Automated Reasoning

John Harrison

University of Cambridge

- What is automated reasoning? Automatic vs. interactive.
- Successes of the AI and logic approaches
- Development of formal logic
- History of automated reasoning
- Verification
- Current research topics

What is automated reasoning?

In one sense we'll interpret our title narrowly: we are interested in reasoning in logic and mathematics, rather than everyday life. The field is also called *automated theorem proving*.

In another sense we interpret it broadly: we don't just consider making computers prove theorems automatically, but also ways in which they can support humans. The correct title might be *mechanized theorem proving*.

We'll divide the discussion into (fully) *automatic* systems, and *interactive* systems.

The limits of automated reasoning

It's almost certainly impossible, even in principle, that a computer can prove automatically all the mathematical theorems we are interested in. This follows from *Tarski's theorem on the undefinability of truth* (1936), which implies that the set of true facts of arithmetic is not even *semicomputable*.

However we can set up logical systems that are capable of deducing many, perhaps most, interesting theorems, such that the set of logically valid formulas is at least *semicomputable*. For example, the system of *Zermelo-Fraenkel set theory* based on *first order logic* has this property. But this doesn't include *all* true facts (Gödel 1930, also a corollary to Tarski's theorem). And it is still not *computable*.

Decidable systems

In fact, for some limited areas of mathematics or logic, there are systems for which validity is actually *computable*. A simple example is propositional logic. We can decide if

$$\neg(p \vee q) \Rightarrow \neg p \wedge \neg q$$

is valid simply by considering cases, e.g. writing a truth table. For a more interesting example, the first order theory of reals with multiplication is decidable (Tarski 1948). This theory includes many nontrivial problems.

In general, note that a system that is *complete* and *semicomputable* is also *computable*. (This follows from a classic theorem in computability theory that if a set and its complement are both RE, the set is recursive.)

Down to earth

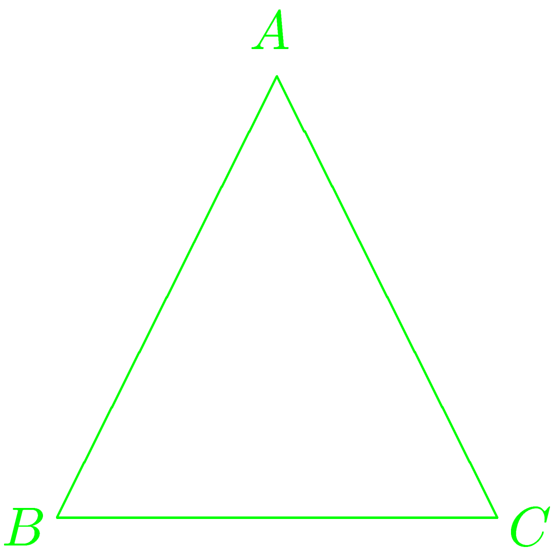
Despite these promising facts, two similar problems remain.

1. Even if a theory is decidable in principle, the time or space usage of the decision procedure may make it ineffective in practice. This applies to the first order theory of reals, for example.
2. In systems where validity is *semicomputable*, we just have to keep searching until we find the theorem. This is also impractical in many cases; typically, we use ingenious tricks to cut down the search space. The tricks are usually drawn either from looking at *human behaviour* or considering *theorems from logicians.*

There was (is?) still a controversy over whether the human-oriented 'AI' approach or the 'logic' approach is better.

A theorem in geometry

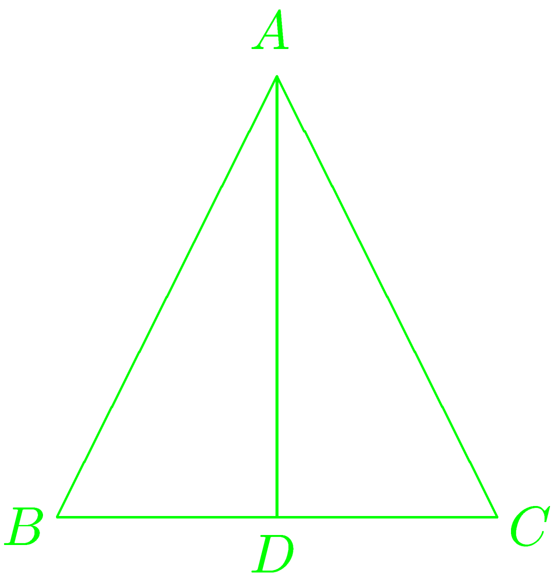
One of the early successes in automated theorem proving (the AI side) was the proof of the following theorem:



If the sides AB and AC are equal (i.e. the triangle is isoseles), then the angles ABC and ACB are equal.

