support strategy* typically restricts a program from exploring the space of conclusions that follow from the axioms.
2. The *expression complexity strategy* restricts a program from considering terms, formulas, or equations conjectured to interfere with effectiveness because of their complexity.
3. The *variable richness strategy* restricts the program from considering formulas or equations whose number of distinct variables appears to make them unattractive.
4. The *term-avoidance strategy* prevents the program from retaining any newly deduced conclusion that contains a term in the class of those designated as unwanted; see Sect. 4 for a fuller discussion.

In contrast to the preceding four strategies that restrict a program's reasoning, the following useful strategies direct its reasoning.

5. The *resonance strategy* instructs a program to focus on conclusions that resemble any of a set designated by the researcher as appealing, in preference to all other available conclusions.
6. The *ratio strategy* instructs a program to choose k conclusions by complexity, 1 by first come first serve, then k, then 1, and the like, where k is assigned by the researcher.
7. The *weighting strategy* directs a program to focus on items whose complexity is smallest, where the complexity is in part determined by user-assigned values.

Among other factors, additional power is derived from a mechanism to simplify and canonicalize information and from a variety of inference rules, one of which treats equality as "understood". However, in contrast to so many reasoning programs, (we maintain that) OTTER's offering a veritable arsenal of strategies is the key.

2 Open Questions Detailed

At this point, we provide the promised details concerning questions 5 and 6 posed in Sect. 1. Each, as noted, remains open.

For the first of the two questions (5), we give the definitions of the combinators B and M and that of a fixed point combinator $\mathbf{F}$. (Expressions in combinatory logic are assumed to be left associated unless otherwise indicated.)

```
Bxyz = x(yz)
Mx = xx
Fx = x(Fx)
```

Does there exist a fixed point combinator $\mathbf{F}$ expressed purely in terms of the combinators B and M such that $\mathbf{F}x = x(\mathbf{F}x)$? To provide a small taste of the nature of this question, we note that BML is a fixed point combinator for the fragment whose basis consists solely of B, M, and L, where $Lxy = x(yy)$.

For the second question (6) that remains open, we give two definitions, the second of which is included for pedagogical reasons.

```
e(x,e(e(e(x,y),e(z,y)),z))  %  XCB
e(x,e(e(y,z),e(e(z,x),y)))  %  XHN
```

Is the formula XCB a single axiom for all of equivalential calculus? The formula XHN is a single axiom. If one were able to deduce some known single axiom, such as XHN, starting with XCB, then one would have established that XCB is also a single axiom. On the other hand, if the goal is to disprove the implied theorem, then an obvious approach is to find an appropriate counterexample in the form of a model.

3 From Mathematics and Logic to Design

Astounding to us, many researchers in the field do not share our enthusiasm for attacking open questions via automated reasoning. Our eagerness is of course based in part on the desire to *know* the answer: Is there a proof, or is there a counterexample in the form of a model? A glance at our research clearly establishes our preference for areas of mathematics and logic.

In addition to contributing to mathematics and to logic, however, our studies of the automation of an attack on open questions have two important benefits. First, because sometimes we fail in our early attempts, we are forced to formulate and then implement new approaches, methodologies, and strategies. We always aim at generality. Therefore, independent of finding an answer to an open question, we produce mechanisms that increase (in a significant manner) the power of reasoning programs. That increase in turn brings into range additional open questions, and the loop continues.

A second benefit is that some of the new approaches, methodologies, and strategies appear to offer much for design and verification. To show how this might be true, we note that another class of open questions exists, outside of the

usually cited classes. That class can be termed *missing proofs*. For one example, if the only proofs of a given theorem are by induction or by some other metaargument, then an axiomatic proof is missing. For a second example, if a theorem is announced without proof by a master (which removes any doubt about its truth), then again a proof is missing. For a third example (of the many types we have identified)—and an example that is pertinent to design—if a proof is in hand and the conjecture is that a rather shorter proof exists but has not yet been found, then again a proof is missing.

In the past two years, we have devoted substantial effort to finding missing proofs, heavily emphasizing the search for shorter proofs. The term-avoidance strategy and the resonance strategy have played key roles in our numerous successes. Those studies, in our view, could be put to great use in design. Indeed, imagine that one has in hand a design of a chip, a circuit, or a computer program. Our approach would be to obtain a constructive proof with OTTER of the given design and then (in the context of the resonance strategy) use the deduced steps of that proof in search of a better design. Then, as part of our effort and for a tiny taste of what we would do next, we would instruct the program to avoid the use of each of the deduced steps (one at a time) to see whether a shorter proof could be found.

Although clearly nothing like an isomorphism or a guarantee, the shorter the proof, the simpler the constructed object—chip, circuit, or program. A simpler object (everything being equal) is easier to verify, is more reliable, makes better use of energy, and produces less heat. In other words (with almost all of the details omitted), we suspect that design and verification would benefit from adapting our recent research.

