# Automated Theorem Proving and Its Prospects

Review of *Automated Development of Fundamental Mathematical Theories* by Art Quaife

**Desmond Fearnley-Sander**
Department of Mathematics
University of Tasmania
HOBART TAS 7001
AUSTRALIA

dfs@hilbert.maths.utas.edu.au

Received: December 28, 1995; Accepted: March 17, 1996; Published: June 19, 1996

PSYCHE, 2(28), June 1996
http://psyche.cs.monash.edu.au/v2/psyche-2-28-fearnley_sande.html

KEYWORDS: automated theorem proving, OTTER, Quaife, resolution, understanding.

REVIEW OF: *Automated Development of Fundamental Mathematical Theories* by Art Quaife (1992). Kluwer Academic Publishers. 271pp. $US123 hbk. ISBN: 0-7923-2021-2.

## 1. Introduction

In his recent books one of the main pre-occupations of Roger Penrose has been to show (to prove!) that computers are intrinsically limited, compared to humans, when it comes to the doing of mathematics. Even those who think that such things can be proved may be interested in the empirical question: what mathematics can computers do? Art Quaife's book addresses the aspect of doing mathematics that is reported in mathematical journals: proof of theorems. It shows us the extent to which this high exercise of rationality can currently be automated, and the extent to which it cannot.

To me this book was fascinating. After reading it, I found myself turning over in my mind the question of the role played in theorem proving by understanding, a quality that Penrose regards as a prime desideratum of consciousness. I'll come back to that at the end of this review.

Using the theorem prover OTTER Art Quaife has proved four hundred theorems of von Neumann-Bernays-Gödel set theory; twelve hundred theorems and definitions of elementary number theory; dozens of Euclidean geometry theorems; and Gödel's incompleteness theorems. It is an impressive achievement. To gauge its significance and to see what prospects it offers one must look closely at the book and the proofs it presents. But first, some basics.

## 2. Resolution theorem proving

Mainstream automated theorem proving, to which Quaife's work belongs, is based on the method of resolution. To get an idea of it, three fundamental notions are needed: *sentence*, *theorem* and *proof*.

Classical logic provides us with a notion of *sentence* that is completely formal. A person or computer can mechanically check whether or not a given string of words is a (grammatically correct) sentence. A typical sentence like:

```
for_all x, for_all y, x*y=y*x
```

may admit interpretation in various mathematical domains; this particular one happens to be true for multiplication of real numbers but not for multiplication of 2-by-2 matrices. The sentence

```
for_all x, there_exists y, x*y=1
```

is true if `*` is interpreted as addition of real numbers (and `1` as the number we call 'one', or even as the real number we call 'zero'), but it is not true if `*` is interpreted as multiplication of real numbers and (and `1` as the number we call 'one').

A theorem has a bunch of sentences called its *hypotheses*, and a single sentence called its *conclusion*; what makes it a *theorem* is the existence of a proof that the conclusion follows from the hypotheses. A *proof* is a guarantee that under any interpretation in which the hypotheses are all true the conclusion must also true. While there are many methods of proof, resolution has the advantage of being more susceptible than most to machine implementation.

Resolution is a method of proof by contradiction: to show the conclusion *C* follows from the hypotheses *H1, H2, ... , Hn* we say "suppose on the contrary that (in some interpretation) the hypotheses are true but the conclusion is not" and proceed to show, if we can, that the set of sentences {*H1, H2, ... , Hn, not C*} is contradictory. Our proof will succeed if we can find a sentence *S* such that both *S* and *not S* follow from the given sentences.

What is required, then, to prove a theorem is to show that an associated set of sentences is unsatisfiable. In fact one may always construct the associated set so that its sentences have a particularly simple form: each sentence is a disjunction of literals -- a literal being an atomic sentence or the negation of an atomic sentence -- that is universally quantified over all its variables. Sentences of this simple form are called *clauses*; for brevity, it is normal in writing clauses, to omit the universal quantifier and to replace each `or` (connecting the literals) by a comma.

