The Automation of Proof: A Historical and Sociological Exploration

DONALD MACKENZIE

Best wishes,
Donald MacKenzie

This article reviews the history of the use of computers to automate mathematical proofs. It identifies three broad strands of work: automatic theorem proving where the aim is to simulate human processes of deduction; automatic theorem proving where any resemblance to how humans deduce is considered to be irrelevant; and interactive theorem proving, where the proof is directly guided by a human being. The first strand has been underpinned by commitment to the goal of artificial intelligence; practitioners of the second strand have been drawn mainly from mathematical logic; and the third strand has been rooted primarily in the verification of computer programs and hardware designs.

Traditionally, the most important relationship of the computer to mathematics has been in the automation of calculation. Since the middle of the 1950s, however, another relationship has emerged: the automation of mathematical reasoning, especially the automation of mathematical proof. Originally intertwined with the emergence of the new field of artificial intelligence, the automation of proof has also spawned two important programming languages, Prolog and ML. It has provided crucial technical underpinnings for the attempt mathematically to verify the correctness of programs and hardware designs. It is beginning to contribute to mathematical research.

The main aim of this article is to provide a history of the automation of proof that places its technical features in their wider intellectual and practical context. It builds upon the historical articles by Davis,[1] Wos and Henschen,[2] Loveland,[3] and O'Leary,[4] and complements Lolli's wide-ranging book.[5] (The latter, unfortunately, is available only in Italian; my attention was drawn to it by a referee of the first draft of this article.)

After this introduction comes a discussion of the first of the two main features of this intellectual context, that is, the development of ideas of proof in mathematics and logic prior to the advent of the digital computer, focusing on ideas that have both informed technically and framed conceptually the automation of mathematical reasoning. This section has no claim to either historical adequacy or sociological insight; it is essentially tutorial in nature, and readers confident of their understanding of this background can skip the section.

Then follows a section discussing the early automated theorem provers of the 1950s, and this brings in the second feature of intellectual context: the emergence of "artificial intelligence." The goal of much of the early automation of proof was explicitly to invent "a thinking machine" rather than a mere calculating machine, and success in the enterprise was seen as a demonstration that machines could indeed "think."

Tensions quickly became evident, however, between this "artificial intelligence" approach to the automation of proof and a different approach rooted more strongly in mathematical logic. The third section of the article shows how early work in the latter culminated in the development in 1963 of the best known of the techniques of automated theorem proof: "resolution," an explicitly machine-oriented form of inference, bearing little immediate resemblance to the processes of deduction by humans that the artificial intelligence approach had sought to simulate.

The fourth section then deals with the history of automatic theorem proving after the development of resolution. Critics of the latter found it to be both too inefficient to prove significant theorems and too irrelevant to what they saw as the proper goals of artificial intelligence. This section accordingly describes "nonresolution theorem proving," which can be seen as the inheritor of the original enterprise of the artificial intelligence pioneers. It also describes, however, the work of the main group of researchers who refused to be swayed by criticism of resolution: the group at the Argonne National Laboratory, whose theorem provers have in recent years achieved some of the most remarkable practical successes in this field, notably solutions to several open questions in mathematics and logic.

Section five introduces the most important practical context of the automation of proof: program and hardware verification in computer science. Wholly automatic theorem provers have so far generally been considered inadequate to these tasks, so this section of the article focuses on the history of some of the leading interactive theorem provers, in which direct guidance is provided by human beings.

The article's last section (the general goal of which is to elaborate the reference to "sociology" in my subtitle) begins with the differences in approach to the automation of proof discussed in the previous sections. These differences cannot be understood without taking into account

1058-6180/95/$4.00 © 1995 IEEE

differences in the disciplinary backgrounds and wider intellectual commitments and goals of those involved; they also resonate with different positions in the wider debate over the possibility of artificial intelligence. Thus there is evidence here for a "sociology of proving," a pattern of social factors associated with different styles of, and different assessments of, automated theorem proving. Furthermore, the automation of mathematical reasoning raises an additional sociological question, one not just about styles of proving, but about proof itself: What kinds of mathematical argument are taken, by whom, as constituting proofs? It is by signaling this issue of the "sociology of proof" that the article ends.

One final introductory remark is necessary. The word "exploration" in my subtitle is to be taken seriously. This article is not an attempt to review all historically significant efforts at the automation of proof. Instead, I have focused on work that seemed to me to illustrate well the wider contours of the history of the field; that a particular theorem prover is not discussed here does not mean I consider it unimportant. Nor does the discussion at the end of the article exhaust the sociologically interesting aspects of the automation of proof, and I hope to return to these elsewhere.

Proof in mathematics and formal logic

Like much else in Western culture, mathematical proof has conventionally been seen as beginning with the ancient Greeks. Modern history of mathematics has cast doubt on the uniqueness of the Greek concern to make rigorous the arguments underpinning mathematical conclusions, but this concern was undoubtedly substantial. Its epitome was the work of Euclid of Alexandria, around 280 B.C., which sought to derive geometry from five elementary, self-evident postulates, together with associated definitions and "common notions."

For two millennia, the demonstrations in Euclid's *Elements* remained unchallenged as paradigms of proof.[6] Mathematics did not, of course, stand still. By the early nineteenth century, whole fields unknown to, or little developed by, the Greeks had opened up: a vast flourishing of algebra; Descartes' analytical geometry, in which geometric entities such as lines are given expression not just in diagrams but as algebraic equations; the differential and integral calculus, with its multifold applications to physics; and many other topics.

The developers of these fields were typically concerned more with the creation of new techniques and results than with the construction of proofs of the Euclidean kind. As the nineteenth century proceeded, however, more and more attention began to be given to the foundations of the new fields of mathematics.

Simultaneously, the basic framework of Euclid's *Elements* began to be challenged. For centuries, there had been unease about the fifth of Euclid's postulates, the so-called "parallels postulate." (In the words of Heath's classic translation of the *Elements*, this states "that, if a straight line falling on two straight lines makes the interior angles on the same side less than two right angles, the two straight lines, if produced indefinitely, meet on that side on which are the angles less than two right angles."[7])

The nineteenth century saw the emergence of alternative, "non-Euclidean," geometries that rejected the parallels postulate. This dealt a considerable blow not merely to the detail of the *Elements* but to the whole program of deriving truths about the physical world by rigorous deduction from self-evident axioms. Some — notably the so-called formalists, led by the German mathematician David Hilbert (1862-1943) — began to feel that axioms should be considered to be (in principle) arbitrary creations rather than obvious truths, and that the relationship of theorems deduced from them to the physical world was a matter that lay outside of mathematics.

At around the same time (the end of nineteenth century), the nature of deduction itself also came under scrutiny. The Greeks — notably Aristotle — had again provided paradigms of valid deduction, the so-called "syllogisms." Arguments such as

Every B is an A
<u>Every C is a B</u>
Therefore, every C is an A

were held to provide a secure means of deducing a conclusion from premises, whatever the subject matter being reasoned about.

No serious attempt was made to reduce mathematical reasoning to Aristotle's syllogisms. The deductions in Euclid's *Elements*, and in other established parts of mathematics, were taken as obviously valid, without this intuition being given formal shape. Nevertheless, Gottfried Leibniz (1646-1716) sought to develop a universal scientific language and logical calculus, which he hoped would, as he wrote in 1679, make it possible to "judge immediately whether propositions presented to us are proved . . . with the guidance of symbols alone, by a sure and true analytical method."[8] His efforts had only limited practical success. George Boole (1815-64) similarly sought to turn logic into mathematical form, also with the goal of providing a means of verifying the correctness of an argument purely formally, without reference to its subject matter. Boole's algebraic representation of the "connectives" common in reasoning — AND, OR, NOT, and so on — has been of lasting significance.

The most crucial formalization of logic, however, derives from that provided by Gottlob Frege (1848-1925), especially in his 1879 *Begriffsschrift*.[9] Although the *Begriffsschrift*'s cumbersome diagrammatic notation has long since been abandoned, Frege founded what is now called the predicate calculus. (The predicate calculus is a formal system that contains logical connectives, terms [constants, variables, or functions], predicates, and quantifiers. Predicates are expressions that turn terms into propositions: The predicate "is prime" applied to the term "3" forms the proposition "3 is prime." Quantifiers ["there exists" or "for all"] indicate the "quantity" of a proposition, allowing, for example, the true proposition "there exists an integer which is prime" to be distinguished from the false proposition "all integers are prime.")

One of Frege's goals was to reconstruct mathematics logically, particularly arithmetic. The attempt to derive mathematics from basic logical principles — thus avoiding arbitrary axioms — was also extensively pursued by Alfred North Whitehead (1861-1947) and Bertrand Russell (1872-1970) in the three volumes of their *Principia Mathematica*, first published between 1910 and 1913. They started by postulating basic rules of logic, such as that

> the assertion of [a proposition] *p*, and the assertion that *p* implies [another proposition] *q*, permit the assertion of *q*.

(This is the ancient principle known as *modus ponens*.)[10] From these basic rules of logic, other logical principles were deduced as theorems, and a theory of sets or classes was developed. This then permitted Whitehead and Russell (and, similarly, Frege) to define numbers. Thus, for example, the number "two" is the class of all two-membered classes: the class of all classes $\{x,y\}$, with x not equal to y.

The attempt by Frege, Russell, and Whitehead to found mathematics upon logic alone quickly ran into difficulties. In particular, unconstrained use of notions such as a "set of sets" or "class of classes" can readily generate paradoxes. The formalists, permitting themselves not merely logical principles but substantive mathematical axioms, had much greater practical success. Yet even their approach was to encounter a profound problem.

For the formalists, a proof was, in principle (practice, as we shall see, was different), a finite sequence of formulae, where the last formula is the theorem to be proved, and each formula in the sequence is either an axiom or derived from previous formulae by rules of logical inference such as *modus ponens*. These rules were understood as formal in the sense that they operated "syntactically" on formulae considered simply as strings of symbols; their application in no way depended upon the meaning of those symbols. In that sense, formalism sought to avoid dubious appeals to human intuition.

However, if axioms were arbitrary creations rather than truths about the world, there was no a priori guarantee that they were mutually consistent. Yet this was a basic precondition of sound inference from them, since otherwise there might be proofs of both a proposition *p* and its negation, not-*p*. So proofs of the consistency of formal systems (that is to say, of sets of axioms together with rules of inference) were, for the formalists, a key goal.

It was on this issue that the formalist program hit its most celebrated difficulty: Kurt Gödel's 1931 result that any formal system rich enough to express the arithmetic of whole numbers could not be proven consistent, except in some larger system whose consistency would in turn be problematic.[11] This was a corollary of Gödel's famous "incompleteness theorem," which stated that any finite formal system *S* adequate to express arithmetic must be incomplete, that is, that it must contain a meaningful proposition *p* such that there is no proof within *S* of either *p* or its negation.[12]

While Gödel's result was generally taken as showing inherent limitations of the overall formalist program, it did not undermine the notion of proof at the heart of formalism. Quite the opposite. For one perceived virtue of the formal notion of proof was that because of its strictness of definition, it made it possible to reason mathematically about proof; indeed, it was put forward for that reason rather than as a model to which actual mathematical proofs had to conform. The formal notion of proof made possible "proof theory," of which Gödel's theorems are the most famous exemplars, but which was productive of many other results too (one of these, Herbrand's theorem, is crucial to the "resolution" approach to automated theorem proving — see the sidebar titled "Resolution").

Turing made the notion of "mechanical procedure" precise by imagining what has become known as the "Turing machine."

There were, of course, those who found formalism just too mechanical, as both Lolli and historian of mathematics Herbert Mehrtens note.[13,14] In 1908, for example, the French mathematician, physicist, and philosopher Henri Poincaré wrote, disdainfully, that following formalism "one could imagine a machine where one would put in axioms at one end while getting theorems at the other end, like that legendary machine in Chicago which pigs go into alive and come out transformed into hams and sausages."[15] But Poincaré was one of a beleaguered band of "countermodernists" who sought to preserve a role for intuition and reference in the philosophy of mathematics; the tide of the times was flowing in the opposite direction.[16]

The "mechanical" metaphor was famously crystallized in 1937 by the English mathematician Alan Turing (1912-54).[17] Turing was addressing the problem, first formulated by Hilbert, of whether mathematics is "decidable" — in other words, whether there could be an algorithm or mechanical procedure that could be applied in order to determine, in a finite number of steps, whether any arbitrary mathematical proposition is a theorem. To answer this question, Turing made the notion of "mechanical procedure" precise by imagining what has become known as the "Turing machine," a simple computer responding to symbols on a tape. Like Gödel's, Turing's result was negative: There is no algorithm or mechanical procedure that permits us to solve all mathematical problems (although "decision procedures" exist for certain particular, limited formal systems, as we shall note below).

