

A short survey of automated reasoning

John Harrison

Intel Corporation, JF1-13
2111 NE 25th Avenue, Hillsboro OR 97124, USA
johnh@ichips.intel.com

Abstract. This paper surveys the field of automated reasoning, giving some historical background and outlining a few of the main current research themes. We particularly emphasize the points of contact and the contrasts with computer algebra. We finish with a discussion of the main applications so far.

1 Historical introduction

The idea of reducing reasoning to mechanical calculation is an old dream [75]. Hobbes [55] made explicit the analogy in the slogan ‘Reason [...] is nothing but Reckoning’. This parallel was developed by Leibniz, who envisaged a ‘*characteristica universalis*’ (universal language) and a ‘*calculus ratiocinator*’ (calculus of reasoning). His idea was that disputes of all kinds, not merely mathematical ones, could be settled if the parties translated their dispute into the *characteristica* and then simply calculated. Leibniz even made some steps towards realizing this lofty goal, but his work was largely forgotten.

The *characteristica universalis*

The dream of a truly universal language in Leibniz’s sense remains unrealized and probably unrealizable. But over the last few centuries a language that is at least adequate for (most) mathematics has been developed.

Boole [11] developed the first really successful symbolism for logical and set-theoretic reasoning. What’s more, he was one of the first to emphasize the possibility of applying formal calculi to several different situations, and doing calculations according to formal rules without regard to the underlying interpretation. In this way he anticipated important parts of the modern axiomatic method. However Boole’s logic was limited to propositional reasoning (plugging primitive assertions together using such logical notions as ‘and’ and ‘or’), and it was not until the much later development of *quantifiers* that formal logic was ready to be applied to general mathematics.

The introduction of formal symbols for quantifiers, in particular the *universal quantifier* ‘for all’ and the *existential quantifier* ‘there exists’, is usually credited independently to Frege, Peano and Peirce. Logic was further refined by Whitehead and Russell, who wrote out a detailed formal development of the foundations of mathematics from logical first principles in their *Principia Mathematica* [109]. In a short space of time, stimulated by Hilbert’s foundational programme (of which more below), the usual logical language as used today had been developed.

English	Symbolic	Other symbols
false	\perp	$0, F$
true	\top	$1, T$
not p	$\neg p$	$\bar{p}, -p, \sim p$
p and q	$p \wedge q$	$pq, p\&q, p \cdot q$
p or q	$p \vee q$	$p + q, p \mid q, p \text{ or } q$
p implies q	$p \Rightarrow q$	$p \rightarrow q, p \supset q$
p iff q	$p \Leftrightarrow q$	$p = q, p \equiv q, p \sim q$
for all x, p	$\forall x. p$	$(x)p$
there exists x such that p	$\exists x. p$	$(Ex)p$

At its simplest, one can regard this just as a convenient shorthand, augmenting the usual mathematical symbols with new ones for logical concepts. After all, it would seem odd nowadays to write ‘the sum of a and b ’ instead of ‘ $a + b$ ’, so why not write ‘ $p \wedge q$ ’ instead of ‘ p and q ’? However, the consequences of logical symbolism run much deeper: arriving at a precise formal syntax means that we can bring deeper logical arguments within the purview of mechanical computation.

Hilbert’s programme

At various points in history, mathematicians have become concerned over apparent problems in the accepted foundations of their subject. For example, the Pythagoreans tried to base mathematics just on the rational numbers, and so were discomfited by the discovery that $\sqrt{2}$ must be irrational. Subsequently, the apparently self-contradictory treatment of infinitesimals in Newton and Leibniz’s calculus disturbed many, as later did the use of complex numbers and the discovery of non-Euclidean geometries. Later still, when the theory of infinite sets began to be pursued for its own sake and generalized, mainly by Cantor, renewed foundational worries appeared.

Hilbert [53] suggested an ingenious programme to give mathematics a reliable foundation. In the past, new and apparently problematic ideas such as complex numbers and non-Euclidean geometry had been given a foundation based on some well-understood concepts, e.g. complex numbers as points on the plane. However it hardly seems feasible to justify infinite sets in this way based on finite sets. Hilbert’s ingenious idea was to focus not on the mathematical structures themselves but on the *proofs*. Given a suitable formal language, mathematical proofs could themselves become an object of mathematical study — Hilbert called it *metamathematics*. The hope was that one might be able to show in this way that concrete conclusions reached using some controversial abstract concepts could nevertheless be shown still to be valid or even provable without them.

The calculus ratiocinator

Gödel’s famous incompleteness theorems [40, 98, 37] show that formal systems for deducing mathematics have essential weaknesses. For example, his first theorem is that any given formal system of deduction satisfying a few natural conditions is incomplete

in the sense that some formally expressible and true statement is not formally provable. It is generally agreed that Gödel's results rule out the possibility of realizing Hilbert's programme as originally envisaged, though this is a subtle question [65, 96]. What is certainly true is that Gödel's theorem was the first of a variety of 'impossibility' results that only really become possible when the notion of mathematical proof is formalized.

