The Review of Modern Logic
Volume 9 Numbers 1 & 2 (November 2001–November 2003) [Issue 29], pp. 195–202.

Larry Wos and Gail W. Pieper (eds.)
The Collected Works of Larry Wos
Volume I: *Exploring the Power of Automated Reasoning*
Volume II: *Applying Automated Reasoning to Puzzles, Problems, and Open Questions*
Singapore: World Scientific, 2000
xvii + 1639 pp. ISBN 9810240015

REVIEW

VLADIK KREINOVICH

What is "automated reasoning"? Crudely speaking, *automated reasoning* is an area whose main objective is to make computers solve hard open problems in mathematics and in other well-defined formal areas.

Is it successful? Oh yes, there have been quite a few success stories when automated reasoning programs succeeded in solving long-standing open mathematical problems.

Probably the most well-known of these problems is the antiautomorphism semigroup problem. This problem was originally formulated by a well-known algebraist I. Kaplansky as a challenge to the automated reasoning community. Let S be a *semigroup*, *i.e.*, a set with an associative operation $*$. A mapping $f : S \to S$ is called an *antiautomorphism* if $f(x * y) = f(y) * f(x)$ for all x and y. Many semigroups have antiautomorphisms: *e.g.*, for every natural number n, transposition is an antiautomorphism on the semigroup of all $n \times n$ matrices. This particular antiautomorphism is an *involution*, *i.e.*, $f(f(x)) = x$ for every x. At the time when Kaplansky formulated this question, in every known finite semigroup with an antiautomorphism there was also an antiautomorphism which is an involution. It was therefore conjectured that every finite semigroup with an antiautomorphism has an antiautomorphism which is an involution. Mathematicians tried hard, but could neither prove this conjecture nor find a counterexample.

An automated reasoning program succeeded in finding a non-trivial counterexample — a semigroup of 83-th order!

There have been many other successful examples of computers solving open mathematical problems. Many long-standing questions about independence of axioms, about the shortest single axiom for different classes of algebraic structures, about combinatorial logic, *etc.*, have been successfully solved by using automated reasoning.

Why are these success stories not getting the praise they deserve? Chess computers — whose success was largely due to their pure processing speed — made it to the front pages, but a much more intellectually interesting computer proofs remain relatively unknown. Why?

One reason: these successes were not surprising. One reason for this is that these computer proof successes were not unexpected. In the 1950s and 1960s, every journalist writing about computers knew that very soon, computers will be able to prove theorems. All you need to do is start with axioms, apply inference rules again and again, and wait until the desired statement (or its negation) pops up. OK, maybe not so easy, maybe you need to also start with the statement itself, but anyway, there are successes already, and with faster computers, mathematicians would be soon obsolete.

Well, it did not happen that soon as optimists predicted, and mathematicians are not yet obsolete, but it is finally happening. To many people outside the automated reasoning community, what is surprising is not that computers can solve open mathematical problems, but rather that it happened so slowly and on a much smaller scale than the original hype suggested.

Second reason: people grossly underestimate intelligence behind the corresponding programs. Researchers who are not very familiar with automated reasoning sometimes underestimate the intellectual effort behind the success stories. This underestimation is easy to understand. When they browse the papers describing details of these success stories, they see mentions of thousands and millions of computer generated intermediate statements, they see a lot of seemingly random examples of such statements, and they get the erroneous impression that what is going on here is exactly what the original journalists described: random search for a proof, OK, maybe somewhat trimmed and pruned, but still largely random.

This impression is easy to disprove: for a proof consisting of 80+ steps (and computers produced such proofs already in the 1980s), even if we had 2 choices on each step, a random generation would require

testing 2^{80} ($\approx 10^{24}$) possible proofs — clear impossibility. In reality, such a computer proof usually includes thousands of intermediate formulas. From 10^{24} to 10^3 — some trimming! This is not just one idea, this downsizing of the search space is what automated reasoning has achieved by its numerous successful ideas.

In computing, there is always a kind of a rivalry between hardware and software people. Hardware folks tend to sometimes make fun of our successes: when we put a lot of effort and speed up an algorithm 10 times, this is a big success story, all over math news — but they have been steadily achieving the same speedup every 5 years. Automated reasoning is our success story: the $10^{24}/10^3 = 10^{21}$ times speedup is something that hardware folks cannot match!

Third reason: confusion with AI. The third reason why this area does not get its due is because of the confusion with AI. Many AI program reason, so why not use these programs (*e.g.*, Prolog-based) to prove theorems as well?