Again, Turing's result did not undermine the formal notion of proof. Nevertheless, the triumph of the latter remained an abstract triumph. Proofs in actual mathematical practice were seldom strictly formal. While proofs

Resolution

To apply resolution, predicate-calculus formulae must first be translated into what is called "clausal form," where each formula is a "clause," that is to say, a disjunction of "literals." A "literal" is a proposition or a negated proposition; in clausal form the logical connective between literals is always OR. Quantifiers are not permitted. Universal quantification is indicated by leaving variables free, while existential quantifiers are replaced by functions known, after the Norwegian logician Thoralf Skolem, as Skolem functions. For example, instead of "for all *x* there exists *y* such that *R*(*x*,*y*)," where *R* is a predicate asserting something about *x* and *y*, we would have simply *R*(*x*, *f*(*x*)). *f* is the Skolem function we introduce to replace the existential quantifier. If, unlike in this example, an existential quantifier is not governed by a universal quantifier, we have a "Skolem constant" rather than a Skolem function.

The resolution principle is that "from ... two clauses *C* and *D*, one may infer a resolvent of *C* and *D*."[1] The underlying idea is easiest to understand — my exposition is here influenced by Bundy's 1983 text[2] — if we first assume that *D* contains the negation of a proposition found in *C*. Assume, therefore, that *C* has the form (*A* or *P*) and *D* the form (*B* or not-*P*), where *A* and *B* are the remaining literals in *C* and *D* respectively. *P* and not-*P* are known as "complementary literals." Elementary considerations of the propositional calculus then tell us that from these two "parent" clauses we can infer the "resolvent" clause (*A* or *B*). In other words,

A or *P*
<u>*B* or not-*P*</u>
A or *B*

Of course, any arbitrary pair of clauses, *C* and *D*, will not generally contain complementary literals. However, it may be possible to find a substitution to apply to *C* and *D* that creates complementary literals. That is to say, it may be possible to write *C* as (*A* or *P*) and *D* as (*B* or not-*Q*), where there is a substitution ø that makes *P*ø identical to *Q*ø. "Unification" — the key idea drawn by Robinson from Prawitz[3,4] — is an algorithm that yields the (unique) most general substitution that allows this identity to hold. We then have the rule for resolving two parent clauses ("binary resolution," as it is now known):

A or *P*
<u>*B* or not-*Q*</u>
(*A* or *B*) ø

where ø is the "most general unifier" of *P* and *Q*. It may also be possible to eliminate not just one pair of literals (as here) but several at once. That is what is called "full resolution." (See Bundy's text for details.[5])

The proof of the completeness of resolution draws ultimately upon a result from the 1930 PhD thesis by the mathematical logician Jacques Herbrand.[6] The key issue is the substitutions that need to be tried when searching for a contradiction. According to Herbrand's theorem, it is necessary to try only those involving terms in what is now called the "Herbrand universe" — the set of all variable-free terms that can be formed from the constants and functions in the axioms and conjecture (a "variable-free term" is a term that contains no variables but which may contain constants and functions). Because the Herbrand universe must always be either finite or countably infinite (that is to say, its members can be set in one-to-one relation with the natural numbers), the search, if conducted for long enough, is bound to find a contradiction if one exists.

References

1. J.A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *J. ACM*, Vol. 12, No. 1, 1965, p. 33.
2. A. Bundy, *The Computer Modelling of Mathematical Reasoning*, Academic Press, London, 1983, pp. 74-77.
3. D. Prawitz, "An Improved Proof Procedure," *Theoria*, Vol. 26, 1960.
4. A. Robinson, personal communication.
5. Bundy, *Computer Modelling of Mathematical Reasoning*, p. 76.
6. J. Herbrand, *Recherches sur la Théorie de la Démonstration*, PhD thesis, Univ. of Paris, 1930.

were typically deductions from sets of axioms, these deductions were seldom expressed as sequences of primitive inferences (that is, applications of individual rules of logical inference). Explicit appeal to particular rules such as *modus ponens* was indeed rare. While it was generally assumed that valid mathematical proofs *could* be transformed into sequences of primitive inferences, they seldom actually *were*.

Even those mathematicians most concerned with rigor, such as the group of French mathematicians writing under the pseudonym Nicholas Bourbaki, did not attempt full formal proof:

> But formalized mathematics cannot in practice be written down in full, and therefore we must have confidence in what might be called the common sense of the mathematician . . . We shall therefore very quickly abandon formalized mathematics.[18] (I owe this reference to Boyer and Moore.[19])

"Hawkish" logicians could, as a consequence, argue that even those derivations that mathematicians found most compelling were still not proofs. Thus P.H. Nidditch complained in 1957:

> In the whole literature of mathematics there is not a single valid proof in the logical sense . . . The number of original books or papers on mathematics in the course of the last 300 years is of the order of 106; in these, the number of even close approximations to

really valid proofs is of the order of 101 . . . In the relatively few places where a mathematician has seriously tried to give a valid proof, he has always overlooked at least some of the rules of inference and logical theorems of which he has made use and to which he has made no explicit reference.

To the counterargument that the "ordinary proofs" in mathematics could be made "complete if [the mathematician] was willing to take the trouble," Nidditch replied:

Since no mathematician has ever constructed a complete proof, his reputed capacity for doing so has no better status than an occult quality.[20]

Even as Nidditch was writing, however, answers to his complaint were under development; but they were machines rather than (as he clearly hoped) mathematicians inculcated with the priorities of formal logic. It is to the history of these machines that I now turn.

Inventing "a thinking machine"

As noted above, the formalization of mathematics and logic could be seen as the precursor of its mechanization. Until well into the twentieth century, however, nearly all efforts to automate mathematics were directed to mechanizing arithmetic rather than wider aspects of mathematical and logical reasoning. There were, it is true, exceptions. For example, the English philosopher and economist W.S. Jevons (1835-82) designed and commissioned the construction of a "logical piano" for automating syllogistic reasoning. Involving a keyboard and a system of rods, Jevons' machine permitted the automated deduction of conclusions from premises in simple syllogistic reasoning.[21]

Sustained efforts to automate mathematical reasoning, however, came only with the advent of digital computers. Crucially, these efforts coincided with the origins of the specialist field of "artificial intelligence." Although there was slightly earlier work by logician Martin Davis (see below), the program generally taken to be the first automated theorem prover was the "Logic Theory Machine." This was developed by artificial intelligence pioneers Allen Newell and Herbert Simon, together with J.C. "Cliff" Shaw, who was responsible for implementing their ideas on the Rand Corporation's "Johnniac" (an early digital computer modeled on the machine developed in the late 1940s by John von Neumann and colleagues at the Princeton Institute for Advanced Study). The Logic Theory Machine has been hailed as "the first intelligent computer program" and "first justification of the claim that artificial intelligence was a science."[22]

Newell and Simon first met at Rand in 1952. They shared what was then an unusual interest (at least outside of the closed world of cryptanalysis) in non-numerical applications of the digital computer. Simon, a political science PhD already well known for his work on organizational behavior and thought processes, had developed a strong interest in using computers to simulate human problem solving. Newell, too, was interested in such applications, notably in programming computers to play chess. In 1955, Newell came to join Simon in the Graduate School of Industrial Administration at the Carnegie Institute of Technology (now Carnegie Mellon University) in Pittsburgh.[23]

With the limited computer power available to them, chess appeared too difficult a problem. Simon recalls, "In the fall of '55 we decided that a chess machine was not the thing to start on — that an easier task was to build a theorem prover." Their first thought was of a system to prove

Crucially, these efforts coincided with the origins of the specialist field of "artificial intelligence."

theorems in geometry, but they were deterred by the way geometric problems were typically represented by diagrams. Formal logic, where proof is entirely a matter of the manipulation of symbols, was an attractive alternative, but there was an element of happenstance in the choice: "Since I had [Russell and Whitehead's] *Principia* on my shelf, I pulled it out and I said, 'Why don't we do this?' "[24]

By early December, Simon had the outline of a procedure for producing proofs, and he and Newell "worked hard to sharpen up the program, and put it into a form where one could consider coding it for the machine." Their overall goal remained firmly in mind throughout this detailed work. Another artificial intelligence pioneer, Edward Feigenbaum, who was then a student at the Carnegie Institute for Technology, recalls Herbert Simon coming into his class in January 1956 and telling them that "over Christmas Allen Newell and I invented a thinking machine."[25]

The Logic Theory Machine proved theorems in the propositional calculus (the part of formal logic that deals with the relationship between propositions, but, unlike the full predicate calculus, not their internal structure). Newell, Shaw, and Simon provided the Logic Theory Machine with the five basic axioms of propositional logic given in chapter one of the *Principia Mathematica*, together with three rules of inference: "substitution" ("any expression may be substituted for any variable in a theorem, provided the substitution is made throughout the theorem wherever that variable appears"), "replacement" (a logical "connective can be replaced by its definition, and vice versa, in any of its occurrences"), and "detachment" (*modus ponens*).[26] They then gave it theorems from chapter two of the *Principia* to prove.

One way of finding such proofs would be to start from the five initial axioms and systematically apply the three rules of inference to construct all possible sequences of valid logical deductions — the "British Museum Algorithm," as Newell, Shaw, and Simon called it in a

1957 article.[27] That would patently have been a poor simulation of human intelligence as well, they argued, as being hopelessly inefficient. Instead, the Logic Theory Machine was programmed to begin with the theorem to be proved and to search for a proof of it. If no immediate one-step proof could be found, a set of subgoals was generated (formulae from which the theorem could be proved in one step), and proofs of these were searched for; and so on iteratively. Like a human prover, the Logic Theory Machine was allowed to use previously proven theorems in constructing the proof of a new theorem.

The Logic Theory Machine found proofs for 38 of the 52 theorems it was presented with from the *Principia*, in particular finding a straightforward proof of theorem 2.85 (a simple propositional logic theorem) where Whitehead and Russell had given a more cumbersome — indeed, defective — proof.[28] Lord Russell was impressed, writing to Simon in November 1956:

> I am delighted to know that Principia Mathematica can now be done by machinery . . . I am quite willing to believe that everything in deductive logic can be done by machinery.[29]

The editor of the prestigious *Journal of Symbolic Logic*, however, was not won over, and the journal refused to publish an article coauthored by the Logic Theory Machine describing the proof of theorem 2.85.[30]

Despite its successes, it is clear that Simon and Newell were far from content with the Logic Theory Machine. Their goal was not to find a decision procedure or theorem-proving algorithm, but to discover human-like "heuristics": processes "that *may* solve a given problem, but offer . . . no guarantees of doing so."[31] Simon recalls:

> What we were interested in from the beginning was not simply getting it to prove theorems in logic. We were interested in having it do it in a humanoid way: not by number crunching — not by any kind of crunching — but with selective search. From the beginning our central interest was in selective search.[32]

Nevertheless, the Logic Theory Machine's search was not actually very selective. Subproblems, for example, were considered simply in the order in which they were generated, with no attempt at a heuristic "guess" as to which was most likely to yield a proof. This unselective search meant that the Logic Theory Machine could, in practice, find only very short proofs. As the number of steps in a proof increased, the amount of search required to find that proof grew enormously, soon outstripping the computing power available. Right at the start of automated theorem proving, its central problem had appeared: the "combinatorial explosion" of the necessary search space as the number of steps in a proof increases.