Inspired by techniques used in Gödel's incompleteness results, Church [23] and Turing [106] proposed definitions of 'mechanical computability' and showed that one famous logical decision question, Hilbert's *Entscheidungsproblem* (decision problem for first-order logic) was unsolvable according to their definitions. Although this showed the limits of mechanical calculation, Turing machines in particular were an important inspiration for the development of real computers. And before long people began to investigate actually using computers to formalize mathematical proofs.

In the light of the various incompleteness and undecidability results, there are essential limits to what can be accomplished by automated reasoning. However, Gödel's results apply to human reasoning too from any specific set of axioms, and in principle most present-day mathematics can be expressed in terms of sets and proven from the axioms of Zermelo-Fraenkel set theory (ZF). Given any conventional set of mathematical axioms, e.g. a finite set, or one described by a finite set of schemas, such as ZF, there is at least a *semi-decision* procedure that can in principle verify any logical consequence of those axioms. Moreover many suitably restricted logical problems *are* decidable. For example, perhaps the very first computer theorem prover [29] could prove formulas involving quantifiers over natural numbers, but with a linearity restriction ensuring decidability [85].

2 Theorem provers and computer algebra systems

Before we proceed to survey the state of automated reasoning, it's instructive to consider the similarities and contrasts with computer algebra, which is already an established tool in biology as in many other fields of science. In some sense theorem provers (TPs) and computer algebra systems (CASs) are similar: both are computer programs to help people with formal symbolic manipulations. Yet there is at present surprisingly little common ground between them, either as regards the internal workings of the systems themselves or their respective communities of implementors and users. A theorem prover might be distinguished by a few features, which we consider in the following sections.

Logical expressiveness

The typical computer algebra system supports a rather limited style of interaction [27]. The user types in an expression E ; the CAS cogitates, usually not for very long, before returning another expression E' . The implication is that we should accept the theorem $E = E'$. Occasionally some slightly more sophisticated data may be returned, such as a set of possible expressions E'_1, \dots, E'_n with corresponding conditions on validity, e.g.

$$\sqrt{x^2} = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x \leq 0 \end{cases}$$

However, the simple equational style of interaction is by far the most usual. By contrast, theorem provers have the logical language available to express far more sophisticated mathematical concepts such as the $\epsilon - \delta$ definition of continuity:

$$\forall x \in \mathbb{R}. \forall \epsilon > 0. \exists \delta > 0. \forall x'. |x - x'| < \delta \Rightarrow |f(x) - f(x')| < \epsilon$$

In particular, the use of a full logical language with quantifiers often unifies and generalizes existing known concepts from various branches of mathematics. For instance, the various algorithms for quantifier elimination in real-closed fields starting with Tarski's work [103] can be considered a natural and far-reaching generalization of Sturm's algorithm for counting the number of real roots of a polynomial. At the same time, quantifier elimination is another potentially fruitful way of viewing the notion of projection in Euclidean space. Chevalley's constructibility theorem in algebraic geometry 'the projection of a constructible set is constructible', and even some of its generalizations [45], are really just quantifier elimination in another guise.

Clear semantics

The underlying semantics of expressions in a computer algebra system is often unclear, though some are more explicit than others. For example, the polynomial expression $x^2 + 2x + 1$ can be read in several ways: as a member of the polynomial ring $\mathbb{R}[x]$ (not to mention $\mathbb{Z}[x]$ or $\mathbb{C}[x]$...), as the associated function $\mathbb{R} \rightarrow \mathbb{R}$, or as the value of that expression for some particular $x \in \mathbb{R}$. Similarly, there may be ambiguity over which branch of various complex functions such as square root, logarithm and power is considered, and it may not really be clear in what sense 'integral' is meant to be understood. (Riemann? Lebesgue? Just antiderivative?) Such ambiguities are particularly insidious since in many situations it doesn't matter which interpretation is chosen (we have $x^2 + 2x + 1 = (x + 1)^2$ for any of the interpretations mentioned above), but there are situations where the distinction matters.

By contrast, theorem provers usually start from a strict and precisely defined logical foundation and build up other mathematical concepts by a sequence of definitions. For example, the HOL system [42] starts with a few very basic axioms for higher-order logic and a couple of set-theoretic axioms, and these are given a rather precise semantics in the documentation. From that foundation, other concepts such as natural numbers, lists and real and complex numbers are systematically built up without any new axioms.

Logical rigour

Even when a CAS can be relied upon to give a result that admits a precise mathematical interpretation, that doesn't mean that its answers are always right. With a bit of effort, it's not very hard to get incorrect answers out of any mainstream computer algebra system. Particularly troublesome are simplifications involving functions with complex branch cuts. It's almost irresistible to apply simplifications such as $\log(xy) = \log(x) + \log(y)$ and $\sqrt{x^2} = x$, and many CASs will do this kind of thing freely. Although systematic approaches to keeping track of branch cuts are possible, most mainstream

systems don't use them. For example, using the concept of 'unwinding number' $u(z)$ [28], we can express rigorously simplification rules such as:

$$w \neq 0 \wedge z \neq 0 \Rightarrow \log(wz) = \log(w) + \log(z) - 2\pi i u(\log(w) + \log(z))$$

Most users probably find such pedantic details as branch cut identification a positively unwelcome distraction. They often know (or at least think they know) that the obvious simplifications are valid. In any case, if a CAS lacks the expressiveness to produce a result that distinguishes possible cases, it is confronted with the unpalatable choice of doing something that isn't strictly correct or doing nothing. Many users would prefer the former.