The usual proof

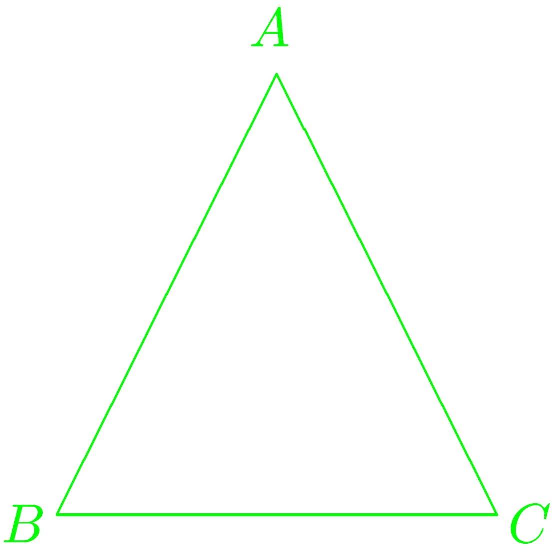
The usual proof proceeds by dropping a perpendicular down from the point A to the side BC, meeting it at a point D:



and then using the fact that the triangles ABD and ACD are congruent.

The computer's proof

The computer found an ingenious proof which had been missed by most writers on geometry (though it had already been used by Pappus).



Simply, the triangles ABC and ACB are congruent. Q.E.D.

The Robbins Conjecture (1)

A very recent success in automated reasoning, this time on the logic side, was the proof by McCune's program EQP of the Robbins Conjecture.

Huntington (1933) presented the following basis for Boolean algebra:

$$\begin{aligned} x + y &= y + x \\ (x + y) + z &= x + (y + z) \\ n(n(x) + y) + n(n(x) + n(y)) &= x \end{aligned}$$

Shortly thereafter, Herbert Robbins conjectured that the Huntington equation can be replaced with a simpler one:

$$n(n(x + y) + n(x + n(y))) = x$$

The Robbins Conjecture (2)

This conjecture went unproved for more that 50 years, despite being studied by many mathematicians, even including Tarski.

It because a popular target for researchers in automated reasoning.

In May 1996, it was claimed that a proof had been found automatically using the REVEAL prover. However this was traced to a bug in REVEAL.

The in October 1996 a correct proof was found by McCune's program EQP.

The successful search took about 8 days on an RS/6000 processor and used about 30 megabytes of memory.

Origins of mechanization

The idea of mechanizing reasoning in a manner similar to arithmetic calculation is an old one, going back at least to Hobbes.

> Reason [. . .] is nothing but Reckoning. For as Arithmeticians teach to adde and subtract in *numbers* [...] The Logicians teach the same in consequences of words [...] And as in Arithmetique, unpractised men must, and Professors themselves may often erre, and cast up false; so also in any other subject of Reasoning the ablest, most attentive, and most practised men, may deceive themselves, and inferre false conclusions.

Leibniz envisaged a *calculus ratiocinator.* First however we need a *characteristica universalis.*

Development of formal logic

We can highlight several important phases in the development of formal logic.

- The Socratic method
- Aristotle's syllogisms
- Leibniz's attempts at a *characteristica*
- Boole's algebra of logic
- Frege's *Begriffsschrift*
- Peano's *Formulaire*
- Russell and Whitehead's *Principia Mathematica.*
- Hilbert's programme
- Metamathematical studies (Gödel, Tarski, Church, Turing, ...)

Early computer experiments

The earliest uses of computers in theorem proving were in the late 50s and early 60s. Among the pioneers were:

- Newell and Simon (AI)
- Gelentner's geometry machine (AI)
- Gilmore (logical)
- Wang (logical)
- Prawitz (logical)

The logical approach proved successful, but soon reached its limits. Prawitz's method is quite close to modern *tableaux* provers. But more powerful methods were needed.

More recent methods

The two most efficient general first order theorem proving methods were invented in the 60s.

- **Resolution**, invented by Alan Robinson, is a bottom-up, local, proof method based on a single, very simple, inference rule:

$$\frac{p \vee q \quad \neg p}{q}$$

- **Model elimination**, invented by Donald Loveland, is a top-down, global, proof method which in many versions is quite similar to Prolog.

These are still the big two methods today, represented by `SETHEO` (from Munich) and `Otter` (from Chicago), probably the most powerful general first order provers at present.