After the Logic Theory Machine, Newell and Simon moved away from automated theorem proof, in the first instance to work on what they called the "General Problem Solver," an ambitious attempt to extract, from human beings' accounts of the mental processes of problem solving, the necessary elements of a task-independent reasoning system.[33] The Logic Theory Machine's most immediate successor was the Geometry Machine, developed by Herbert Gelernter at the IBM Research Center in Poughkeepsie, New York, following a suggestion from Nathaniel Rochester, the center's manager of information research, another of the early pioneers of artificial intelligence. Like Newell and Simon, Gelernter and Rochester's goal was to demonstrate "intelligent behavior in machines."[34]

Crucially, Gelernter found that the expression of geometric problems in the form of diagrams — the issue that had deterred Newell and Simon — actually provided a simple, effective way of restricting the search for a proof. He and his colleagues encoded the relevant diagram numerically. As the program (implemented on an IBM 704) generated subgoals, it did not indiscriminately seek a proof of each of them, as the Logic Theory Machine had done. Instead, it ignored those subgoal expressions that were not valid in the diagram and concentrated on searching for a proof of those subgoals that were valid. The resulting heavily pruned search was remarkably successful: The Geometry Machine "found solutions to a large number of problems taken from high-school textbooks and final examinations in plane geometry."[35]

Formal logic and automated theorem proof

In both the Logic Theory and Geometry Machines, commitment to the general project of "artificial intelligence" was clear. Algorithms for finding proofs were rejected not simply on the grounds of their claimed inefficiency, but because the goal was to find heuristic procedures akin to those of human problem solving. As Simon puts it: "It would be nice to have a good theorem-proving program, but that was not our central interest. Our central interest is: how does the human mind manage to do these things?"[36]

The goals of understanding human problem solving and going on to build intelligent machines were, however, by no means universal among the early developers of automated theorem provers. In particular, most of those whose background was in mathematical logic seem to have found computers attractive not because they were potentially intelligent but because they were "persistent plodders," as the logician Hao Wang put it in 1960.[37] The overall goal of formal logic, from Leibnitz to Frege, had been a symbolic system that would guarantee the correctness of reasoning, quite independently of any understanding of the material being reasoned about. As Wang wrote, "Logicians had worked with the fiction of man as a persistent and unimaginative beast who can only follow rules blindly, and then the fiction found its incarnation in the machine."[38]

Furthermore, a key technical issue tackled by the generation of logicians who came after Frege was completeness. A formal system is "complete" if it can be shown that there is a formal proof, within it, of any true proposition that can be expressed in that system. While Gödel had (as noted above) shown that a formal system powerful enough to express arithmetic could not be complete, he had also given, in 1930, the first proof that the "first-order" predicate calculus was complete.[39] (In the "first-order" predicate calculus, quantification ranges only over variables. Quantification over functions and predicates is not permitted. For example, one is not permitted propositions of the form "for all predicates, such-and-such is true." An example of a proposition that cannot be expressed straightforwardly in first-order predicate calculus is the principle of mathematical induction — see below.)

There are various ways of formulating the first-order predicate calculus, and various proof procedures for it. These were still very much live research issues for mathematical logicians in the 1950s, and demonstrations of the completeness of particular proof procedures were being sought by logicians such as Burton Dreben, writing in 1950,[40] and W.V. Quine, writing in 1955.[41] In particular, a natural approach to proving completeness was to give an algorithm for finding the proof of a proposition (or for finding a contradiction that resulted from assuming its negation), and to show that if the proposition is true, the algorithm must terminate.[42] Not surprisingly, it struck several logicians that it might be possible to implement such an algorithm on a computer and thus produce an automated theorem prover for the first-order predicate calculus. Given that much of mathematics could be expressed in the latter, such a theorem prover could potentially be of considerable significance.

The first contributions of logicians to automated theorem proving actually to be implemented were not, however, as ambitious as that; they concentrated on automating "decision procedures" (see above) for those restricted parts of mathematics and logic where such procedures were known to exist. Thus Mojzesz Presburger had shown in 1929 the existence of a decision procedure for a restricted fragment of arithmetic, involving only the addition of integers.[43] In 1954, the mathematical logician Martin Davis implemented Presburger's decision procedure on the computer at the Princeton Institute for Advanced Study.[44,45] By 1960, Hao Wang, a Harvard and Oxford University logician, had implemented decision procedures for both the propositional calculus and a decidable fragment of the full predicate calculus.[46,47] (Turing had shown that there was no decision procedure for the full predicate calculus. The decidable fragment on which Wang focused — the AE predicate calculus — contains only propositions that can be transformed into a form in which no existential quantifier ["there exists"] governs any universal quantifier ["for all"].)

Wang's programs were vastly more efficient than the Logic Theory Machine. They proved all the theorems from the *Principia* proven by the Logic Theory Machine, using less than a minute of central processing time in total, and also proved many theorems that were beyond the latter's capacity. There was "a fundamental inadequacy" in Newell, Shaw, and Simon's approach, Wang wrote:

> There is no need to kill a chicken with a butcher's knife, yet the net impression is that Newell-Shaw-Simon failed even to kill the chicken with their butcher's knife. They do not wish to use standard algorithms such as the method of truth tables . . . [But] to argue the superiority of "heuristic" over algorithmic methods by choosing a particularly inefficient algorithm seems hardly just.[48]

He was not alone in his criticism. The logician E.W. Beth wrote in 1958:

> Newell and Simon . . . and Gelernter . . . seem to think of using those heuristic devices which are often applied by human beings in finding a proof or solving a problem. It seems not unfair to draw a comparison between such devices and certain tricks which one resorts to in mental computation. A number of these tricks may be of sufficient importance to be taken into account in designing a computer, or, perhaps, rather in programming a specific problem. But many tricks, helpful though they may be in mental computation, will be devoid of any value with a view to computational machinery.[49]

In contrast to what they seem to have felt to be the amateurish efforts of the pioneers of artificial intelligence, logicians like Beth and Wang were convinced that, as Abraham Robinson put it in 1957, "Mathematical Logic [could] do more than provide a notation for the detailed formulation of a proof on a computer."[50] They also knew, however, that to fulfill their discipline's promise, they had to go beyond implementing decision procedures and show how a theorem prover for the full first-order predicate calculus could be built.

By 1960, three such theorem provers had been constructed: two in the US, by Paul Gilmore[51] and Hao Wang,[52] and one in Sweden, by the logician Dag Prawitz and colleagues.[53] This first generation of predicate-calculus theorem provers had, however, definite practical limitations. Procedures which it could be shown would eventually find a proof were of little use if "eventually" was in practice sufficiently long that only very elementary theorems could be proven. In 1960, for example, Gilmore described how his pioneering predicate-calculus theorem prover was easily crippled by the combinatorial explosion.[54]

The most important single step forward in this tradition was the work of John Alan Robinson on "resolution." Born in Yorkshire in 1930, Robinson came to the United States in 1952 with a classics degree from Cambridge University. He studied philosophy at the University of Oregon before moving to Princeton, where he received his PhD in philosophy in 1956. Temporarily "disillusioned with philosophy," he went to work as an operations research analyst for DuPont, where he learned programming and taught himself mathematics.[55] Robinson moved to Rice University in 1961, spending his summers as a visiting researcher at the Argonne National Laboratory's Applied Mathematics Division. Its then-director, William F. Miller, pointed Robinson in the direction of theorem proving. Miller "was thinking a great deal about 'what could be automated,' including data analysis, control of experiments, aids to theory development, design automation, and aids to programming. I was much influenced by Allen Newell and had many discussions with him."[56]

Miller showed Robinson a 1960 paper by Martin Davis and Hilary Putnam (coincidentally, the latter had been Robinson's PhD supervisor) proposing a predicate-calculus proof procedure that seemed potentially superior to Gilmore's, but which they had not yet turned into a practical computer program.[57] Miller suggested that Robinson use his programming skills to implement Davis and Putnam's procedure on the Argonne IBM 704. Robinson quickly found that their procedure remained very inefficient. However, while implementing a different procedure also suggested in 1960 by Dag Prawitz,[58] Robinson came to see how the two sets of ideas could be combined into a new, far more efficient, automated proof procedure for first-order predicate logic: "resolution."

Resolution (described in the sidebar on page 10) is an explicitly "machine-oriented" rather than "human-oriented" form of inference:

> Traditionally, a single step in a deduction has been required, for pragmatic and psychological reasons, to be simple enough, broadly speaking, to be apprehended as correct by a human being in a single intellectual act. No doubt this custom originated in the desire that each single step of a deduction should be indubitable, even though the deduction as a whole may consist of a long chain of such steps.

However, the smallness of such inference steps contributed greatly to the "combinatorial explosion" that dogged theorem provers: The "space" that had to be searched to find a proof grew too fast as the number of steps in the proof increased. By comparison, resolution was far more complex than the individual steps in human inference: Resolution "condones single inferences which are often beyond the ability of the human to grasp (other than discursively)."[59] Since resolution was designed to be implemented on a computer, the fact that it was opaque to human beings did not matter, according to Robinson.

In essence, a resolution theorem prover is provided with the axioms of the field of mathematics in question, and with the negation of the conjecture whose proof is sought. It then systematically resolves pairs of clauses (see sidebar on page 10 for what this means) until the empty clause, which represents a contradiction, is reached (the clauses *p* and not-*p* resolve into the empty clause). Robinson showed that first-order predicate logic, with resolution as the sole rule of inference, was complete: If the conjecture is indeed a theorem, this procedure must terminate. There was, therefore, no need for any other rule of inference, and so no need for choices between rules that would add to the proliferation of the search space.

Automated theorem proving after resolution

Resolution greatly increased the potential power of predicate-calculus theorem provers. Once its significance was realized — Robinson's 1965 paper on resolution languished unread on the desk of a referee for over a year after its September 1963 submission to the *Journal of the Association for Computing Machinery*[60] — it generated intense interest. Alan Robinson rapidly elaborated on his early work, as did others: at Argonne (where the leading figures were Larry Wos and George Robinson); at Stanford University (where John McCarthy used Lisp to write and test a resolution theorem prover, taking, to Alan Robinson's amazement, only a few hours to do so); at Edinburgh University (where the enthusiasm of Bernard Meltzer, a reader in electrical engineering, for resolution played an important part in establishing what became the Department of Artificial Intelligence); and elsewhere.[61,62]

Among topics treated were

- the generalization of resolution from resolving pairs of clauses to resolving arbitrarily large sets of clauses,
- strategies for optimizing the order in which clauses were resolved,
- formalisms alternative to clausal form, and
- ways of incorporating an "understanding" of equality into the resolution procedure. (Human mathematicians have a finely tuned intuitive understanding of how to handle the substitution of "equals for equals," but explicit axioms governing equality greatly increase the search spaces of resolution theorem provers.)

By 1978, Donald Loveland was able to include 25 variants of resolution in his textbook of automated theorem proving.[63]

The "resolution boom"[64] in the middle and late 1960s was, however, based on more than an interest in automated theorem proof. The predicate calculus, after all, was a general system for formal reasoning, not one specific to mathematical reasoning. *If* human knowledge could successfully be expressed in the predicate calculus, and *if* resolution provided efficient automated theorem provers for that calculus, then the general problem of artificial intelligence would be well on the way to solution. While these two preconditions seemed plausible, the two approaches we have been discussing — artificial intelligence and mathematical logic — seemed to converge rather than diverge. Alan Robinson, for example, was invited to join the editorial board of the former field's leading journal, *Artificial Intelligence*.

The plausibility of the two preconditions was, however, precarious. Among the early pioneers of artificial intelligence, the most prominent proponent of the predicate calculus was John McCarthy, who from 1962 headed the Artificial Intelligence Laboratory at Stanford University.[65] McCarthy had long felt that the predicate calculus, suitably extended, could be used satisfactorily to express human knowledge. But not all of the field's leaders were convinced. Criticism coalesced around McCarthy's former colleague at the MIT Artificial Intelligence Laboratory, Marvin Minsky. Minsky, along with Seymour Papert and others in the MIT group, argued vigorously that formal logic was inadequate for the representation of knowledge required by any general approach to artificial intelligence.

For example, the MIT group felt that resolution was an overly uniform approach to deduction. As Papert put it, "As long as your methods are supposed to be good for proving everything, they're not likely to be good for proving anything." As a general proof procedure for the predicate calculus, resolution did not make use of domain-specific knowledge; for example, the specificities of particular areas of mathematics were represented only through the presence of the axioms of each field. More generally, the predicate calculus itself was, the MIT group felt, an overly static and rigid framework for representing knowledge. When humans solve problems, said Papert, they "draw on very disparate [knowledge], not necessarily coherent and so not representable in strictly logical form."[66]

Instead, the MIT group believed it preferable to embody knowledge in procedures to be applied to input data. "Even such a simple statement as, 'If there are no cars coming, cross the road' . . . is more naturally transcribed as, 'Look left, look right, if you haven't seen a car, cross,'" rather than as a predicate-calculus formula, they wrote in 1971. Imagine, they went on caustically, a theorem prover "trying to prove by resolution or other logical principles that no car is coming!"[67]

What one participant has described as a "bitter civil war" broke out within artificial intelligence between "logicists" (such as McCarthy) and "proceduralists" (such as Minsky and Papert). Within the field, the two camps even earned the jargon labels of "neats" (logicists) and "scruffies" (proceduralists).[68] The "scruffies" seemed in practice to win the early skirmishes in this civil war, and interest in automated theorem proof suffered accordingly.