By contrast, most theorem provers take considerable care that all alleged 'theorems' are deduced in a rigorous way, and all conditions made explicit. Indeed, many such as HOL actually construct a complete proof using a very simple kernel of primitive inference rules. Although nothing is ever completely certain, a theorem in such a system is very likely to be correct.

What's wrong with theorem provers?

So far, we have noted several flaws of the typical computer algebra systems and the ways in which theorem provers are better. However, on the other side of the coin, CASs are normally easier to use and much more efficient. Moreover, CASs implement many algorithms useful for solving real concrete problems in applied (and even pure) mathematics, e.g. factoring polynomials and finding integrals. By contrast, theorem provers emphasize proof search in logical systems, and it's often non-trivial to express high-level mathematics in them. Thus, it is not surprising that CASs are more or less mainstream tools in various fields, whereas interest in theorem provers is mainly confined to logicians and computer scientist interested in formal correctness proofs for hardware, software and protocols and the formalization of mathematics.

Since the strengths and weaknesses of theorem provers and CASs are almost perfectly complementary, a natural idea is to somehow get the best of both worlds. One promising idea [50] is to use the CAS as an 'oracle' to compute results that can then be rigorously *checked* in the theorem prover. This only works for problems where checking a result is considerably easier than deriving it, but this does take in many important applications such as factoring (check by multiplying) and indefinite integration in the sense of antiderivatives (check by differentiating).

3 Research in automated reasoning

We can consider various ways of classifying research in automated reasoning, and perhaps some contrasts will throw particular themes into sharp relief.

AI versus logic-oriented

Some researchers have attacked the problem of automated theorem proving by attempting to emulate the way humans reason. Crudely we can categorize this as the ‘Artificial Intelligence’ (AI) approach. For example in the 1950s Newell and Simon [81] designed a program that could prove many of the simple logic theorems in *Principia Mathematica* [109], while Gelerntner [38] designed a prover that could prove facts in Euclidean geometry using human-style diagrams to direct or restrict the proofs. A quite different approach was taken by other pioneers such as Gilmore [39], Davis and Putnam [31], and Prawitz [84]. They attempted to implement proof search algorithms inspired by results from logic (e.g. the completeness of Gentzen’s cut-free sequent calculus), often quite remote from the way humans prove theorem.

Early indications were that machine-oriented methods performed much better. As Wang [107] remarked when presenting his simple systematic program for the AE fragment of first order logic that was dramatically more effective than Newell and Simon’s:

The writer [...] cannot help feeling, all the same, that the comparison reveals a fundamental inadequacy in their approach. There is no need to kill a chicken with a butcher’s knife. Yet the net impression is that Newell-Shore-Simon failed even to kill the chicken with their butcher’s knife.

Indeed, in the next few decades, far more attention was paid to systematic machine-oriented algorithms. Wos, one of the most successful practitioners of automated reasoning, attributes the success of his research group in no small measure to the fact that they play to a computer’s strengths instead of attempting to emulate human thought [111].

Today, there is still a preponderance of research on the machine-oriented side, but there have been notable results based on human-oriented approaches. For example Bledsoe attempted to formalize methods often used by humans for proving theorems about limits in analysis [10]. Bledsoe’s student Boyer together with Moore developed the remarkable NQTHM prover [13] which can often perform automatic generalization of arithmetic theorems and prove the generalizations by induction. The success of NQTHM, and the contrasting difficulty of fitting its methods into a simple conceptual framework, has led Bundy [20] to reconstruct its methods in a general science of reasoning based on *proof planning*. Depending on one’s point of view, one can regard the considerable interest in proof planning as representing a success of the AI approach, or the attempt to present aspects of human intelligence in a more machine-oriented style.

Automated vs. interactive

Thanks to the development of effective algorithms, some of which we consider later, automated theorem provers have become quite powerful and have achieved notable successes. Perhaps the most famous case is McCune’s solution [76], using the automated theorem prover EQP, of the longstanding ‘Robbins conjecture’ concerning the axiomatization of Boolean algebra, which had resisted human mathematicians for some time. This success is just one particularly well-known case where the Argonne team has used Otter and other automated reasoning programs to answer open questions. Some more can be found in the monograph [77].

However, it seems at present that neither a systematic algorithmic approach nor a heuristic human-oriented approach is capable of proving a wide range of difficult mathematical theorems automatically. Besides, one might object that even if it were possible, it is hardly desirable to automate proofs that humans are incapable of developing themselves [35]:

[...] I consider mathematical proofs as a reflection of my understanding and 'understanding' is something we cannot delegate, either to another person or to a machine.

A more easily attained goal, and if one agrees with the sentiments expressed in that quote perhaps a more desirable one, is to create a system that can verify a proof found by a human, or assist in a more limited capacity under human guidance. At the very least the computer should act as a humble clerical assistant checking the correctness of the proof, guarding against typical human errors such as implicit assumptions and forgotten special cases. At best the computer might help the process substantially by automating certain parts of the proof. After all, proofs often contain parts that are just routine verifications or are amenable to automation, such as algebraic identities. This idea of a machine and human working together to prove theorems from sketches was already envisaged by Wang [107]:

[...] the writer believes that perhaps machines may more quickly become of practical use in mathematical research, not by proving new theorems, but by formalizing and checking outlines of proofs, say, from textbooks to detailed formalizations more rigorous than *Principia* [Mathematica], from technical papers to textbooks, or from abstracts to technical papers.