For example the clause `{not is_a_man(x), is_mortal(x)}` is a formal abbreviation for the sentence "for every x, either x is not a man or x is mortal"; which is more familiar, perhaps, in the logically equivalent form "all men are mortal." Let us prove the famous theorem that has this and the clause `{is_a_man(Socrates)}` as its hypotheses and `{is_mortal(Socrates)}` as its conclusion. The method of resolution requires that a contradiction be derived from the set of three clauses:

```
{{not is_a_man(x), is_mortal(x)},
{is_a_man(Socrates)},
{not is_mortal(Socrates)}}
```

Using the first and the second of the clauses we may derive the new clause `{is_mortal(Socrates)}`. This is a very simple instance of what is called a *resolution step*; we will not go into detail here, but all that is involved is careful, elaborate, systematic matching and generation of patterns of symbols. Adding the new clause to our set, we now have:

```
{{not is_a_man(x), is_mortal(x)},
 {is_a_man(Socrates)},
{not is_mortal(Socrates)},
{is_mortal(Socrates)}}
```

and there, staring us in the face, is the contradiction we need. A resolution prover would perform a second resolution step, by convention deriving the null clause {} from the final two clauses above, and ending with

```
{{not is_a_man(x), is_mortal(x)},
{is_a_man(Socrates)},
{}}
```

It is the presence of the null clause that signals success -- a contradiction has been derived.

This may seem a mindless way to prove the quintessential example of obviousness, but it's the way that the method of resolution works. One keeps adding new clauses to those already generated, sometimes (as in the second step of our example) deleting old ones. Resolution is the main way of generating new clauses but another kind of step, called factoring is also needed. Any other sound way of deriving new clauses may be used. If we succeed in deriving the empty clause we are done: the theorem is proved. What makes this worth trying is the fact that for any unsatisfiable set of clauses there exists a finite sequence of resolution and factoring steps that leads to a contradiction. We only need to find it.

Quaife's book actually uses a more perspicuous Prolog-like notation in which `A, B -> C, D` stands for the clause `{not A, not B, C, D}`, and displays resolution proofs in familiar linear fashion, presenting each new clause that is generated with a brief explanation of its origin. For example, the argument above would appear in the form:

1 `is_a_man(x) -> is_mortal(x)`
2 `-> is_a_man(Socrates)`
3 `is_mortal(Socrates) ->`
4 `-> is_mortal(Socrates)` [resolution, 1, 2]
5 -> [resolution, 3, 4]

## 3. OTTER

OTTER was developed at Argonne National Laboratory by William McCune (1990). It is based on work of Larry Wos, Ross Overbeek, Ewing Lusk and others (Lusk and Overbeek 1982, Wos et al. 1965, Wos et al. 1967, Boyer et al. 1986) that goes back over more than two decades and stems ultimately from the resolution algorithm of Alan Robinson (1963). OTTER is a state-of-the-art theorem prover incorporating many refinements of the basic resolution method (though at the time of publication it lacked associative-commutative unification).

Theorem-proving, from this point of view, is like searching an immense, a mind-bogglingly immense, maze. If the statement we are trying to prove is indeed a theorem -- and of course we are not supposed to know that it is - then among the many paths that lead from the set of clauses at hand to sets of derived clauses there will be some that lead to an exit -- a set that includes a null clause. If the statement happens not to be a theorem, then the maze has no exit. We accumulate clauses in the hope that they may play a role in filling in the path we seek. In the vast amount of information that is generated, most is dross. The problem is to control the derivation process in such a way that the null clause comes along before the information at hand becomes too large for any computer to handle. And this is to be done without semantic input, without understanding. The author quotes Henri Poincaré, with apparent approval:

Thus be it understood, to demonstrate a theorem, it is neither necessary nor even advantageous to know what it means.

It is an uncharacteristic sentence. When one tracks it down -- a non-trivial task, since like all of Quaife's many quotations it is unreferenced -- it can be seen that Poincaré was only paraphrasing sentiments of Hilbert with which he disagreed. The relevant passage is in Chapter 3 of Poincaré's *Science and Method* (1958, p. 147).

In Quaife's work binary resolution is abandoned and sophisticated variants, hyperresolution and UR-resolution are used instead. As he puts it (p. 14) these "make larger inference steps in one fell swoop, without saving intermediate results to further clog up the clause space. They are effective steps in fighting the combinatorial explosion." In addition, reasoning about equality is done using paramodulation and demodulation. The latter brings in the powerful algebraic device of rewriting to normal form. A most important role is also played by two common-sense strategies: set-of-support, which ensures that each conclusion drawn is relevant to the theorem at hand, and subsumption, which discards a derived clause if it is less general than some other derived clause.