In the 1970s, furthermore, the focus of many of the remaining "logicists" within artificial intelligence was no longer primarily on automated theorem proof per se but on the burgeoning new field of logic programming. Logic programming is in a sense an application of automated theorem proof; among those contributing to its development was Robert Kowalski, a logician in Meltzer's group at Edinburgh, who had done important work on resolution.[69] For example, in the logic programming language Prolog (developed at the University of Marseilles and refined at Edinburgh), problems are stated in the form of goal clauses, or theorems, which the system then seeks to prove, using a resolution theorem prover. The requisite theorem prover, however, does not have to be a powerful or sophisticated one; many practical Prolog programs require only very limited search.

It was fortunate that this was so, because the early resolution-based theorem provers were in practice disappointing. This, at least as much as any overall "ideological" reaction against logicism, led the resolution "boom" of the late 1960s to turn to "bust" by the early 1970s. In essence, the problem was that while resolution ameliorated the combinatorial explosion, it did not eliminate it. The resolution-based automatic theorem provers of the period still got bogged down in the search for proofs beyond quite modest lengths. So, for example, when Sir James Lighthill reviewed progress in artificial intelligence on behalf of the UK Science Research Council early in 1972, he was told by "those most involved" that "this is particularly an area where hopes have been disappointed through the power of the combinatorial explosion in rapidly canceling out any advantages from increase in computer power."[70]

Although there is no evidence that they directly influenced perceptions of resolution, results in the early 1970s from the burgeoning field of the complexity of algorithms indicated that even simple forms of automated theorem proving, such as checking whether a given formula in the propositional calculus is a tautology, were "NP-complete" problems,[71] and the full predicate-calculus automated theorem proving attempted by resolution was worse, in that the search space for a nontheorem can be infinite. (At issue in "NP-completeness" is the behavior of an algorithm as a function of the size of input cases [for example, the length of the formula in the propositional calculus whose "tautologyhood" is being checked]. "P" is the class of problems that can be solved by a standard deterministic Turing machine in a time that is a polynomial function of the size of input cases. "NP-complete" problems are a set of problems, convertible into each other, that have the properties (1) that if one could guess a correct solution [such as a truth assignment making a formula false], it could be checked in polynomial time, and (2) that their only known deterministic solutions are exponential functions, rather than polynomial functions, of the size of input cases. The set is "complete" in that any of these problems is convertible into any other, and so if any one of them belonged to P, they all would.[72])

In essence, the problem was that while resolution ameliorated the combinatorial explosion, it did not eliminate it.

The combination of ideological reaction and practical disappointment meant a rapid decline in the 1970s in the salience of automated theorem proof as a topic in artificial intelligence. A key role in preventing it from being eclipsed altogether was played by Woodrow Wilson Bledsoe. Born in Oklahoma in 1921, Bledsoe served as a captain in the US Army Corps of Engineers before taking his PhD in mathematics from the University of California. By the late 1950s, he was the manager of the Systems Analysis Department of the Sandia nuclear weapons laboratory in Albuquerque, New Mexico. However, he developed an interest in artificial intelligence that caused him to leave Sandia in 1960 to help found one of the earliest commercial firms in artificial intelligence, Panoramic Research Inc. In his 1985 presidential address to the American Association for Artificial Intelligence, he recalled:

Twenty-five years ago I had a dream. . . . I dreamed of a special kind of computer, which had eyes and ears and arms and legs, in addition to its "brain.". . . My dream was filled with the wild excitement of seeing a machine act like a human being, at least in many ways. . . . When I awoke from this day-dream, I found that we didn't have these things, but we did have some remarkable computers, even then, so I decided then and there to quit my job and set about spending the rest of my life helping to bring this dream to reality. . . . Recently a reporter asked me, "why do you scientists do AI research?" My answer, "well certainly not for money, though I wouldn't mind being rich. It goes deeper, to a yearning we have to make machines act in some fundamental ways like people."[73]

In 1966, Bledsoe became professor of mathematics at the University of Texas at Austin and played a major role in the development of computer science and artificial intelligence there.[74]

Bledsoe's early work on automated theorem proving was in the resolution approach, but, like others in the early 1970s, he was sharply disappointed with the limited success of resolution theorem provers. He responded, however, not by abandoning the enterprise of automating theorem proving, but by devoting himself to what in a 1977 article he christened "non-resolution theorem-proving":

The word "knowledge" is a key to much of this modern theorem-proving. Somehow we want to use the knowledge accumulated by humans over the last few thousand years, to help direct the search for proofs.[75]

Like the MIT group, Bledsoe felt that resolution was an overly uniform proof procedure. He was impressed by the new procedural programming languages, such as Planner, developed by Carl Hewitt of the MIT group,[76] which were designed to facilitate domain-specific inference procedures. Furthermore, Bledsoe felt that much of the legacy of mathematical logic was inappropriate to "human-like" automated theorem proof. Inference rules should be "natural," he argued.[77] Logicians had typically been concerned to restrict to a minimum the number of inference rules employed in logical systems. That simplified proofs of "theorems *about* logics,"[78] but Bledsoe did not want to prove theorems about logics; he wanted to design systems that proved mathematical theorems in a way at least loosely analogous to how human mathematicians did it. In particular, unlike the logicians, who prized completeness, Bledsoe was unconcerned if the proof procedures he used were incomplete, so long as they yielded more natural and powerful proofs.

Like the original artificial intelligence pioneers, Bledsoe felt that the development of appropriate heuristics was essential. His first efforts in this direction, reported in 1971, were in set theory. He developed heuristics (some general, some specifically designed for application to set theory) whose "effect is to break the theorem [to be proven] into parts which are easier to prove." Resolution, "when used, is relegated to the job it does best, proving relatively easy assertions."[79] He tested out his theorem prover on theorems from a 1965 set-theory textbook written by his mathematics PhD supervisor, A.P. Morse.[80] In at least one case, Bledsoe's system found a proof that was "more direct and natural" than Morse's: The new heuristics "reduced [the theorem] to its essence," and the automated system "acted 'humanlike' in getting to the 'nub of the problem.'"[81]

Soon Bledsoe abandoned resolution altogether in favor of a more "natural" procedure for handling deductions in the predicate calculus "which we believe is faster and easier to use (though not complete)" and which "bears a closer resemblance to the proof techniques of the mathematician than does resolution."[82] The new procedure, together with a "limit heuristic" specifically designed to aid proofs in the calculus, and routines for algebraic simplification and for solving linear inequalities, produced a powerful theorem prover that could prove many theorems of elementary calculus, an area in which existing resolution provers had had little success.

Despite the overall reaction against resolution, and the surge of interest — centered around Bledsoe — in nonresolution theorem proving, some remained loyal to the original technique. The major site that did so, throughout the bleak years of the 1970s and early 1980s, was the Argonne National Laboratory, location of Alan Robinson's original introduction to automated theorem proving. At Argonne, Larry Wos and George Robinson continued theoretical work on proof procedures, for example, combining unification (see sidebar on page 10) with an inference rule for equality to yield the powerful rules of "demodulation"[83] and "paramodulation."[84] To these theoretical advances were added continuous practical improvements (focusing on matters such as indexing and the storage and retrieval of clauses), notably by Ross Overbeek and Ewing Lusk.

The Argonne systems — Aura (Automated Reasoning Assistant), now replaced by Otter, developed by William McCune — are quite different from the simple goal-directed inference engine at the heart of Prolog. Fast and sophisticated, Aura and Otter usually proceed with a "best-first" search, focusing on relevant parts of the theory, the special hypotheses, and the goal, rather than simply searching from the goal backwards, in a "depth-first" search (as in the Prolog prover). For several years the Argonne systems attracted relatively little attention. At the end of the 1970s, and in the early 1980s, however, they began to be applied successfully, not just to theorems whose proofs were known, but to the proof of open conjectures in several specialized fields of mathematics (conjectures that humans had formed but had been unable themselves to prove). (See the articles by Winker and Wos in 1978,[85] by Winker, Wos, and Lusk in 1981,[86] by Winker in 1982,[87] by Wos in 1982,[88] and by Wos and colleagues in 1983.[89])

The Argonne group's successes contributed directly to a distinct revival, in the 1980s, in the fortunes of automated theorem proving in general and in resolution theorem proving in particular. Several new groups emerged, and the field began to take on something of the character of what Thomas Kuhn calls "normal science."[90] A set of standard theorem-proving tasks have emerged that serve as the benchmark for competing systems, and researchers seek to improve the performance of their systems in terms of the time taken to produce the benchmark proofs. Passing on the editorship of the *Journal of Automated Reasoning* in 1992, Larry Wos was able to note with satisfaction the change since the dark days of the mid-1970s when the "basic problem" of automated theorem proving had been seen "to be monumental to solve."[91]

Interactive theorem provers and program proof

Another reaction, however, to the difficulties faced by the theorem provers of the 1950s and 1960s was to abandon the attempt to make them wholly automatic and to make provision for direct human guidance of their search for proof. With early mainframe computers operating in "batch mode," this would have been very cumbersome, but interactive theorem proof was made much more attractive in the late 1960s and early 1970s by the rapidly increasing availability of time-sharing computer systems.

It can be misleading to make too much of the contrast between "automatic" and "interactive" theorem provers. In actual practice, even resolution theorem provers are often "tuned" by humans for particular problems, for example, by altering the "weights" governing search strategies. Even an apparently irrelevant matter such as the order in which the user enters axioms into the system can influence whether a given theorem can be proven, because it determines the default order in which clauses are considered. Nevertheless, the automaticity of the early theorem provers was important to their attractiveness both to the artificial intelligence pioneers and to mathematical logicians. Explicitly interactive systems indicated the emergence of developers with rather different interests.

One such group of developers worked in the mid and late 1960s at the Applied Logic Corporation in Princeton, New Jersey. With funding from the Air Force and the Advanced Research Projects Agency, they developed a series of systems they called SAM, or semi-automated mathematics:

> Semi-automated mathematics . . . seeks to combine automatic logic routines [with] 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.[92]

In 1966, this group was using the fifth of these systems, SAM V, implemented on a time-sharing PDP-6 computer, to construct proofs of theorems from a 1965 article by Robert Bumcrot on lattice theory.[93] The mathematician using SAM V realized that the system had proven an intermediate result that yielded "as an immediate consequence" the proof of a conjecture Bumcrot had posited but not proven.[94] This result — SAM's lemma, as it became known — was widely hailed as the first contribution of automated reasoning systems to mathematics.

Others saw automated but interactive systems as contributing to mathematics in a subtly different way, as "proof checkers." These systems were to be provided with a full (or nearly full) formal proof, constructed by a human being, and were to check whether this proof was indeed a correct one. The most influential such system was Automath, developed in 1967/68 at the Technische Hogeschool, Eindhoven, by N.G. de Bruijn and colleagues. Automath

. . . the true disciplinary home of interactive theorem provers was computer science.

> . . . is defined by a grammar, and every text written according to its rules is claimed to correspond to correct mathematics. . . . The rules are such that a computer can be instructed to check whether texts written in the language are correct.[95]

For example, L.S. van Benthem Jutting, a student of de Bruijn, successfully translated into Automath, and automatically checked, the proofs from Edmund Landau's text *Grundlagen der Analysis*.[96,97]

Systems such as Automath could indeed be seen as answers to Nidditch's complaint, quoted above, about the lack of full formal proof in mathematics. The bulk of mathematicians, however, were exercised neither by the complaint nor by the remedy. Neither "semi-automated mathematics" systems nor proof checkers were widely adopted by them. As two mathematicians put it in 1987:

> The formalized counterpart of normal proof material is difficult to write down and can be very lengthy. . . . Mathematicians are not really interested in doing this kind of thing.[98]

Instead, the true disciplinary home of interactive theorem provers was computer science. In this field, there was a direct practical motive for interest in proof checkers and in more automatic, but still interactive, theorem provers: the goal of proving the correctness (that is to say, correspondence to specification) of computer software, and to a lesser extent hardware.

During the 1960s, there was a growing sense of the propensity of software to be late, over-budget, and full of errors, a sense crystallized in the diagnosis of a "software crisis" at the October 1968 NATO Conference on Software Engineering in Garmisch Partenkirchen.[99] The idea that this could be remedied by the rigorous application of mathematics to computer science gathered force.[100] Mathematical proofs of the correctness of computer programs stretch back into the second half of the 1940s; for mathematicians like John von Neumann and Alan Turing, nothing could have been more natural.[101,102] Only in retrospect, however, has it been seen that this was early research on "program verification." The latter field began to take shape only in the 1960s,[103] with the now-classic papers by John McCarthy in 1963,[104] Peter Naur in 1966,[105] Robert W. Floyd in 1967,[106] and C.A.R. Hoare in 1969.[107]

The earliest proofs to which this body of work gave birth were human, manual proofs (see, for example, McCarthy and Painter's 1967 proof of correctness of a simple compiler[108] and the proofs in Ralph London's 1970 article[109]). However, for all but the very simplest programs, such hand proofs were intricate, "tedious,"[110] and typically lacked the elegance and general interest prized by mathematicians. Computer implementation of them therefore seemed highly desirable.