Logical foundations

Tableaux, model elimination and resolution all rely on a number of fundamental theorems in logic, due to Gödel, Skolem, Gentzen, Herbrand and others. The most important is the 'uniformity theorem', also called the Skolem-Gödel-Herbrand theorem, which states that if:

$$\exists x_1, \ldots, x_n.\, P[x_1, \ldots, x_n]$$

is valid then there are terms such that the following is too:

$$P[t^1_1, \ldots, t^1_n] \vee \cdots \vee P[t^k_1, \ldots, t^k_n]$$

This can be proved either by semantic or syntactic means. In the latter version it is properly known as Herbrand's theorem.

Obviousness

A problem with automation is that what humans and computers find obvious are not the same. For example computers find:

$$
\begin{aligned}
&(\forall x\ y\ z.\ P(x,y) \wedge P(y,z) \Rightarrow P(x,z)) \wedge \\
&(\forall x\ y\ z.\ Q(x,y) \wedge Q(y,z) \Rightarrow Q(x,z)) \wedge \\
&(\forall x\ y.\ Q(x,y) \Rightarrow Q(y,x)) \wedge \\
&(\forall x\ y.\ P(x,y) \vee Q(x,y)) \\
&\Rightarrow (\forall x\ y.\ P(x,y)) \vee (\forall x\ y.\ Q(x,y))
\end{aligned}
$$

very obvious, but most people need to think about it. Conversely, most people find McCarthy's 'mutilated checkerboard' obvious (when shown the trick) but computers have trouble. Computers are really oriented towards 'logical' obviousness.

The Boyer-Moore Prover

Boyer and Moore's NQTHM is unusual in that it doesn't work in pure logic. Instead it uses a very simple system of 'primitive recursive arithmetic' (Skolem, Goodstein).

It has the remarkable ability to do proofs by induction automatically.

These properties make it much more useful in many real situations than provers for pure logic.

It has been used for many impressive applications, mainly in verification, which we consider later.

It is fully automatic. Nevertheless, the user still has to guide it in some way by selecting a sequence of lemmas. And there is not much control over what it does.

Interactive theorem proving

Given the limitations of automation, why not build systems to combine automation with human control and guidance? There were pioneering attempts in the SAM (semi-automated mathematics) project. Other pioneering proof checkers appeared in the 70s:

- AUTOMATH (de Bruijn)
- Mizar (Trybulec et al.)
- Stanford LCF (Milner)

However, these tended to be tedious to use. What was needed was a better mix of automation with the manual controllability.

Edinburgh LCF

One of the most important developments in theorem proving was the development of Edinburgh LCF (Milner et al.)

This provides low-level security, and at the same time, programmability.

The user can write completely arbitrary procedures in the ML programming language.

At the same time, inside the machine, everything happens by simple primitive inferences.

Many descendants including HOL (Gordon), Isabelle (Paulson), Coq (Huet et al.) and Nuprl (Constable et al.)

Formalized Mathematics

One application of theorem provers is to check large bodies of existing mathematics, making them completely formal.

Peano started such a project with his *Formulaire* but did not really formalize *proofs*.

Bourbaki seems to believe in formalization 'in principle', but not in practice.

However with the help of the computer we can actually achieve formalization.

The most impressive example is the Mizar project.

There is a recent proposal for a QED Project to extend this formalization much further.

Verification

The idea of verification is to make sure computer systems (hardware, software) work correctly by formal verification of the design. It is a more systematic approach than testing. It's important to understand exactly what this means.

1. The informal requirements
2. **Formal specification**
3. **Model of the implementation**
4. Actual implementation

We try to link levels 2 and 3. The connections between 1 and 2 and between 3 and 4 are informal, though we can try hard to make them small.

Conclusions

We can draw the following conclusions:

- Automated reasoning is one of the most interesting applications of symbolic processing. It is also a controlled testground for ideas from Artificial Intelligence.
- There is much research waiting to be done. Many problems have not been solved and there are many competing and completely different theorem provers in the world.
- The formalization of mathematics seems to be an interesting project, with particular value for education.
- It may be that theorem proving is the way to make the next generation of computer systems more reliable.

Postscript

A theorem prover in 6 lines of Prolog (Beckert and Possega):

```
prove(Fml,VarLim) :- nonvar(VarLim),!,prove(Fml,[],[],[],VarLim).
prove(Fml,Result) :-
        iterate(VarLim,1,prove(Fml,[],[],[],VarLim),Result).

prove_uv(Fml,VarLim) :- nonvar(VarLim),!,prove(Fml,[],[],[],[],[],VarLim).

prove_uv(Fml,Result) :-
        iterate(VarLim,1,prove(Fml,[],[],[],[],[],VarLim),Result).

iterate(Current,Current,Goal,Current) :- nl,
        write('Limit = '),
        write(Current),nl,
        Goal.

iterate(VarLim,Current,Goal,Result) :-
        Current1 is Current + 1,
        iterate(VarLim,Current1,Goal,Result).
```