Important to note is an easily overlooked subtlety when a shorter proof is the goal. Imagine that the goal is to find a proof (shorter than that in hand) of Q **and** R **and** S. As the attack proceeds, shorter and still shorter proofs may be found of, say, S, which might lead one to the conclusion that ever-shorter proofs of the conjunction are in the making. Quite often, such is not the case. Indeed, a shorter proof of a member of the conjunction may be such that the omitted steps (from the longer proof) are useful in the total proof, where their replacements serve no other purpose than that of producing a shorter proof of the cited member. The situation in focus may be familiar to programmers or circuit designers. Reliance on a subroutine with fewer instructions or reliance on a subcircuit with fewer components does not necessarily add efficiency for the larger program or circuit. This amusing subtlety makes the finding of shorter proofs far more difficult than it might at first appear.

As part of our recent studies, we have also focused on term avoidance. The decision to employ the strategy might be based on mere curiosity (as so often occurs in mathematics and in logic), or it might be based on practical considerations (as frequently occurs in design and verification). Indeed, regarding the former, one might wonder about the existence of a proof in which nested negation is forbidden, a proof in which no deduced step contains a term of the form $n(n(t))$ for any term t where the function n denotes negation. As for the latter,

because of economy or efficiency, one might wish to seek an object in which some type of component or instruction is absent. For example, one might wish to avoid the use of **NOR** gates.

In contrast to the just-cited motivations for use of the term-avoidance strategy, we find that its use markedly increases the likelihood of success in the context of automating the search for answers to open questions. The explanation is quite subtle; indeed, adding a constraint might on the surface make finding a proof much harder. Note that the space of deducible conclusions can grow exponentially as a program's attack proceeds. This property is directly addressed by (apparently arbitrarily) choosing a type of term to be avoided and then preventing the program from venturing into the subspace of deducible conclusions each of which contains one or more occurrences of such a term. Of course, depending on the type of term to be avoided, one's intuition might balk. For example, in the case of avoiding nested negation, one might understandably doubt that the objective can be reached. However, we have almost always succeeded in the presence of this constraint. Perhaps the explanation rests with (1) the existence of many, many more proofs than one might expect and (2) the removal of (apparently) distracting information. Put another way, by avoiding conclusions of a specified type, it appears that the density of good information within that which is retained is sharply increased.

In view of our recent successes, we conjecture that those involved in design and verification might benefit from our various methodologies focusing on finding proofs in which some class of terms is absent.

4 The Future

We have focused on how our recent successes in finding proofs with an automated reasoning program are potentially valuable to design and validation. In the near-term future, perhaps some firm will submit to us a design in the clause language OTTER employs. Also required is the property that OTTER can produce a constructive proof that the design meets the specifications. We would then attempt to simplify the proof, fewer steps, less complex expressions, and perhaps the avoidance of certain classes of term. If success were to occur, the firm might then have a better design.

We also envision in the future a rather odd use of a type of parallelism. Specifically, the likelihood of success regardless of the goal (we conjecture) would be sharply increased if one had access to a large network of computers. When the objective was identified, the set of computers (perhaps 10,000) would each separately attack the problem, each in a manner somewhat different from the rest. For example, each might employ a different bound on the complexity of retained information, a different value for the parameter that governs the actions of the ratio strategy, a different set of resonators, and the like. All members of the set of assigned computers would simultaneously attack the problem. Currently, we rely on a miniscule version of this approach, sometimes resulting in success.

In summary, we estimate the current state of automated reasoning to be fifty years ahead of what an optimist might have predicted but twenty years ago. The explanation rests mainly with the formulation of new and diverse strategies. For dramatically greater advances, we conjecture that strategy still holds the key.

References

1. McCune, W.: Automatic Proofs and Counterexamples for Some Ortholattice Identities, Information Processing Letters **65** (1998), 285–291
2. McCune, W.: Otter 3.0 Reference Manual and Guide. Technical report ANL-94/6. Argonne National Laboratory, Argonne, Illinois (1994)
3. McCune, W.: Solution of the Robbins Problem, J. Automated Reasoning **19**, no. 3 (December 1997) 263–276
4. McCune, M., and Padmanabhan, R.: Automated Deduction in Equational Logic and Cubic Curves. Lecture Notes in Computer Science, Vol. 1095. Springer-Verlag, New York (1996)
5. Moore, J S.: System Verification (special issue), J. Automated Reasoning **5**, no. 4 (December 1989) 409–544
6. Wos, L.: Otter and the Moufang Identity Problem, J. Automated Reasoning **17**, no. 2 (1996) 215–257
7. Wos, L., and Pieper, G. W.: A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning. World Scientific, Singapore (1999)