On the surface, it may sound like a good idea, but in reality, AI programs do not work well in automatic reasoning. The reason is simple: in AI, typically, we solve problems that are not that difficult to solve. For example, a typical AI problem is robot navigation. It is not a very easy problem to solve automatically, but an operator can easily navigate a robot. Similarly, playing chess: it is not easy to design a program but many humans play chess reasonably well. Because AI is mainly about such problems, AI heuristics help solve such problems faster and better.

In automated reasoning, problems are very different. Here, the main difficulty is not so much automating human reasoning, but coming up with some reasoning at all, because humans cannot solve such problems either. These problems are different, and not surprisingly, standard AI heuristics do not work well.

Who is Larry Wos and what this book is about. A good illustration of the fact that automated reasoning does not get the recognition it deserves is that, based on the reaction of some of my colleagues, I have to explain who Larry Wos is.

Larry Wos is *the* Mr. Automated Reasoning. He is the recognized leader of this field, the founding editor of the *Journal of Automated Reasoning* (the main journal in this area), the first winner of the American Mathematical Society prize in Automated Reasoning, and of many other prizes.

Larry was in the field when it started, he followed it up, and this book, with a chronological collection of his papers — most of which were breakthroughs and milestones — give a pretty good picture of how this field evolved. And it did evolve, from the original ideas (that were, honestly, not too far away from the random search as envisioned by the science journalists), to modern successes, to the unbelievable 10^{21} speedup.

How the field evolved. The book gives a good overview of how the field evolved. Of course, our subdivision into stages is very schematic. In reality, these stages overlapped, but we still give it because it provides a reasonable big picture:

(1) *Theory* (1967–73) At first, the main effort was on finding possible proof search strategies. Here, the main emphasis was not yet on efficiency, but rather on correctness of the results (soundness) and on the possibility to actually find the proofs of all provable statements by using these strategies (completeness). This was a difficult analysis because it was produced before any programming started, with no way to run computer simulations and thus check and guide results.
(2) *Known problems* (1974–76) After this analysis was over and several reasonable strategies were proven to be sound and complete, the next step was to try these strategies on known mathematical results. These tries were used to adjust and tune the strategies until they finally covered the known results.
(3) *New problems* (1977–80) After the strategies were evolved enough to be successful on known problems, Wos and others tried them on open problems. This led to the first true successes: several open problems have been solved.
(4) *Challenging problems* (1981–83) On the third stage, researchers applied their strategies to several open problems, with the positive result that *some* of these problems got solved (but others did not). On one hand, it was a success story, because quite a few open problems were actually solved. On the other hand, the ideal automated reasoning tool should be able not just to solve a few of open problems, but hopefully most of them. So, on the next stage of this work, the authors solicited open problems and tried to adjust their programs in such a way so as to solve these particular problems. This lead to the 1981 success with the above Kaplansky problem. Several other open problems were successfully solved in this manner.

(5) *Practical problems* (1984–95) Many applied problems, when well understood, can be naturally reformulated in mathematical terms. So, if we have a tool that can prove theorems, we therefore have a tool that can potentially solve practical problems. This is the main idea, it is not that simple, but after some adjustment, the automated reasoning tools were indeed successfully applied to solving such practical problems as circuit design, proving program correctness, and job scheduling (a known NP-hard problem).
(6) *Elegant proofs* (1995–98) Readers familiar with discussions related to the computer proof of the four colors problems will easily recall that the main problem with that proof is that it was tedious, ugly, and impossible to understand. This example is a good illustration of what computer proofs used to be: not elegant. It turns out that in automated reasoning, in addition to looking for *a* proof, we can also successfully look for an *elegant* proof. As a result, modern automated reasoning tools can be used not only to find proofs for open problems, but also to look for elegant proofs for problems for which only ugly proofs were previously known.

What methods were used. At each stage of the above evolution, new methods, algorithms, and ideas were developed. Crudely speaking, we can see three stages in this development:

- At first, the main emphasis was on pruning and limiting the search space. One of the main ideas here is the idea of subsumption: that if a statement is a particular case of an already proven one, then we can keep the more general statement and dismiss less general statements. This idea led Wos to his more sophisticated 1967 idea of *demodulation*: discarding not only particular cases of a given statement, but, crudely speaking, also instances of repeated application of a given equality.
- After pruning techniques became more and more sophisticated, it became more and more difficult to achieve improvement by further pruning. So, the main theoretical emphasis somewhat shifted to developing new ("linked") inference rules from the old ones. Such rules speed up the search for a proof because in one application of a new rule, we are able, in effect, to achieve the same result as by applying several old ones. This "meta-approach" (looking for new rules instead of simply applying them) turned out to be also very successful.