OTTER offers the user a substantial degree of control over the process of generating new clauses. In particular, each generated clause is assigned a weight that may be used to discard the clause (if its weight is large) or to give it preference in the drawing of further inferences (if it is small). More importantly, the program, like human mathematicians, can invoke previously proved results, however the choice of which theorems to prove first is up to the user of the program.

The question must be asked: can proofs obtained with so much input from the human user be said to be automatic? The author calls them "semi-automatic" and, revealingly, he consistently refers to them in the possessive as "my proofs." Referring to the range of possibilities between a prover that merely verifies each minute step of an argument entirely presented to it by the user and one that presents a complete proof given only the statement of the theorem (including the axioms of the relevant theory), he makes a case that his development is a positive step in the direction of automation. He argues that whereas earlier resolution provers were successful only when relevant lemmas were supplied by the user, in his presentation all previously proved lemmas are freely available. There is therefore less human intervention.

Art Quaife is optimistic that human intervention can be eliminated. Are there grounds for this optimism? In the case, for example, of number theory the order of proof of theorems and introduction of concepts is grounded in hundreds of years of human experience. Deep semantic insights of generations of creative mathematicians have been called upon. If there are ways of doing away with that, none are offered in this book. To quote Poincaré again (1958, p. 193):
Logic remains barren unless it is fertilized by intuition.

## 4. Set theory

The first of the theories considered is von Neumann-Bernays-Gödel set theory. The author offers his own modifications of a published clausal version of Gödel's axioms and says (p. 26):
Considerations of machine efficiency will be important in my conversion, and thus the clauses I supply do not result from a direct clausification of Gödel's axioms. Rather, I believe that with proper definitions they are provably equivalent to his.
As elsewhere his honesty is to be commended, but in the context -- where accuracy is essential and subtle errors are easily made -- such ad hoc moves are worrying.

Perhaps the main value of the book, to a non-expert reader, is the inside view of the actuality of resolution-based automated theorem proving that is offered by passages like the following, made in connection with the set theory material (p. 52):
In my first attempt at a proof, I turn on UR-resolution, paramodulation into and from, and back demodulation. If these settings fail to obtain a proof within a reasonable length of time, I try turning on hyperresolution. I then also assign low weights to other functors that I expect must appear in the proof. This step is frequently necessary.
The discipline of adhering to general inference rules must usually be departed from by bringing in theorem-specific weightings. A more radical departure lies in the choice of the many lemmas that proof of a major theorem usually requires. For example, in one of the culminating set-theory results presented in this section -- Cantor's theorem that there is no function mapping a set onto its power set -- the essence of the diagonal argument (the definition of the Cantor class and its membership conditions) must be provided by the user and proved as lemmas prior to the theorem itself.

## 5. Number theory

The book includes a very large number of theorems of elementary number theory that have been proved semi-automatically on the basis of Peano's axioms. It must be realized that the list includes such items as the definition (pp. 192-3)

```
-> ((x+(y-x)) = max(x,y))
```

(where `y-x` is defined to be `0` if `y  x`) and the theorems

```
-> (max(0,y) = y)
```

and

```
-> (max(x,x) = x).
```

Fully-formalized arithmetic is broad as well as deep! The culminating results are two famous theorems of Euclid. The first is the irrationality of the square root of 2 (or indeed any prime number). The negation of the theorem is expressed in clausal form as (p. 81):

- ➔ `PRIME(p).`
- ➔ `-> ((p.(a.a)) = (b.b)).`

```
(gcd(a,b) = 0) -> .
```

What is actually proved is that a prime cannot be a ratio of squares of natural numbers; the reason, the author says, is that "I haven't yet developed the theory of fractions, let alone square roots." The proof takes only 25 lines. But it must be realized that the thirteen lemmas that were needed to produce it include such minor items as (p. 81)

```
((u.(x.x)) = (y.y)) -> ((u.((x/gcd(x,y)).(x/gcd(x,y)))) =
((y/gcd(x,y)).(y/gcd(x,y)))).
```

and such major items as

```
(gcd(x,y) = 1), DIV(x,(y.z)) -> DIV(x,z).
```

While it is true that these lemmas are selected by the prover from among the four hundred-odd previously proved theorems, their particularity strongly suggests how vast a gulf must be bridged to fully automate the proof.