The idea of a proof assistant began to attract particular attention in the late 1960s, perhaps because the abilities of fully automated systems were apparently starting to plateau. Many proof assistants were based on a batch model, the machine checking in one operation the correctness of a proof sketch supplied by a human. But a group at the Applied Logic Corporation who developed a sequence of theorem provers in the SAM (Semi-Automated Mathematics) family made their provers interactive, so that the mathematician could work on formalizing a proof with machine assistance. As they put it [46]:

Semi-automated mathematics is an approach to theorem-proving which seeks to combine automatic logic routines with ordinary proof procedures in such a manner that the resulting procedure is both efficient and subject to human intervention in the form of control and guidance. Because it makes the mathematician an essential factor in the quest to establish theorems, this approach is a departure from the usual theorem-proving attempts in which the computer *unaided* seeks to establish proofs.

In 1966, the fifth in the series of systems, SAM V, was used to construct a proof of a hitherto unproven conjecture in lattice theory [19]. This was certainly a success for the semi-automated approach because the computer automatically proved a result

now called “SAM’s Lemma” and the mathematician recognized that it easily yielded a proof of the open conjecture. Not long after the SAM project, the AUTOMATH [32, 33], Mizar [104, 105] and LCF [43] proof checkers appeared, and each of them in its way has been profoundly influential. Many of the most successful interactive theorem provers around today are directly descended from one of these.

Nowadays there is active and vital research activity in both ‘automated’ and ‘interactive’ provers. Automated provers for first-order logic compete against each other in annual competitions on collections of test problems such as TPTP [102], and the Vampire system has usually come out on top for the last few years. There is also intense interest in special provers for other branches of logic, e.g. ‘SAT’ (satisfiability of purely propositional formulas), which has an amazing range of practical applications. More recently a generalization known as ‘SMT’ (satisfiability modulo theories), which uses techniques for combining deduction in certain theories [80, 93], has attracted considerable interest. Meanwhile, interactive provers develop better user interfaces and proof languages [48], incorporate ideas from automated provers and even link to them [58], and develop ever more extensive libraries of formalized mathematics. For a nice survey of some of the major interactive systems, showing a proof of the irrationality of $\sqrt{2}$ in each as an example, see [110].

Proof search vs. special algorithms

Right from the beginning of theorem proving, some provers were customized for a particular theory or fragment of logic (such as Davis’s prover for linear arithmetic [29]), while others performed general proof search in first-order logic from a set of axioms. The explicit introduction of unification as part of Robinson’s resolution method [88] made it possible for the machine to instantiate variables in an entirely algorithmic way which nevertheless has an almost “intelligent” ability to focus on relevant terms. This gave a considerable impetus to general first-order proof search, and for a long time special algorithms were subordinated to resolution or similar principles rather than being developed in themselves. There are numerous different algorithms for general proof search, such as tableaux [7, 54], model elimination [70] as well as resolution [88] and its numerous refinements [64, 71, 72, 34, 87, 97]. Despite the general emphasis on pure first-order logic, there has also been research in automating higher-order logic [1], which allows quantification over sets and functions as part of the logic rather than via additional axioms.

However, there have been some successes for more specialized algorithms. In particular, there has always been strong interest in effective algorithms for purely equational reasoning. Knuth-Bendix completion [63] led to a great deal of fruitful research [3, 4, 56]. Automated proof of geometry problems using purely algebraic methods has also attracted much interest. The first striking success was by Wu [108] using his special triangulation algorithm, and others have further refined and applied this approach [22] as well as trying other methods such as resultants and Gröbner bases [61, 89]. Incidentally Gröbner bases [16, 17] are more usually considered a part of computer algebra, but as a tool for testing ideal membership they give a powerful algorithm for solving various logical decision problems [95, 60].

4 Applications of automated reasoning

At present there are two main applications of automated reasoning.

Formal verification

One promising application of formalization, and a particularly easy one to defend on utilitarian grounds, is to verify the correct behaviour of computer systems, e.g. hardware, software, protocols and their combinations. We might wish to prove that a sorting algorithm really does always sort its input list, that a numerical algorithm does return a result accurate to within a specified error bound, that a server will under certain assumptions always respond to a request, etc.

In typical programming practice, programs are usually designed with clear logical ideas behind them, but the final properties are often claimed on the basis of intuitive understanding together with testing on a variety of inputs. As programmers know through bitter personal experience, it can be very difficult to write a program that always performs its intended function. Most large programs contain ‘bugs’, i.e. in certain situations they do not behave as intended. And the inadequacy of even highly intelligent forms of testing for showing that programs are bug-free is widely recognized. There are after all usually far too many combinations of possibilities to exercise more than a tiny fraction. The idea of rigorously *proving* correctness is attractive, but given the difficulty of getting the formal proof right, one might wish to check the proof by machine rather than by hand.