- In spite of these successes, there were cases when the tools did not work well. One reason for this is that when a researcher explains an open problem to a student, she not only describes this problem in informal terms, she also provides some informal intuition about this problem:
  - This intuition may include the idea that some mathematical statements may be useful in this proof.
  - It may also include the idea (in analogy with known proofs of similar statements) that a natural way to the desired proof is through a lemma of certain type.

  In the 1990s, Wos developed two ideas which enabled the automated reasoning program to use such information:
  - a *hot list* strategy, in which statements marked by a researcher as possibly useful are used more frequently than others; and
  - a *resonance* strategy in which a program looks for proofs with lemmas of a certain type.

  When combined, these two strategies greatly improved the programs.

Lessons learned. At first glance, it may sound as if the history of automated reasoning followed a very natural path: intuitively natural ideas were transformed into programs, tuned, and successfully applied. Alas, life is never so straightforward. As Larry Wos mentions (on pp. 853, 1182–1186, *etc.*), the following seemingly natural ideas turned out to be completely wrong:

- *Exponential growth?* Since the search space grows exponentially with the size of the proof, it is natural to expect that even after trimming, we will still get an exponential dependence of proof time on the proof length. This expectation may have been one of the reasons why many researchers did not even bother to try to find proofs with 80+ steps. And this pessimistic expectation turned out to be wrong. In reality, after smart trimmings, the computation time grows much more slowly than exponentially and, as a result, we can actually get a proof. This counter-intuitive result is good news. It is in good accordance with the fact that, *e.g.*, logic programming, in which most problems are NP-hard, is actually a very successful practical tool: exponential growth is about worst case, not about the practically important average complexity.
- *Now or never?* Intuitively, if we try and try and try and do not succeed, we need to give up, cut the losses, and try something

else. This may be a good commonsense advice, but interestingly, this advise turned out to be too pessimistic for automated reasoning: often, when a program does not find a proof in reasonable time, it finds it eventually. This fact should not be so surprising: after all, this is how open problems are solved by humans: we try and try and then eventually, someone succeeds.

- *From simple to complex?* In general, how do we analyze complex objects? First, we try simple ones, learn the techniques, and then scale these techniques to more complex ones. This is how we analyze and design circuits, this is how we analyze and design programs, *etc.* It may sound natural to expect that this is how we should look for proofs. On the example of simple statements, we find which proof strategies work the best, and then apply these best strategies to the desired complex open problem. Alas, it does not work. Why? Because for those problem for which it works, we have already succeeded in using known proof strategies in our brain, and as a result, we have already proved the results. What remains is challenging problems which are still open, and the very fact that they are open means that a simple scaling of known strategies did not work well, new ideas are needed.
- *Specialists are needed?* If an algebraist who is normally in groups cannot solve a problem related to semigroups, then a natural idea is to consult a semigroup specialist. Similarly, it seems to make sense, for each area of mathematics, to fine-tune search techniques to this particular area and then use these tuned techniques to solve open problems. Does not work. Why? For the same reason as in the previous example: if it worked then this problem would not have been open. The very fact that this problem is open means that the methods well used in this particular area have not succeeded, so other techniques are necessary.
- *There is the best strategy?* We are accustomed to thinking is sports-like terms: one computer is better, one algorithm is better. From this viewpoint, we expect that of different proof search strategies, one would be the best, and the question is how to find this The Best Strategy. Experimentally, however, there is never a clear winner: on some problems, one strategy is better, or other problems, the other is better. In such situations, a natural idea is to combine the best ideas of these two good strategies into a single strategy that would be better

than each of them. This combination is possible, but this combination can be done in several different ways. Usually, there is no clear winner between these different combinations, so we end up with several combined strategies and again without a single "the best" one. By the way, it is not clear how we can theoretically explain this empirical fact (pp. 1193 ff.).

Conclusion. This book gives us a unique opportunity to look into the area of automated reasoning, into its insights, evolution, successes, and challenges. It shows that seemingly intuitive ideas do not work and thus, it changes your viewpoint. And it make you think.

Department of Computer Science, University of Texas at El Paso, El Paso, TX 79968, USA
E-mail address: vladik@cs.utep.edu