The author speculates (p. 91) "that if a human has not proved the Goldbach conjecture within about forty years, the odds will shift in favor of a machine first finding the proof." Forty years is a good deal longer than the usual span for AI predictions, and if one will accept a proof with hundreds of lemmas, carefully chosen by the human user, following the research of human mathematicians, the prediction is not outlandish. A fully automatic proof would be a completely different matter. So far as we know the background needed for proving deep results in number theory goes far beyond what might be expected from the simplicity of the axioms and the theorem statements. There is no indication in this work that a resolution prover might in time automatically develop the theory of elliptic functions in response to a request for a proof of Fermat's last theorem.

## 6. Euclidean geometry

Euclidean geometry appears to be an ideal prospective domain for automatic proof. The pioneering work of Gelerntner (1963) (see also Gilmore 1970) served mainly to show that the appearance is deceptive. Axiomatic presentations of Euclidean geometry have not proved to be very amenable to automation. The development presented by Art Quaife confirms that observation.

He uses a complete axiomatization, due to Alfred Tarski (1951), based on equidistance and betweenness as primitive relations on points. The most difficult of the theorems proved is that the diagonals of a rectangle bisect each other. The proof invokes 23 previously proved results and takes 555 seconds on a VAX 8800 running ULTRIX 2.0. It seems like overkill for a theorem that holds for parallellograms in general, not just rectangles, and that basically comes down to the trivial vector implication

*B-A = C-D => 1/2(A+C)= 1/2(B+D).*

The idea that an algebraic approach might pay off is not mistaken. Quaife briefly mentions the work done by Shang-Ching Chou (1988) at the University of Texas during the nineteen-eighties. Chou implemented an algorithm of the eminent Chinese mathematician Wu Wen-Tsun (1986) and was able to efficiently generate fully automatic proofs of hundreds of theorems of Euclidean geometry, including theorems that humans find difficult to prove such as Simson's theorem. Lemmas are not required. The significance of this tour de force has not adequately been recognized. If runs on the board are what counts, it is *the* major achievement to date in automated theorem proving. It suggests the possibility that an entirely different approach to automatic reasoning -- in which deep, domain-specific, computational knowledge reduces the need to search, or does away with it entirely -- may pay off in areas other than geometry.

## 7. Logic

It is not surprising that the main successes of Quaife's prover and others like it have been in proving theorems of logic. Here if anywhere the method is suited to the matter. In the final chapter, semiautomatic proofs are offered of Löb's theorem and of Gödel's first and second incompleteness theorems. This is achieved within a formalization of the metatheory of the modal logic K4 -- it is not theorems in logic that are proved, but theorems about logical theorems -- with heavy use of demodulation. As with the other chapters there is a nice brief presentation of the relevant background.

The final hundred pages or so of the book are devoted to an edited list of the theorems proved in NBG set theory and Peano arithmetic, material that should be useful to other workers in the area. For them the book may well be as valuable as its extraordinary price suggests.

## 8. Conclusion

Art Quaife writes clearly, honestly and with infectious enthusiasm. His interesting book is a compilation of research papers which retains the excitement and focus on detail that his creative use of OTTER engendered. I can't help liking a scholarly book that carries the ancient Greek or southern Californian dedication:

To the pursuit and realization of unlimited pleasure.

Perhaps the book is mis-named. It contains no automated development of theories. On the contrary the development of the theories that are presented is carried out entirely by the author, with a combination of skill, experience, trial and error and, above all, knowledge. Little explanation is offered of empirical facts that might play a role in the further automation of the proof process, such as that hyper-resolution is preferred to UR-resolution (with the latter apparently mainly switched off) for the Tarski geometry whereas the reverse is the case for Peano arithmetic.

Before sceptics will grant that what is going on here is truly automatic theorem proving it will be necessary to automate the control that currently must be exercised by the user of a resolution-based theorem prover. One may doubt whether that is feasible. Application of the cut rule, the rule of classical logic that allows the use of lemmas, is the main problem; it is here that knowledge is brought to bear which the program has no access to. To circumvent the problem, a different line of attack might pay off.

The hard bits are easy and the easy bits are hard. That is a view that some AI people have. It means something like this: it's easy to prove Gödel's theorems but hard to recognize your mother. For a machine, that is. A look at mainstream automated theorem-proving shows that matters are not so simple. The hard bits are hard too. The real work in proving a deep theorem lies in the development of the theory that it belongs to and its relationships to other theories, the design of definitions and axioms, the selection of good inference rules, and the recognition and proof of more basic theorems.