Formal verification first attracted interest in the 1970s as a response to the perceived “software crisis”, the fundamental difficulty of writing correct programs and delivering them on time, as well as interest in computer security; see [74] for a good discussion. But over the last couple of decades there has been increased interest in formal verification in the *hardware* domain. This is partly because hardware is usually a more amenable target for highly automated techniques. Such techniques include SAT (propositional satisfiability testing), using new algorithms or high-quality implementations of old ones [14, 100, 92, 79, 41], sophisticated forms of symbolic simulation [15, 91], and temporal logic model checking [24, 86, 25]. Also, hardware verification is particularly attractive because fixing errors is often invasive and potentially expensive. For example, in response to an error in the FDIV (floating-point division) instruction of some early Intel® Pentium® processors in 1994 [83], Intel set aside approximately \$475M to cover costs.

Since the 1980s there has been extensive research in formal verification of microprocessor designs using traditional theorem proving techniques [57, 26, 44, 59, 99]. Generally there has been more emphasis on the highly automated techniques like model checking that lie somewhat apart from the automated reasoning mainstream. However, recently there has been something of a convergence, as interest in SMT (satisfiability modulo theories) leads to the incorporation of various theorem-proving methods into highly automated tools. There has also been renewed interest in applications to software, particularly partial verification or sophisticated static checking rather than complete functional verification [5]. And for certain applications, especially implementations of mathematically sophisticated algorithms, more general and interactive theorem

proving is needed. A particularly popular and successful target is the verification of floating-point algorithms [78, 90, 82, 49].

The formalization of mathematics

The *formalizability in principle* of mathematical proof is widely accepted among professional mathematicians as the final arbiter of correctness. Bourbaki [12] clearly says that ‘the correctness of a mathematical text is verified by comparing it, more or less explicitly, with the rules of a formalized language’, while Mac Lane [73] is also quite explicit (p377):

As to precision, we have now stated an absolute standard of rigor: A Mathematical proof is rigorous when it is (or could be) written out in the first-order predicate language $L(\in)$ as a sequence of inferences from the axioms ZFC, each inference made according to one of the stated rules. [...] When a proof is in doubt, its repair is usually just a partial approximation to the fully formal version.

However, before the advent of computers, the idea of actually formalizing proofs had seemed quite out of the question. (Even the painstaking volumes of proofs in *Principia Mathematica* are for extremely elementary results compared with even classical real analysis, let alone mathematics at the research level.) But computerization can offer the possibility of *actually* formalizing mathematics and its proofs. Apart from the sheer intellectual interest of doing so, it may lead to a real increase in reliability. Mathematical proofs are subjected to peer review before publication, but there are plenty of well-documented cases where published results turned out to be faulty. A notable example is the purported proof of the 4-colour theorem by Kempe [62], the flaw only being noticed a decade later [51], and the theorem only being conclusively proved much later [2]. The errors need not be deep mathematical ones, as shown by the following [69]:

Professor Offord and I recently committed ourselves to an odd mistake (*Annals of Mathematics* (2) 49, 923, 1.5). In formulating a proof a plus sign got omitted, becoming in effect a multiplication sign. The resulting false formula got accepted as a basis for the ensuing fallacious argument. (In defence, the final result was known to be true.)

A book written 70 years ago by Lecat [68] gave 130 pages of errors made by major mathematicians up to 1900. With the abundance of theorems being published today, often emanating from writers who are not trained mathematicians, one fears that a project like Lecat’s would be practically impossible, or at least would demand a journal to itself! Moreover, many proofs, including the modern proof of the four-colour theorem [2] and the recent proof of the Kepler conjecture [47], rely on extensive computer checking and it’s not clear how to bring them within the traditional process of peer review [66].

At present we are some way from the stage where most research mathematicians can pick up one of the main automated theorem provers and start to formalize their own research work. However, substantial libraries of formalized mathematics have been built up in theorem provers, notably the mathematical library in Mizar, and a few quite

substantial results such as the Jordan Curve Theorem, the Prime Number Theorem and the Four-Colour Theorem have been completely formalized. As mathematical libraries are further built up and interactive systems become more powerful and user-friendly, we can expect to see more mathematicians starting to use them.

5 Conclusions

Automated reasoning is already finding applications in formal verification and the formalization of mathematical proofs. At present, applications to mainstream applied mathematics are limited, and so it may be premature to seek applications in computational biology. However, theorem proving has sometimes been applied in unexpected ways. For instance, many combinatorial problems are solved better by translating to SAT than by customized algorithms! Perhaps this short survey will lead some readers to find applications of automated reasoning in the biological sciences. In any case, we hope it has given some flavour of this vital and fascinating research field.