The first automated program verifier was constructed by James Cornelius King, a PhD student in the Department of Computer Science at Carnegie Mellon University, and reported in his 1969 thesis.[111] King was a student of verification pioneer Robert W. Floyd and was also in touch with Alan Newell. King's verifier was an automatic one and was used to prove the correctness of some simple example programs. The next verifier to be built, however, was interactive. It was constructed by Donald I. Good, a PhD student at the University of Wisconsin, supervised by Ralph London.[112] Although it had some automatic features, Good's system "was largely a sophisticated bookkeeping system for accepting and keeping track of manual proofs."[113]

These systems were harbingers of a decade in which, especially in the US, program verification and the development of automated tools to assist this verification became hot topics for computer science researchers. As the interest in automated theorem proof in artificial intelligence fell off sharply in the early 1970s, it was replaced by this new interest from computer scientists. For the latter, there was no reason (as there had been for those whose concern was artificial intelligence) to insist that proof systems be automatic. By the mid-1970s, hopes were high that in the medium-term future, interactive, mechanized proofs of "real" (rather than "toy") programs would be produced.

Those hopes found their single most important receptive audience in the computer security community in the United States. As computers played an increasingly important role in the operations of the US military, the question naturally arose of how to translate the traditional document-based system of security classification and management to make it suitable for the new electronic systems. How computer systems could be made secure, and how they could be *shown* to be secure, were vitally important questions.

In 1973, D.E. Bell and L.J. LaPadula of the Mitre Corporation put forward what became the paradigmatic mathematical model of computer systems security.[114] All subjects (including, but not restricted to, human users) and objects (such as data files) are assigned one of a number of fixed, hierarchically ordered, security classes. One rule in the Bell-LaPadula model, closely analogous to that in a document-based system, is that a subject can have *read* access only to objects whose security class is less than or equal to the security class of the subject. Their model also includes the rule, without direct, explicit human analogue, that subjects can have *write* access only to objects whose security class is greater than or equal to the security class of the subjects. (Without this second property, security could be violated by, for example, "secret" information being written to an "unclassified" file.) The rules are summarized as *read* down, *write* up.

Influential figures in the US computer security community were attracted by the notion that it might be possible mathematically to prove that systems conformed to the Bell-LaPadula or similar models, and accordingly national security interests (including the Defense Advanced Research Projects Agency and the National Security Agency) channeled substantial research funds into the area of program proof, supporting several major efforts of this kind in the 1970s. For example, Donald Good was funded to develop a full-scale system to support "the development of software systems and formal, mathematical proofs about their behaviour."[115] The resultant system, christened the Gypsy Verification Environment, contains a programming and specification language (based on Pascal), an editor, a compiler, a verification condition generator (which produces the predicate-calculus formulae that have to be proven in order to show that a program satisfies its specification), and an interactive automated theorem prover.

For the latter, Good (who in 1970 had joined the Computer Science Department of the University of Texas at Austin) adapted an interactive prover developed by Woody Bledsoe (also at Austin) and his colleague Peter Bruell.[116] The Gypsy prover, wrote Good in 1983,

> is really more like a proof checker than a proof constructor. It provides a list of sound deductive manipulations that a user can apply interactively to the current subgoal. The user directs the prover by telling it what manipulations to perform until the formula is reduced to TRUE.[117]

The Gypsy Verification Environment, and its associated theorem prover, were used to perform one of the most impressive early program proofs, the 4,200-line Encrypted Packet Interface:

> The problem was to take two communicating computers across the network and in between each computer [and] the network to put a box that basically encrypted messages going out of the network and decrypted them coming back.[118] [119]

This kind of success was, however, far from universal. Several major security-funded program verification efforts of the late 1970s failed to achieve their goals.

Confidence in the state of the art in program verification had therefore ebbed by the early 1980s when the US Department of Defense came to formulate its standards for secure computing systems.[120] The resultant standards, known from the color of the cover of the document containing them as the "Orange Book," did not demand proof of program correctness, even for the highest category of security assurance, A1.[121]

Instead, what was required for this category were "design proofs": proofs that a system's specifications implemented a suitable "security policy" (essentially the Bell-LaPadula model, together with an analysis of "covert channels" through which information can flow without files being read). Nevertheless, even this requirement placed considerable demands on automated theorem-proving systems. The National Computer Security Center, set up as a "visible" unit of the National Security Agency to oversee the implementation of the Orange Book, supported two such systems: Gypsy and the System Development Corporation's FDM (Formal Development Methodology). Like Gypsy, FDM's theorem prover requires detailed human guidance, and the two systems are in a broad sense similar.[122]

This involvement with computer security has led to Gypsy and FDM becoming subject to US government arms export control limitations. In 1980, for example, the Computer Network Critical Technology Expert Group recommended that "automatic program correctness verification [systems], in particular when applicable to secure systems' development," be placed on the Department of Defense's list of "critical technologies" whose "exports should be restricted for reasons of national security."[123] This kind of restriction, together with the heavy demands placed by implementation of the Orange Book on the available pool of US personnel skilled in automated theorem proving, has led some to question whether the Orange Book may have harmed rather than helped the development of verification technology in the US.[124]

Another highly influential US interactive theorem-proving system, one less tied to computer security interests, has been the Boyer-Moore theorem prover, or NQTHM (New, Quantified Theorem Prover), "an uninspired parochialism that has taken on a life of its own."[125] In 1971, common membership in Bernard Meltzer's group at Edinburgh University brought together Robert S. Boyer, a PhD student of Woody Bledsoe's at the University of Texas, and J. Strother Moore, an MIT mathematics graduate with a strong interest in artificial intelligence and considerable programming experience. Boyer and Moore continued their collaboration on their return to the United States, first at SRI International and Xerox PARC, then at the University of Texas at Austin, and finally at Computational Logic Inc., the firm they established along with Don Good and colleagues in 1983.[126,127]

The Boyer-Moore theorem prover's most distinctive feature is its emphasis on mathematical induction. Suppose, to take the simplest kind of induction, one wants to show that a formula $P(n)$ is true for all the natural numbers, n (that is, 0, 1, 2, . . .). The principle of mathematical induction permits one to assert this, provided that one has proven (1) that $P(0)$ is true and (2) that if for an arbitrary natural number k, if $P(k)$ is true, then $P(k + 1)$ is true.

This principle cannot be expressed straightforwardly in first-order predicate calculus because it involves quantification over all predicates (see above). It is, therefore, typically not employed in resolution theorem provers. In

The Boyer-Moore theorem prover's most distinctive feature is its emphasis on mathematical induction.

NQTHM, by contrast, induction is provided as a rule of inference,[128] and much of the structure of the system revolves around it. Induction is an important technique in proving properties of programs, where it can appear in forms other than its above form as induction over the natural numbers.[129] For example, the original application of the Boyer-Moore theorem prover was to John McCarthy's Lisp programming language, in which inductively constructed objects, notably lists, play a central role, and for which induction over these structures is thus a vital proof technique.[130] (A theorem about lists can be proven by [1] showing it holds for the empty list and [2] showing that if it holds for an arbitrary list, it also holds for that list with a new element added.)

The chief difficulties in automating induction proofs are the choice of parameter on which to attempt induction and the choice of the induction hypothesis, which may, for example, need to be more general than the theorem whose proof is sought. Heuristics for guiding these choices are essential to the Boyer-Moore theorem prover; the influence of Boyer's supervisor Bledsoe is clear here.[131] "I guess the foremost remarkable thing about the Boyer-Moore theorem prover is its little trick of sometimes guessing an induction to do," says Boyer.[132]

This "trick" is actually integrally linked to the logic chosen by Boyer and Moore, which was (except for its inclusion of induction as a rule of inference) simpler than normal first-order predicate calculus. Explicit use of the normal predicate-calculus quantifiers ("for all," "there exists") was not permitted. Recursive functions were used instead. (A function f over the natural numbers can be defined recursively by [1] defining the value of $f[0]$ and [2] giving a means of calculating the value of $f[k + 1]$ from that of $f[k]$.) This has the effect

> of forcing the user to hint implicitly how to prove conjectures: try inductions that mirror the definitions of the recursive functions used in the conjectures.[133]

Unlike resolution theorem provers, the Boyer-Moore theorem prover does not leave the choice of axioms entirely to the user. Built into it are the recursively structured axioms of Peano arithmetic and similar axioms for ordered pairs. (Unlike Russell and Frege, Guiseppi Peano [1858-1932] defined the natural numbers in terms of a successor function s, where $s[0] = 1$, $s[1] = 2$, etc. Addition, for example, is then defined by the axioms

$$X + 0 = X$$
$$X + s[Y] = s[X + Y].)$$

New user-supplied axioms must pass a "correctness test" (essentially of the well-foundedness of the definition of recursive functions) designed to stop the inadvertent introduction of inconsistencies.

Although it was originally wholly automatic (and it is far more automated than many other interactive provers), the Boyer-Moore prover requires detailed human guidance for difficult proofs:

> When you are proving theorems with our theorem prover you type in conjectures. And if the conjectures are of a certain form, and you are extremely lucky, then the theorem prover will say that they are true. But, more likely than not, it will start printing out all kinds of messages about what it's trying to do, and so forth. And after using the thing for a number of years you can decipher these messages and say, "Aha! What the problem is, is that the theorem prover does not know about the following intermediate lemma." So you go off and you say, "First try to prove such-and-such," so it goes off and tries to prove such-and-such, and if you are extremely lucky it will prove that lemma. Then it will prove the theorem that you wanted in the first place. More than likely you will have to do this process hundreds of times.[134]

In order to successfully guide the Boyer-Moore prover by suggesting lemmas, users need to understand both the proof being sought and the internal operation of the theorem prover. "You [also] have to understand the logic [and] how to represent mathematics in that logic . . . [Since the] logic is extremely weak, not first-order . . . you have to know all kinds of tricks to represent the mathematics."[135] These skill-intensive aspects of its use have meant that "nearly all the successful users of NQTHM have in fact also taken a course from [Boyer and Moore] at the University of Texas at Austin on proving theorems in our logic."[136]

Graduate students from this course, which began in 1981, have been the source of many of the most impressive recent uses of the Boyer-Moore system to prove both mathematical theorems and program and hardware correctness. These students "saw our mechanically checked proofs of the simpler theorems . . . and came to us believing they could lead the expeditions to the harder theorems."[137] The most striking example to date has been the automated proof by Natarajan Shankar, in his 1986 thesis, of Gödel's incompleteness theorem.[138]

A different, less highly automated, approach to interactive theorem proving, with human control more direct than in the Boyer-Moore prover, is represented by LCF (Logic for Computable Functions), developed by Robin Milner, first at Stanford University and then at the University of Edinburgh.[139] Milner's goal was to mechanize proofs in PPLambda, a family of logical calculi, rooted in the lambda calculus, which had been proposed by Dana Scott as particularly appropriate for the study of the semantics of programming languages. (Introduced by the logician Alonzo Church, the lambda calculus hinges on the operation of "abstraction," indicated by the Greek letter lambda. It allows reasoning about functions [and, for example, the operation of functions on other functions], while "abstracting away" from the particular variables to which functions apply. It therefore permits escape from the restrictions of first-order logic.)

In its original 1972 Stanford version, Milner's system included a facility for the creation of subgoals, a mechanism for automatic simplification of formulae, and a means of storing theorems so that the user could build a "library" of theorems over several sessions of work.[140] Nevertheless, proof construction using Stanford LCF required the user to type in large numbers of often highly repetitive commands.

By 1977, however, Milner had introduced a new "Edinburgh" version of LCF. The most important change was the introduction of the "metalanguage," ML, with which the user could write proof procedures.[141] Central to ML's operation is compile-time type checking. (In programming, one often has different kinds of data structures, such as numbers, arrays, and lists. Each kind of data structure can be assigned a "type," and type checking can then detect some forms of errors, such as the instruction to add $a + n$, where a is an array and n a number.)