Currently, no resolution-based program, when faced with the stark problem of proving a hard theorem, can do all this. That is not surprising. No person can either. Remarks about standing on the shoulders of giants are not just false modesty. Great theorems require great theories and theories do not, it seems, emerge from thin air. Their creation requires sweat, knowledge, imagination, genius, collaboration and time. As yet there is not much serious collaboration of machines with one another, and we are only just beginning to see real symbiosis between people and machines in the exercise of rationality.

## 9. Afterword: the role of understanding

Quaife's experience seems to confirm Penrose's beliefs concerning understanding. What the program OTTER lacks is understanding. It has no conception of which results are important, let alone which are interesting. It lacks the ability, surely a component of human approaches to proof, to reject purported theorems that are plainly false on semantic grounds (such as the use of symmetry or of diagrams in the case of geometry). Art Quaife seems not to feel a need for such ingredients, but others in automated theorem proving do. They face very difficult problems of implementation, especially if they abhor ad hoc methods.

But that is not the end of the story, or even of the story so far. Think about Chou's program, nicknamed the China prover. It will certainly beat any mathematician who's not seen it before to finding a proof of a difficult geometry theorem such as Morley's theorem (which is the assertion that the trisectors of the angles of any triangle meet at the vertices of an equilateral triangle). Does the China Prover understand? Many would say no. A less glib answer might be: it understands enough to prove the theorems. The point is that a certain kind of understanding, admittedly quite different to the kind that you and I have, is embedded in the algorithm that the prover uses. Unlike OTTER's method, the algorithm has deep mathematical content. Humans, of course, are responsible for the algorithm and the associated theory and proofs of correctness. Most prominent among them is David Hilbert. That's a nice touch on the part of whoever is writing the script for all this.

It would be easy, but a mistake on several grounds, to dismiss the success of the China Prover because it is in the quaint field of Euclidean geometry. For many centuries Euclidean geometry was regarded (rightly) as one of the peaks of human intellectual achievement, and it remains the geometry of the world in which robots and most people operate. Moreover, essential features of the geometry theorem proving algorithm do carry over to other domains.

On the question of understanding there is one further point that is quite telling, I think. It was observed by Chou as he ran his prover that many long-accepted results of Euclidean geometry were in fact subtly incorrect. Humans tend not to notice tricky but common special cases in which geometry theorems break down. It is the mistake we observe in undergraduates who forget that x must not be zero when they divide by it. The China prover does not need to be given such degeneracy conditions: it actually generates them. Who understands better, then, man or machine?

## Notes

A very brief version of this article appeared in the *Australian Computer Journal*, February 1995.

## References

Boyer, R., Lusk, E., McCune, W., Overbeek, R., Stickel, M. and Wos, L. (1986) Set theory in first-order logic: Clauses for Gödel's axioms. *Journal of Automated Reasoning 2*, 287-327.

Chou, S. (1988) *Mechanical geometry theorem proving.* Reidel.

Gelerntner, H. (1963) Realization of a geometry theorem-proving machine. In Feigenbaum and Feldman (eds.) *Computers and thought*, 134-152. McGraw-Hill.

Gilmore, P. C. (1970) An examination of the geometry theorem-proving machine. *Artificial Intelligence 1*, 171-187.

Lusk, E. and Overbeek, R. (1982) Experiments with resolution-based theorem-proving algorithms. *Computers and Mathematics with Applications* 8, 141-152 (1982).

McCune, W. (1990) *Otter 2.0 user's guide.* ANL-90/9, Argonne National Laboratory.

Poincaré, H. (1958) *Science and method*. Dover.

Robinson, J. A. (1963) A machine oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery* 10, 163-174.

Tarski, A. (1951) *A decision method for elementary algebra and elementary geometry*. University of California Press.

Wen-Tsun, Wu. (1986) Basic principles of mechanical theorem proving in geometries. *Journal of the Automated Reasoning 2*, 221-252.

Wos, L., Robinson, G. and Carson, D. (1965) Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the Association for Computing Machinery 12*, 536-541.

Wos, L., Robinson, G., Carson, D. and Shalla, L. (1967) The concept of demodulation in theorem proving. *Journal of the Association for Computing Machinery 14*, 698-709.