References

1. P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16:321–353, 1996.
2. K. Appel and W. Haken. Every planar map is four colorable. *Bulletin of the American Mathematical Society*, 82:711–712, 1976.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
4. L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures. Volume 2: Rewriting Techniques*, pages 1–30. Academic Press, 1989.
5. T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. Rajamani, and A. Ustuner. Thorough static analysis of device drivers. In *Proceedings of EuroSys'06, the European Systems Conference*, 2006.
6. P. Benacerraf and H. Putnam. *Philosophy of mathematics: selected readings*. Cambridge University Press, 2nd edition, 1983.
7. E. W. Beth. Semantic entailment and formal derivability. *Mededelingen der Koninklijke Nederlandse Akademie van Wetenschappen, new series*, 18:309–342, 1955.
8. N. L. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory 1736–1936*. Clarendon Press, 1976.
9. G. Birtwistle and P. A. Subrahmanyam, editors. *VLSI Specification, Verification and Synthesis*, volume 35 of *International Series in Engineering and Computer Science*. Kluwer, 1988.
10. W. W. Bledsoe. Some automatic proofs in analysis. In W. W. Bledsoe and D. W. Loveland, editors, *Automated Theorem Proving: After 25 Years*, volume 29 of *Contemporary Mathematics*, pages 89–118. American Mathematical Society, 1984.
11. G. Boole. The calculus of logic. *The Cambridge and Dublin Mathematical Journal*, 3:183–198, 1848.
12. N. Bourbaki. *Theory of sets*. Elements of mathematics. Addison-Wesley, 1968. Translated from French ‘Théorie des ensembles’ in the series ‘Eléments de mathématique’, originally published by Hermann in 1968.

13. R. S. Boyer and J. S. Moore. *A Computational Logic*. ACM Monograph Series. Academic Press, 1979.
14. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.
15. R. E. Bryant. A method for hardware verification based on logic simulation. *Journal of the ACM*, 38:299–328, 1991.
16. B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, Mathematisches Institut der Universität Innsbruck, 1965. English translation to appear in *Journal of Symbolic Computation*, 2006.
17. B. Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. *Aequationes Mathematicae*, 4:374–383, 1970. English translation ‘An Algorithmical Criterion for the Solvability of Algebraic Systems of Equations’ in [18], pp. 535–545.
18. B. Buchberger and F. Winkler, editors. *Gröbner Bases and Applications*, number 251 in London Mathematical Society Lecture Note Series. Cambridge University Press, 1998.
19. R. Bumcrot. On lattice complements. *Proceedings of the Glasgow Mathematical Association*, 7:22–23, 1965.
20. A. Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991.
21. B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and monographs in symbolic computation. Springer-Verlag, 1998.
22. S.-C. Chou. An introduction to Wu’s method for mechanical theorem proving in geometry. *Journal of Automated Reasoning*, 4:237–267, 1988.
23. A. Church. An unsolvable problem of elementary number-theory. *American Journal of Mathematics*, 58:345–363, 1936.
24. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Yorktown Heights, 1981. Springer-Verlag.
25. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
26. A. Cohn. A proof of correctness of the VIPER microprocessor: The first level. In Birtwistle and Subrahmanyam [9], pages 27–71.
27. R. M. Corless and D. J. Jeffrey. Well... it isn’t quite that simple. *SIGSAM Bulletin*, 26(3):2–6, August 1992.
28. R. M. Corless and D. J. Jeffrey. The unwinding number. *SIGSAM Bulletin*, 30(2):28–35, June 1996.
29. M. Davis. A computer program for Presburger’s algorithm. In *Summaries of talks presented at the Summer Institute for Symbolic Logic, Cornell University*, pages 215–233. Institute for Defense Analyses, Princeton, NJ, 1957. Reprinted in [94], pp. 41–48.
30. M. Davis, editor. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvability Problems and Computable Functions*. Raven Press, NY, 1965.
31. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
32. N. G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In Laudet et al. [67], pages 29–61.
33. N. G. de Bruijn. A survey of the project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606. Academic Press, 1980.
34. H. de Nivelle. *Ordering Refinements of Resolution*. PhD thesis, Technische Universiteit Delft, 1995.