Among ML's types is the type thm ("theorem"). Objects of type thm can be constructed, in Edinburgh LCF, only by applying the rules of inference of PPLambda to existing objects of type thm, which may include user-defined "axioms." Rules of inference are thus seen in ML as functions from theorems to theorems. Type checking should guarantee that anything of type thm has indeed been formed only from axioms and rules of inference, so that a proposition denoted by a term of type thm is indeed a theorem. Soundness therefore depends not upon the entirety of an LCF-style system, but on the program segments encoding axioms and the rules of inference of the logic, together with the ML implementation's type-checking code. Although the former can be hundreds of lines long, and the latter over a thousand, they are still smaller, and thus more readily checkable, than entire complex theorem-proving systems. Proponents of LCF-style theorem proving suggest that it can therefore give greater assurance of soundness than more complex kinds of theorem prover.[142]

It is interesting to note, however, that in theorem proving with Edinburgh LCF, no explicit sequence of steps from axioms to theorem is typically produced. Rigor lies in ML's type-checking discipline rather than in the production of a sequence of formulae. This feature of Edinburgh LCF, as distinct from, for example, Stanford LCF, "released one from the need to preserve the whole proof as a sequence (though does not deny this possibility); a user of LCF should consider that he is performing a proof, or guiding its performance, not generating it."[143]

As well as permitting this subtle shift of perspective, ML allowed the encoding not just of primitive inference steps but of "recipes"[144] for the performance of proofs; this, indeed, was Milner's primary motivation for developing the language. These recipes include "tactics" — ways of reducing the goal, a desired theorem, to "a list of subgoals such that if every subgoal holds then the goal holds also"[145] — and higher level "tacticals," which combine tactics.

LCF has been followed by several other interactive theorem provers that are directed by users through proof procedures expressed as ML programs. The flexibility of this design permits relatively easy implementation of a wide range of different logics. Thus the "Nuprl" system developed at Cornell University by R.L. Constable and colleagues employs not the "classical" logic of Hilbert and the formalists but a "constructive" type theory that has its roots in the logic pioneered by Hilbert's opponent, the famous Dutch mathematician L.E.J. Brouwer (1881-1966).

The key (though not the only) divide between classical and constructive logic is over the law of the excluded middle — that either a proposition or its negation must be true. Brouwer argued that excluded middle should not be used in proofs concerning infinite sets. It nevertheless remained widely used in such proofs. It is implicit in the resolution rule and, indeed, in most automated theorem provers.

Brouwer's original rejection of excluded middle had complex philosophical, professional, and even psychological grounds.[146] Constable's constructivism, by contrast, is motivated by the fact that a "constructive" proof of the existence of a mathematical entity has to show how to construct such an entity, since it cannot rely (as in many "classical" proofs) on showing a contradiction that would follow from its nonexistence. Nuprl therefore supports an alternative approach to proof of program correctness: Rather than write a program and then try to show it meets its specification, first produce a (constructive) proof of the existence of the desired mathematical object and then from that automatically synthesize a program that will generate the object.[147]

Classical (rather than constructive) higher order logic has also been implemented in an ML-style theorem prover. (In higher order logic, quantification is not restricted merely to variables as it is in first-order logic. It is therefore more expressive than first-order logic, but it suffers from the disadvantage that proofs in it are much harder to automate. Unification algorithms [see sidebar on page 10], for example, are frequently nonterminating.) The main motivation for recent interest in theorem provers for higher order logic has come from hardware verification rather than program verification. The use of higher order logic for hardware verification was first suggested by Keith Hanna of the University of Kent in the early 1980s. Thus, Hanna and Daeche in a 1986 article note that "a waveform is a function from time to voltage," and hence to be able to assert a property of waveforms requires not first- but second-order logic.[148] A behavioral specification of a logic gate might involve the assertion that under all conditions a particular relationship must hold between different waveforms, and thus involves third-order logic, and so on.

There has been long-standing work on the automation of higher order logic by Peter Andrews and colleagues in the United States.[149] In the context of hardware verification, however, the most influential implementation of higher order logic has been by Michael J.C. Gordon, who worked with Robin Milner on Edinburgh LCF before moving to the University of Cambridge. His HOL system

The key (though not the only) divide between classical and constructive logic is over the law of the excluded middle — that either a proposition or its negation must be true.

is implemented on top of the Cambridge version of LCF, and its inference rules are expressed as ML functions.[150]

HOL proofs require intensive, skilled human guidance; thus, Cohn, in a 1987 paper, reports that a HOL proof of the correspondence of two levels of description of a simple microprocessor (Viper, Verifiable Integrated Processor for Enhanced Reliability) took six person-months to generate.[151] However, learning HOL does not appear to necessitate the extensive personal contact that the Boyer-Moore theorem prover usually requires,[152] and in recent years the HOL user community has grown considerably, making the system one of the most widely used automated theorem provers.

Discussion

A sociology of proving? The previous pages have revealed a variety of different approaches to the automation of proof. Three general strands can be identified: automatic theorem proving where the simulation of human processes of deduction is a goal; automatic theorem proving where any resemblance to how humans deduce is considered to be irrelevant; and interactive theorem proof, where the proof is directly guided by a human being.

The most straightforward of these clusters to comprehend is the third, interactive proof. As we have seen, interactive theorem provers and automated proof checkers have generally not taken root within mathematics itself. Instead, the practical goal underpinning the development of many of these systems has been the verification of computer programs, and, to an increasing extent, of hardware designs. Note that there is no evidence that the interactive nature of these systems is the result of a "principled" commitment to human guidance; indeed, the developers of these systems would be pleased to see them more automatic than they are. Rather, pragmatic considerations — above all the restricted capacities of wholly automatic provers, but also, for example, the desire for the expressive power of higher order logic — have dictated the role of the human being in guiding the proof.

More slippery, but also more tantalizing, is the contrast between the two camps of automatic theorem proving: the "human-like" and the "non-human-like."[153] True, there is no absolute divide between the two camps. Thus we have seen that Newell, Shaw, and Simon's Logic Theory Machine was largely algorithmic in its operation, despite the "heuristic" rhetoric of its developers. Furthermore, many modern theorem provers can be seen as hybrids of the two approaches. Even resolution theorem provers have to be equipped with (nowadays often elaborate) strategies to guide their search, and nonresolution theorem provers often turn out to rely upon an underlying proof procedure that has an "algorithmic" feel to it similar to that of resolution.

Nor are particular individuals necessarily aligned permanently with only one of the camps. Thus Woody Bledsoe, doyen of nonresolution theorem proving and the human-like approach, began his work on automated theorem proof by concentrating on resolution, and also made important contributions to the implementation of wholly algorithmic decision procedures.[154] Others have had a foot in both camps from the start of their careers, such as Donald Loveland, who was a master's student with artificial intelligence pioneer Marvin Minsky but did his PhD with mathematical logician Martin Davis and contributed to both the Geometry Machine and resolution-style theorem proving.[155]

Nevertheless, slippery though it is, the distinction between "human-like" and "non-human-like" automated proving is not altogether vacuous. It shows how practitioners with experience of both camps see the field as being divided.[156] It can be felt sharply not just in the debates of the late 1950s, but also in the 1970s, when, for example, Arthur Nevins (a leading proponent of the "human-like" approach) titled an article "A Human Oriented Logic for Automatic Theorem Proving,"[157] in explicit counterpoint to Robinson's earlier "Machine-Oriented Logic."[158] Even today, despite a definite attenuation of the distinction, its traces can still be felt, for example, in Larry Wos's statement that "our [Argonne, resolution-based] approach to the automation of reasoning differs sharply from approaches one often finds in artificial intelligence; indeed, our paradigm relies on types of reasoning and other procedures that are not easily or naturally applied by a person."[159]

The divide has even spilled over into areas of the automation of mathematical reasoning other than the automation of proof. Thus, reactions to Douglas B. Lenat's 1976 "AM,"[160] an ambitious attempt to automate not proof but mathematical discovery, closely echoed reactions to the Logic Theory Machine two decades earlier. AM was enthusiastically welcomed by those, such as Simon, who retained a commitment to a "strong" version of artificial intelligence. Others, though, were much cooler. The Logic Theory Machine's critic, Hao Wang, wrote that he found "Dr. D. Lenat's large dissertation . . . thoroughly unwieldy and could not see how one might further build on such a baffling foundation."[161]

Beneath this kind of disagreement lies a difference in goals. Herbert Simon recalls:

> I remember explaining to him [Wang] on several occasions that he was in a different business than we were . . . we were trying to understand how human beings do it . . . Everything we know about human beings suggests that the reason they are able to do interesting things is heuristic search. Occasionally things that human beings do that way, we can bludgeon through on a computer, but why go that way?

Simon sees the proper model for automated theorem proof as being not logic, but mathematics:

> Mathematicians introduce all sorts of inference rules which perhaps can be justified — who knows . . . They maybe can be justified, but you don't do that when you're doing the mathematics. You use rules which other mathematicians will agree: "Yeah, you could justify that." So you are very free about introducing those. You also use heuristics — you don't demand that your methods are complete.[162]

Different disciplinary commitments underpin these differences in goals. Newell, Simon, Gelernter, Bledsoe, and Lenat all saw their work as contributing directly to the development of artificial intelligence. Bledsoe's avowed "dream" is unique only in the explicitness and passion of its statement of commitment. The other central figures in this camp shared a broadly similar vision of the ultimate purpose of their work on the automation of mathematical reasoning.

What of the other camp? A coherent "dream" cannot be identified, but a coherent disciplinary base can be: logic. While older than artificial intelligence, this specialism is still, in a sense, relatively new. True, logic was part of the medieval university curriculum, but formal, mathematical logic as a distinct academic specialty stretches back only a few decades before the start of the period we focus on here. Despite predecessors like Boole and Frege, the occupation of mathematical logician is essentially a twentieth-century creation. Like artificial intelligence, mathematical logic too has manifested a mixture of confidence and insecurity. The insecurity is that of a proto-discipline positioned between the two existing powerful disciplines of mathematics and philosophy. Logicians could — and can — easily feel themselves to be, as Wang did, too philosophical for mathematicians and too mathematical for philosophers.[163] Career prospects in the field have sometimes been less than excellent, as recent heavy flows of logicians into computer science can testify.

On the other hand, logicians have had one great source of confidence: The work of Frege, Russell, and especially Gödel and his contemporaries established logicians as the recognized experts on the nature of mathematical proof. The hawks among logicians, like Nidditch, are even prepared to contrast their rigor with the shortcomings of mathematicians. Those involved in the automation of proof who have drawn most heavily upon logic by no means all share this harshness of judgment — Alan Robinson, for example, would see it as quite misplaced[164] — but when the work of the "artificial intelligence" cluster was perceived as "unprofessional" by Wang,[165] there was no doubt that the profession whose criteria were being applied was that of the logician.

However, much more was going on than simply a disciplinary dispute between artificial intelligence and logic. As we have seen, this external divide was replicated within artificial intelligence in the form of the dispute between "proceduralists" and "logicists," or "scruffies" and "neats." There is at least a loose correlation between, on the one hand, the perceived centrality of automated theorem proving to artificial intelligence and, on the other, the fortunes of the "logicist" school. The resolution "boom" coincided with, and indeed fueled, an expanding influence of logicism. The loss of confidence in resolution was linked to the growth of the proceduralist critique of logicism. The recent revival of interest within artificial intelligence in logic as a framework for knowledge representation has coincided with a renewed surge of interest in automated theorem proof.

These "civil wars" have, of course, attracted much less attention than the high-profile, wider debate about artificial intelligence. Given automated theorem proving's early centrality to the latter, it is not surprising that we should find resonances between different attitudes to the automation of proof (especially the divide between the artificial intelligence and logic approaches) and wider positions on artificial intelligence.

Among Simon and Newell's celebrated 1958 predictions for the future of artificial intelligence was "that within 10 years a digital computer will discover and prove an important new mathematical theorem."[166] The visibility of that prediction made it natural for critics of artificial intelligence to focus on weaknesses and disappointed expectations in automated theorem proving. Thus, in 1966, *The New Yorker*, house journal of American humanist intellectuals, seized with delight on Hubert Dreyfus's accusation that the proofs discovered by Gelernter's Geometry Machine were neither novel nor deep. At issue was a review by W. Ross Ashby of Feigenbaum and Feldman's 1963 collection, *Computers and Thought*,[167] in which Ashby had waxed eloquent about the Geometry Machine:

> Gelernter's theorem-proving program has discovered a new proof of the Pons Asinorum that demands no construction, [a proof that] the greatest mathematicians of two thousand years have failed to notice [and] which would have evoked the highest praise had it occurred.

Said *The New Yorker:*

> An ingenuous reader . . . might assume that Mr. Simon's prophecy about the important new theorem has already been realized . . . [However] the pons asinorum, or asses' bridge, is none other than the first theorem to be proved in Euclidean geometry — that opposite angles of an isosceles triangle are equal — and . . . the machine's "new" proof was originally introduced by Pappus in 300 A.D.[168]

It is interesting to note that Dreyfus, the most famous critic of artificial intelligence, is a philosopher, albeit one influenced more by "Continental" phenomenology than by the Anglo-Saxon analytical philosophy to which most of the logicians discussed here owe allegiance. He was happy to assimilate to his position Wang's critique of the artificial intelligence approach to the automation of proof:

> Heuristics are not only unnecessary here, they are a positive handicap, as the superiority of Wang's algorithmic logic program over Newell, Shaw and Simon's heuristic logic program demonstrates.[169]

Dreyfus quoted with approval Wang's description of computers as "slaves which are . . . persistent plodders."[170,171] Conversely, Wang expresses sympathy with Dreyfus's critique of artificial intelligence.[172]

That there are disciplinary issues here too, as well as questions of the overall attractiveness or otherwise of artificial intelligence, is perhaps indicated by the difference between Dreyfus's comments on automated theorem proving and the position adopted in the criticism by Sir James Lighthill, which was practically more damaging, since it led to a sharp decline in the fortunes of the field in the UK. Lighthill is an applied mathematician rather than a logician or philosopher. At the time of the "Lighthill Report" he was Lucasian Professor of Applied Mathematics at the University of Cambridge. He readily accepted the judgment of those in the field who felt that "algorithmic" resolution theorem provers were a dead end. Unlike Dreyfus, Lighthill endorsed the "modern trend . . . to 'heuristic' methods," welcoming the "outward-looking trend" to "utilisation of far more detailed observation of 'how mathematicians actually prove theorems'!"[173] Sir James's exclamation mark presumably indicates a mathematician's surprise that this had not been the basis of automated theorem proving all along.

A sociology of proof?

The above marked, and sometimes sharp, differences of opinion on how best to automate proving do not necessarily translate into differences of opinion on proof. One can, after all, believe that someone else's approach to proving is inefficient, inelegant, even "unprofessional," and yet still accept that what they or their systems produce are nonetheless valid proofs. It is indeed the case that nearly all the disagreement manifest in the history of the automation of mathematical proof is disagreement, in this sense, about "proving" rather than "proof." The "artificial intelligence," "logical," and "interactive" approaches to proving are all based upon the formal notion of what a proof is. Even the most human-oriented approach still seeks to simulate how humans might find a formal proof rather than trying to automate informal proofs. As Alan Robinson puts it, "Our present limited logical knowledge and the rigid algorithmic technology of today's computer science have so far forced us to work with formal proofs."[174]

Indeed, a very considerable, but entirely uncelebrated, achievement of the work whose history is reviewed here is that it has shown the practicality of formal proof. However extreme his view of what constitutes proof, Nidditch was surely right to note that in 1957 full formal proofs were extremely rare in mathematical practice, and that it was, in general, only a conjecture that "ordinary" informal proofs could be formalized. The success of the automation of mathematical reasoning in demonstrating, in many cases, the correctness of this conjecture should not be underestimated merely because intuitive faith in its correctness was strong. Here the work of Boyer, Moore, and their students is perhaps particularly noteworthy.

In that sense, the automation of proof has consolidated the triumph of the formal, Hilbertian notion of proof. Paradoxically, however, in mechanizing formal proof, the automation of mathematical reasoning has potentially reopened questions of what proof should be taken as consisting in. When, in 1927, Hilbert came to restate the essence of the formalist position, he gave a definition of formal proof effectively identical to that given in my second section above, but he added the proviso that the array of formulae constituting a proof "must be given as such to our perceptual intuition."[175]

Yet how possible is it in practice for a human being to "survey" the array of formulae constituting a large formal proof? This issue is indeed raised, quite independently of the automation of proof, by large human-generated proofs. Some large proofs, such as Andrew Wiles's celebrated 1993 claimed proof of Fermat's last theorem, are important enough to motivate other mathematicians to scrutinize them in detail. But is such scrutiny still a realistic possibility when one has a proof, like that of the classification of the finite, simple groups, that is "scattered across the [over 10,000] pages of some 500 articles in technical journals"?[176]

This issue of surveyability, however, arises particularly sharply with mechanized proofs. To some commentators, the automation of proof drives a wedge between the criterion of formality and that of surveyability. The celebrated critique of program verification mounted by computer scientists Richard DeMillo, Richard Lipton, and Alan Perlis in 1979 argued that program verification, and by implication the automated theorem provers that underpin it, are based on what they saw as a misunderstanding of the true nature of mathematical proof. True proofs, they insisted — and still insist[177,178] — have to be surveyable so that they can be subject to assessment through what they called the "social mechanisms of the mathematical community":

> These same mechanisms doom the so-called proofs of software, the long formal verifications that correspond, not to the working mathematical proof, but to the imaginary logical structure that the mathematician conjures up to describe his feeling of belief. . . . Being unreadable and — literally — unspeakable, verifications cannot be internalized, transformed, generalized, used, connected to other disciplines, and eventually incorporated into a community consciousness. They cannot acquire credibility gradually, as a mathematical theorem does; one either believes them blindly, as a pure act of faith, or not at all.[179]

Similar controversy has taken place within mathematics itself, particularly over the 1977 computer-assisted proof by Kenneth Appel and Wolfgang Haken of the four-color conjecture.[180] (This famous mathematical conjecture was first put forward in 1852, but no acceptable proof of it had been found. The conjecture is that four colors suffice to color any map drawn on a plane in such a way that no neighboring countries are the same color. "Neighboring" is taken to mean sharing a border, and the definition of "country" is restricted to regions that are in the topological sense "connected.")

Strictly, the Appel and Haken proof falls outside the purview of this article, in that their use of the computer was more in a "calculating" than a "reasoning" mode. Nevertheless, reactions to their proof from mathematicians are of some interest, because it has been criticized as lacking surveyability and therefore is regarded by some as not a proper proof. One mathematician, F.F. Bonsall, wrote that an argument involving

> computer verification of special cases . . . does not belong to mathematical science at all. . . . We cannot possibly achieve what I regard as the essential element of a proof — our own personal understanding — if part of the argument is hidden away in a box . . . Let us avoid wasting . . . funds on pseudo mathematics with computers and use them instead to support a few real live mathematicians.[181]

Another mathematician, Daniel I.A. Cohen, described the Appel-Haken proof as "computer shenanigans."[182] Their "proof" did not "explain," Cohen says, as any real proof must: "It didn't tell you *why* four is the answer."

> Mathematics is supposed to do exactly that: give you understanding. That's what's wrong with predicate calculus . . . understanding cannot be part of logic . . . there has to be a psychological event that takes place.[183]

Here is not the place to analyze the underpinnings of these different attitudes to what proof is. Suffice it to note that in the criticism of mechanized proofs there are echoes of earlier objections to formalism, notably Poincaré's. Of course, it is not at all clear that automation need mean lack of surveyability. There are, it is true, large mechanized proofs that are hard to survey and that thus have, as DeMillo, Lipton, and Perlis predicted, found it difficult to obtain credibility: notably, some of the proofs developed to satisfy the requirements of the Orange Book.[184] Yet it may well be that, with care in construction and presentation, unsurveyability is not a problem intrinsic to the automation of proof, even in its application to intricate program and hardware verifications.

If, on the other hand, the development of automated theorem proving, and especially its practical use in verification, does lead to significant numbers of formal but in practice unsurveyable proofs, then we may have the basis for dispute not merely over styles of proving but over proof itself. One such dispute — over the proof of the correctness of the design of the Viper microprocessor — has already reached the point of litigation (although surveyability was only a marginal issue in the dispute). If those involved have to choose between formality and surveyability, then to understand the different preferences of different actors will be a most interesting exercise in the sociology of mathematical knowledge. ■

Acknowledgments

This research was supported by the UK Economic and Social Research Council (ESRC) under a grant for "Studies in the Sociology of Proof" (R000234031). It also draws upon work supported by the ESRC under the Programme in Information and Communication Technologies; by the Science and Engineering Research Council (SERC), grant GR/J58619; and by the ESRC/SERC Joint Committee (GR/H74452). I am particularly grateful for the help of the research fellows on those projects, Tony Dale, Eloína Peláez, Margaret Tierney, and Garrel Pottinger, and of Alan Bundy, my fellow grant holder on the first grant listed above. Thanks, too, to those who commented on parts, or the entirety, of earlier drafts of this article, especially Stuart Anderson, Bob Boyer, Alan Bundy, Robert Constable, Tony Dale, Don Good, Cliff Jones, Donald Loveland, Robin Milner, Larry Paulsen, Randy Pollack, Alan Robinson, Larry Wos, and an anonymous referee for *IEEE Annals of the History of Computing*. Responsibility for remaining errors and misinterpretations is, however, entirely mine!

References

1. M. Davis, "The Prehistory and Early History of Automated Deduction," in *Automation of Reasoning: Classical Papers on Computational Logic*, J. Siekmann and G. Wrightson, eds., Springer, Berlin, 1983, Vol. 1, pp. 1-28.

2. L. Wos and L. Henschen, "Automated Theorem Proving, 1965-1970," in *Automation of Reasoning: Classical Papers on Computational Logic*, J. Siekmann and G. Wrightson, eds., Springer, Berlin, 1983, Vol. 2, pp. 1-24.

3. D.W. Loveland, "Automated Theorem Proving: A Quarter Century Review," *Contemporary Mathematics*, Vol. 29, 1984, pp. 1-45.

4. D.J. O'Leary, "*Principia Mathematica* and the Development of Automated Theorem Proving," in *Perspectives on the History of Mathematical Logic*, T. Drucker, ed., Boston, Birkhäuser, 1991, pp. 48-53.

5. G. Lolli, *La Macchina e le Dimostrazioni: Matematica, Logica e Informatica*, il Mulino, Bologna, 1987.

6. T.L. Heath, *The Thirteen Books of Euclid's Elements*, Cambridge Univ. Press, Cambridge, 1908.

7. Heath, *Euclid's Elements*, p. 155.

8. G. Leibniz, "Elements of a Calculus," in *Leibniz: Logical Papers — A Selection*, G.H.R. Parkinson, ed., Clarendon Press, Oxford, 1966, pp. 17-24, quotation on p. 18.

9. G. Frege, *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*, Nebert, Halle, 1879.

10. A.N. Whitehead and B. Russell, *Principia Mathematica* (second edition), Cambridge Univ. Press, Cambridge, 1925, Vol. 1, pp. 94-95 and 132.

11. K. Gödel, "Über Vollständigkeit und Widerspruchsfreiheit," *Ergebnisse eines mathematischen Kolloquiums*, Vol. 3, 1931, pp. 12-13.

12. K. Gödel, "Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I," *Monatshefte für Mathematik und Physik*, Vol. 38, 1931, pp. 173-198.

13. Lolli, *La Macchina e le Dimostrazioni*, pp. 13-25.

14. H. Mehrtens, Symbolische Imperative: Zu Natur und Beherrschungsprogramm des wissenschaftlichen Moderne. In *Die Modernisierung moderner Gesellschaften: Verhandlungen des 25. Deutschen Soziologentages in Frankfurt am Main 1990*, Campus, Frankfurt am Main, 1991, pp. 604-616.

15. H. Poincaré, *Science et Méthode*, Flammarion, Paris, 1908, p. 157 (my translation).

16. H. Mehrtens, *Moderne Sprache Mathematik: Eine Geschichte des Streits um die Grundlagen der Disziplin und des Subjekts formaler Systeme*, Suhrkamp, Frankfurt am Main, 1990.

17. A.M. Turing, "On Computable Numbers, with an Application to the *Entscheidungsproblem*," *Proc. London Mathematical Soc.*, Series 2, Vol. 42, 1939, pp. 230-265.

18. N. Bourbaki, *Elements of Mathematics: Theory of Sets*, Addison Wesley, Reading, Mass., 1968, p 11.

19. R.S. Boyer and J.S. Moore, "A Theorem Prover for a Computational Logic," keynote address to 10th Conf. on Automated Deduction, 1990, p. 8.

20. P.H. Nidditch, *Introductory Formal Logic of Mathematics*, Univ. Tutorial Press, London, 1957, pp. v, 1, and 6-7.

21. W.S. Jevons, "On the Mechanical Performance of Logical Inference," *Philosophical Trans. of the Royal Soc.*, Vol. 160, 1870, pp. 497-518.