35. E. W. Dijkstra. Formal techniques and sizeable programs (EWD563). In E. W. Dijkstra, editor, *Selected Writings on Computing: A Personal Perspective*, pages 205–214. Springer-Verlag, 1976. Paper prepared for Symposium on the Mathematical Foundations of Computing Science, Gdansk 1976.
36. E. A. Feigenbaum and J. Feldman, editors. *Computers & Thought*. AAAI Press / MIT Press, 1995.
37. T. Franzén. *Gödel's Theorem. An Incomplete Guide to its Use and Abuse*. A. K. Peters, 2005.
38. H. Gelerntner. Realization of a geometry-theorem proving machine. In *Proceedings of the International Conference on Information Processing, UNESCO House*, pages 273–282, 1959. Also appears in [94], pp. 99–117 and in [36], pp. 134–152.
39. P. C. Gilmore. A proof method for quantification theory: Its justification and realization. *IBM Journal of research and development*, 4:28–35, 1960.
40. K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931. English translation, ‘On Formally Undecidable Propositions of Principia Mathematica and Related Systems, I’, in [52], pp. 592–618 or [30], pp. 4–38.
41. E. Goldberg and Y. Novikov. BerkMin: a fast and robust Sat-solver. In C. D. Kloos and J. D. Franca, editors, *Design, Automation and Test in Europe Conference and Exhibition (DATE 2002)*, pages 142–149, Paris, France, 2002. IEEE Computer Society Press.
42. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.
43. M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.
44. B. T. Graham. *The SECD Microprocessor: A verification case study*, volume 178 of *Kluwer international series in engineering and computer science*. Kluwer Academic Publishers, 1992.
45. A. Grothendieck. *Éléments de Géométrie Algébrique IV: Étude locale de schémas et des morphismes de schémas*, volume 20 of *Publications Mathématiques*. IHES, 1964.
46. J. R. Guard, F. C. Oglesby, J. H. Bennett, and L. G. Settle. Semi-automated mathematics. *Journal of the ACM*, 16:49–62, 1969.
47. T. C. Hales. The Kepler conjecture. Available at <http://front.math.ucdavis.edu/math.MG/9811078>, 1998.
48. J. Harrison. Proof style. In E. Giménez and C. Paulin-Mohring, editors, *Types for Proofs and Programs: International Workshop TYPES'96*, volume 1512 of *Lecture Notes in Computer Science*, pages 154–172, Aussois, France, 1996. Springer-Verlag.
49. J. Harrison. Floating-point verification using theorem proving. In M. Bernardo and A. Cimatti, editors, *Formal Methods for Hardware Verification, 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006*, volume 3965 of *Lecture Notes in Computer Science*, pages 211–242, Bertinoro, Italy, 2006. Springer-Verlag.
50. J. Harrison and L. Théry. A sceptic's approach to combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.
51. P. J. Heawood. Map-colour theorem. *Quarterly Journal of Pure and Applied Mathematics*, 24:332–338, 1890. Reprinted in [8].
52. J. v. Heijenoort, editor. *From Frege to Gödel: A Source Book in Mathematical Logic 1879–1931*. Harvard University Press, 1967.
53. D. Hilbert. Die logischen Grundlagen der Mathematik. *Mathematische Annalen*, 88:151–165, 1922.
54. J. Hintikka. Form and content in quantification theory. *Acta Philosophica Fennica — Two papers on Symbolic Logic*, 8:8–55, 1955.

55. T. Hobbes. *Leviathan*. Andrew Croke, 1651.
56. G. Huet. A complete proof of correctness of the Knuth-Bendix completion procedure. *Journal of Computer and System Sciences*, 23:11–21, 1981.
57. W. A. Hunt. *FM8501: A Verified Microprocessor*. PhD thesis, University of Texas, 1985. Published by Springer-Verlag as volume 795 of the *Lecture Notes in Computer Science* series, 1994.
58. J. Hurd. Integrating Gandalf and HOL. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics: 12th International Conference, TPHOLS'99*, volume 1690 of *Lecture Notes in Computer Science*, pages 311–321, Nice, France, 1999. Springer-Verlag.
59. J. J. Joyce. Formal verification and implementation of a microprocessor. In Birtwistle and Subrahmanyam [9], pages 129–158.
60. A. Kandri-Rody, D. Kapur, and P. Narendran. An ideal-theoretic approach to word problems and unification problems over finitely presented commutative algebras. In J.-P. Jouannaud, editor, *Rewriting Techniques and Applications*, volume 202 of *Lecture Notes in Computer Science*, pages 345–364, Dijon, France, 1985. Springer-Verlag.
61. D. Kapur. Automated geometric reasoning: Dixon resultants, Gröbner bases, and characteristic sets. In D. Wang, editor, *Automated Deduction in Geometry*, volume 1360 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
62. A. B. Kempe. On the geographical problem of the four colours. *American Journal of Mathematics*, 2:193–200, 1879. Reprinted in [8].
63. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*. Pergamon Press, 1970.
64. R. A. Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2:227–260, 1971.
65. G. Kreisel. Hilbert's programme. *Dialectica*, 12:346–372, 1958. Revised version in [6].
66. C. W. H. Lam. How reliable is a computer-based proof? *The Mathematical Intelligencer*, 12:8–12, 1990.
67. M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger, editors. *Symposium on Automatic Demonstration*, volume 125 of *Lecture Notes in Mathematics*. Springer-Verlag, 1970.
68. M. Lecat. *Erreurs de Mathématiciens des origines à nos jours*. Ancne Libraire Castaigne et Libraire Ém Desbarax, Brussels, 1935.
69. J. E. Littlewood. *Littlewood's Miscellany*. Cambridge University Press, 1986. Edited by Bela Bollobas.
70. D. W. Loveland. Mechanical theorem-proving by model elimination. *Journal of the ACM*, 15:236–251, 1968.
71. D. W. Loveland. A linear format for resolution. In Laudet et al. [67], pages 147–162.
72. D. Luckham. Refinements in resolution theory. In Laudet et al. [67], pages 163–190.
73. S. Mac Lane. *Mathematics: Form and Function*. Springer-Verlag, 1986.
74. D. MacKenzie. *Mechanizing Proof: Computing, Risk and Trust*. MIT Press, 2001.
75. W. Marciszewski and R. Murawski. *Mechanization of Reasoning in a Historical Perspective*, volume 43 of *Poznań Studies in the Philosophy of the Sciences and the Humanities*. Rodopi, Amsterdam, 1995.
76. W. McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19:263–276, 1997.
77. W. McCune and R. Padmanabhan. *Automated Deduction in Equational Logic and Cubic Curves*, volume 1095 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
78. J. S. Moore, T. Lynch, and M. Kaufmann. A mechanically checked proof of the correctness of the kernel of the AMD5K86 floating-point division program. *IEEE Transactions on Computers*, 47:913–926, 1998.

79. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pages 530–535. ACM Press, 2001.
80. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1:245–257, 1979.
81. A. Newell and H. A. Simon. The logic theory machine. *IRE Transactions on Information Theory*, 2:61–79, 1956.
82. J. O’Leary, X. Zhao, R. Gerth, and C.-J. H. Seger. Formally verifying IEEE compliance of floating-point hardware. *Intel Technology Journal*, 1999-Q1:1–14, 1999. Available on the Web as http://developer.intel.com/technology/itj/q11999/articles/art_5.htm.
83. V. R. Pratt. Anatomy of the Pentium bug. In P. D. Mosses, M. Nielsen, and M. I. Schwartzbach, editors, *Proceedings of the 5th International Joint Conference on the theory and practice of software development (TAPSOFT’95)*, volume 915 of *Lecture Notes in Computer Science*, pages 97–107, Aarhus, Denmark, 1995. Springer-Verlag.
84. D. Prawitz, H. Prawitz, and N. Voghera. A mechanical proof procedure and its realization in an electronic computer. *Journal of the ACM*, 7:102–128, 1960.
85. M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu matematyków słowiańskich, Warszawa 1929*, pages 92–101, 395. Warsaw, 1930. Annotated English version by [101].
86. J. P. Queille and J. Sifakis. Specification and verification of concurrent programs in CESAR. In *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 195–220. Springer-Verlag, 1982.
87. J. A. Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.
88. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.
89. J. Robu. *Geometry Theorem Proving in the Frame of Theorema Project*. PhD thesis, RISC-Linz, 2002.
90. D. Rusinoff. A mechanically checked proof of IEEE compliance of a register-transfer-level specification of the AMD-K7 floating-point multiplication, division, and square root instructions. *LMS Journal of Computation and Mathematics*, 1:148–200, 1998. Available on the Web at <http://www.onr.com/user/russ/david/k7-div-sqrt.html>.
91. C.-J. H. Seger and R. E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods in System Design*, 6:147–189, 1995.
92. M. Sheeran and G. Stålmarck. A tutorial on Stålmarck’s proof procedure for propositional logic. In G. Gopalakrishnan and P. J. Windley, editors, *Proceedings of the Second International Conference on Formal Methods in Computer-Aided Design (FMCAD’98)*, volume 1522 of *Lecture Notes in Computer Science*, pages 82–99. Springer-Verlag, 1998.
93. R. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31:1–12, 1984.
94. J. Siekmann and G. Wrightson, editors. *Automation of Reasoning — Classical Papers on Computational Logic, Vol. I (1957-1966)*. Springer-Verlag, 1983.
95. H. Simmons. The solution of a decision problem for several classes of rings. *Pacific Journal of Mathematics*, 34:547–557, 1970.
96. S. Simpson. Partial realizations of Hilbert’s program. *Journal of Symbolic Logic*, 53:349–363, 1988.
97. J. R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14:687–697, 1967.
98. R. M. Smullyan. *Gödel’s Incompleteness Theorems*, volume 19 of *Oxford Logic Guides*. Oxford University Press, 1992.

99. M. K. Srivas and S. P. Miller. Applying formal verification to the AAMP5 microprocessor: A case study in the industrial use of formal methods. *Formal Methods in System Design*, 8:31–36, 1993.
100. G. Stålmarck and M. Säflund. Modeling and verifying systems and software in propositional logic. In B. K. Daniels, editor, *Safety of Computer Control Systems, 1990 (SAFE-COMP '90)*, pages 31–36, Gatwick, UK, 1990. Pergamon Press.
101. R. Stansifer. Presburger's article on integer arithmetic: Remarks and translation. Technical Report CORNELLCS:TR84-639, Cornell University Computer Science Department, 1984.
102. C. B. Suttner and G. Sutcliffe. The TPTP problem library. Technical Report AR-95-03, Institut für Infomatik, TU München, Germany, 1995. Also available as TR 95/6 from Dept. Computer Science, James Cook University, Australia, and on the Web.
103. A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, 1951. Previous version published as a technical report by the RAND Corporation, 1948; prepared for publication by J. C. C. McKinsey. Reprinted in [21], pp. 24–84.
104. A. Trybulec. The Mizar-QC/6000 logic information language. *ALLC Bulletin (Association for Literary and Linguistic Computing)*, 6:136–140, 1978.
105. A. Trybulec and H. A. Blair. Computer aided reasoning. In R. Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 406–412, Brooklyn, 1985. Springer-Verlag.
106. A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society (2)*, 42:230–265, 1936.
107. H. Wang. Toward mechanical mathematics. *IBM Journal of research and development*, 4:2–22, 1960.
108. W. Wen-tsun. On the decision problem and the mechanization of theorem proving in elementary geometry. *Scientia Sinica*, 21:157–179, 1978.
109. A. N. Whitehead and B. Russell. *Principia Mathematica (3 vols)*. Cambridge University Press, 1910.
110. F. Wiedijk. *The Seventeen Provers of the World*, volume 3600 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.
111. L. Wos and G. W. Pieper. *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning*. World Scientific, 1999.