22. P. McCorduck, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*, Freeman, San Francisco, 1979, pp. 104, 145.

23. H. Simon, *Models of My Life*, Basic Books, New York, chap. 13.

24. H. Simon, interviewed by A. Dale, Pittsburgh, Apr. 21, 1994.

25. McCorduck, *Machines Who Think*, p. 116.

26. A. Newell, J.C. Shaw, and H. Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristic," *Proc. Western Joint Computer Conf.*, Vol. 15, 1957, pp. 218-239, quotations on p. 219.

27. Newell, Shaw, and Simon, "Empirical Explorations of the Logic Theory Machine," p. 221.

28. O'Leary, "*Principia Mathematica* and the Development of Automated Theorem Proving," p. 52.

29. B. Russell, letter to Herbert Simon, Nov. 2, 1956, quoted in O'Leary "*Principia Mathematica* and the Development of Automated Theorem Proving," p. 52.

30. McCorduck, *Machines Who Think*, p. 142.

31. Newell, Shaw, and Simon, "Empirical Explorations of the Logic Theory Machine," p. 220.

32. Simon, interviewed by Dale.

33. D. Crevier, *AI: The Tumultuous History of the Search for Artificial Intelligence*, Basic Books, New York, 1993, pp. 52-54.

34. H.L. Gelernter and N. Rochester, "Intelligent Behavior in Problem-Solving Machines," *IBM J. Research and Development*, Vol. 2, 1958, pp. 336-345.

35. H. Gelernter, J.R. Hansen, and D.W. Loveland, "Empirical Explorations of the Geometry-Theorem Proving Machine," in *Computers and Thought*, E.A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, 1963, pp. 153-163, quotation on p. 153.

36. Simon, interviewed by Dale.

37. H. Wang, "Toward Mechanical Mathematics," *IBM J. Research and Development*, Vol. 4, 1960, pp. 2-21, quotation on p. 3.

38. Wang, "Toward Mechanical Mathematics," p. 18.

39. K. Gödel, "Die Vollständigkeit der Axiome des logischen Funktionenkalküls," *Monatshefte für Mathematik und Physik*, Vol. 37, 1930, pp. 349-360.

40. B. Dreben, "On the Completeness of Quantification Theory," *Proc. National Academy of Sciences*, Vol. 38, 1952, pp. 1047-1052.

41. W.V. Quine, "A Proof Procedure for Quantification Theory," *J. Symbolic Logic*, Vol. 20, No. 2, June 1955, pp. 141-149.

42. D. Prawitz, "Preface to 'A Mechanical Proof Procedure . . .'," in *Automation of Reasoning: Classical Papers on Computational Logic*, J. Siekmann and G. Wrightson, eds., Springer, Berlin, 1983, Vol. 1, pp. 200-201.

43. 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 krajów slowianskich, Warszawa 1929*, Sklad Glówny, Warsaw, 1930, pp. 92-101.

44. M. Davis, interviewed by A.J. Dale, New York, Apr. 29, 1994.

45. M. Davis, "A Computer Program for Presburger's Algorithm," paper presented to Cornell Summer Institute for Symbolic Logic, 1957, in *Automation of Reasoning: Classical Papers on Computational Logic*, J. Siekmann and G. Wrightson, eds., Springer, Berlin, 1983, Vol. 1, pp. 41-48.

46. Wang, "Toward Mechanical Mathematics."

47. H. Wang, "Proving Theorems by Pattern Recognition," *Comm. ACM*, Vol. 4, No. 3, 1960, pp. 229-243.

48. Wang, "Toward Mechanical Mathematics," p. 4.

49. E.W. Beth, "On Machines which Prove Theorems," *Simon Stevin Wis- en Natur-kundig Tijdschrift*, Vol. 32, 1958, pp. 49-60, quotation on p. 59.

50. A. Robinson, "Proving a Theorem (as Done by Man, Logician, or Machine)," in *Automation of Reasoning: Classical Papers on Computational Logic*, J. Siekmann and G. Wrightson, eds., Springer, Berlin, 1983, Vol. 1, pp. 74-76, quotation on p. 74.

51. P.C. Gilmore, "A Proof Method for Quantification Theory: Its Justification and Realization," *IBM J. Research and Development*, Vol. 4, 1960, pp. 28-35.

52. Wang, "Toward Mechanical Mathematics."

53. D. Prawitz, H. Prawitz, and N. Voghera, "A Mechanical Proof Procedure and its Realization in an Electronic Computer," *J. ACM*, Vol. 7, 1960, pp. 102-128.

54. Gilmore, "Proof Method for Quantification Theory."

55. J.A. Robinson, interviewed by A.J. Dale, Edinburgh, Feb. 15 and 16, 1994.

56. W.F. Miller, letter to author, June 18, 1993.

57. M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *J. ACM*, Vol. 7, No. 3, 1960, pp. 201-215.

58. D. Prawitz, "An Improved Proof Procedure," *Theoria*, Vol. 26, 1960, pp. 102-139.

59. J.A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *J. ACM*, Vol. 12, No. 1, 1965, pp. 23-41, quotations on pp. 23 and 24.

60. Robinson, interviewed by Dale.

61. J.A. Robinson, "Logic and Logic Programming," *Comm. ACM*, Vol. 35, No. 3, 1992, pp. 41-65.

62. J. Fleck, "Development and Establishment in Artificial Intelligence," in *Scientific Establishments and Hierarchies: Sociology of the Sciences Yearbook, Vol. VI*, N. Elias, H. Martins, and R. Whitley, eds., Reidel, Dordrecht, 1982, pp. 169-217.

63. D.W. Loveland, *Automated Theorem Proving: A Logical Basis*, North-Holland, Amsterdam, 1978.

64. Robinson, "Logic and Logic Programming," p. 45.

65. Crevier, *AI*, p. 64.

66. S. Papert, interviewed by A.J. Dale, Cambridge, Mass., June 7, 1994.

67. M. Minsky, S. Papert, and staff, *Proposal to ARPA for Research on Artificial Intelligence at M. I. T., 1971-1972*, MIT Artificial Intelligence Memo No. 245, 1971, pp. 13-14.

68. P.J. Hayes, "A Critique of Pure Treason," *Computational Intelligence*, Vol. 3, 1987, pp. 179-185, quotations on p. 183.

69. R.A. Kowalski, "The Early Years of Logic Programming," *Comm. ACM*, Vol. 31, No. 1, 1988, pp. 38-42.

70. Sir J. Lighthill, "Artificial Intelligence: A General Survey," in *Artificial Intelligence: A Paper Symposium*, Science Research Council, London, 1973, p. 10.

71. S.A. Cook, "The Complexity of Theorem-Proving Procedures," *Proc. Third Ann. ACM Symp. Theory of Computing*, ACM, New York, 1971, pp. 151-158.

72. C. Cherniak, "Computational Complexity and the Universal Acceptance of Logic," *J. Philosophy*, Vol. 81, No. 12, Dec. 1984, pp. 739-758.

73. W.W. Bledsoe, "I Had a Dream: AAAI Presidential Address, 19 Aug. 1985," *AI Magazine*, Vol. 7, 1986, pp. 57-61, quotation on p. 57.

74. A.O. Boyer and R.S. Boyer, "A Biographical Sketch of W.W. Bledsoe," in *Automated Reasoning: Essays in Honor of Woody Bledsoe*, R.S. Boyer, ed., Kluwer, Dordrecht, 1991, pp. 1-29.

75. W.W. Bledsoe, "Non-Resolution Theorem Proving," *Artificial Intelligence*, Vol. 9, 1977, pp. 1-35, quotation on pp. 2-3.

76. C. Hewitt, *Description and Theoretical Analysis (using Schemas) of Planner: A Language for Proving Theorems and Manipulating Models in a Robot*, PhD thesis, MIT, Cambridge, 1971.

77. Bledsoe, "Non-Resolution Theorem Proving," p. 12.

78. Loveland, "Automated Theorem Proving," p. 24.

79. W.W. Bledsoe, "Splitting and Reduction Heuristics in Automatic Theorem Proving," *Artificial Intelligence*, Vol. 2, 1971, pp. 55-77, quotations on p. 55.

80. A.P. Morse, *A Theory of Sets*, Academic Press, New York, 1965.

81. Bledsoe, "Splitting and Reduction Heuristics," p. 66.

82. W.W. Bledsoe, R.S. Boyer, and W.H. Henneman, "Computer Proofs of Limit Theorems," *Artificial Intelligence*, Vol. 3, 1972, pp. 27-60, quotations on pp. 28 and 35, and details of the procedure on pp. 35-38.

83. L.T. Wos et al., "The Concept of Demodulation in Theorem Proving," *J. ACM*, Vol. 14, No. 4, Oct. 1967, pp. 698-709.

84. G. Robinson and L. Wos, "Paramodulation and Theorem-Proving in First-Order Theories with Equality," *Machine Intelligence*, Vol. 4, 1969, pp. 135-150.

85. S. Winker and L. Wos, "Automated Generation of Models and Counterexamples and its Application to Open Questions in Ternary Boolean Algebra," *Proc. Eighth Int'l Symp. Multiple-valued Logic*, Rosemont, Ill., IEEE and ACM, New York, 1978, pp. 251-256.

86. S. Winker, L. Wos, and E.L. Lusk, "Semigroups, Antiautomorphism, and Involutions: A Computer Solution to an Open Problem, I," *Mathematics of Computation*, Vol. 37, 1981, pp. 533-545.

87. S. Winker, "Generation and Verification of Finite Models and Counterexamples Using an Automated Theorem Prover Answering Two Open Questions," *J. ACM*, Vol. 29, No. 2, 1982, pp. 273-284.

88. L. Wos, "Solving Open Questions with an Automated Theorem-Proving Program," in *Proc. Sixth Conf. Automated Deduction*, New York, June 7-9, 1982, D.W. Loveland, ed., Springer, New York, 1982, pp. 1-31.

89. L. Wos et al., "Questions Concerning Possible Shortest Single Axioms for the Equivalential Calculus: An Application of Automated Theorem Proving to Infinite Domains," *Notre Dame J. Formal Logic*, Vol. 24, No. 2, 1983, pp. 205-223.

90. T.S. Kuhn, *The Structure of Scientific Revolutions*, 2nd ed., Univ. of Chicago Press, Chicago, 1970.

91. L. Wos, "Transition to the Future," *J. Automated Reasoning*, Vol. 9, No. 1, 1992, p. iii.

92. J.R. Guard et al., "Semi-Automated Mathematics," *J. ACM*, Vol. 16, 1969, pp. 49-62, quotation on p. 49.

93. R. Bumcrot, "On Lattice Complements," *Proc. Glasgow Mathematical Assn.*, Vol. 7, 1965, pp. 22-23.

94. Guard et al., "Semi-Automated Mathematics," p. 58.

95. N.G. de Bruijn, "Automath: A Language for Mathematics," in *Automation of Reasoning: Classical Papers on Computational Logic*, J. Siekmann and G. Wrightson, eds., Springer, Berlin, 1983, Vol. 2, pp. 159-200, quotation on p. 159.

96. E.G.H. Landau, *Grundlagen der Analysis*, Akademische Verlag, Leipzig, 1930.

97. L.S. van Benthem Jutting, *Checking Landau's "Grundlagen" in the Automath System*, PhD thesis, Technische Hogeschool, Eindhoven, 1977.

98. P.J. Davis and R. Hersh, "Rhetoric and Mathematics," in *The Rhetoric of the Human Sciences*, J.S. Nelson et al., eds., London, Univ. of Wisconsin Press, 1987, pp. 53-68, quotation on pp. 63-64.

99. P. Naur and B. Randell, eds., *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968*, NATO, Brussels, 1969.

100. E. Peláez, *A Gift from Pandora's Box: The Software Crisis*, PhD thesis, Univ. Edinburgh, 1988.

101. H.H. Goldstine and J. von Neumann, "Planning and Coding of Problems for an Electronic Computing Instrument," in *John von Neumann: Collected Works*, Vol. V, A.H. Taub, ed., Pergamon, Oxford, 1963, pp. 80-151.

102. A.M. Turing, "Checking a Large Routine," in *Report of a Conference on High Speed Automatic Calculating Machines*, Cambridge Univ. Mathematical Laboratory, Cambridge, 1949, pp. 67-69, reprinted in F.L. Morris and C.B. Jones, "An Early Program Proof by Alan Turing," *Annals of the History of Computing*, Vol. 6, No. 2, Apr. 1984, pp. 139-143.

103. C.B. Jones, *The Search for Tractable Ways to Reason about Programs*, Univ. Manchester, Dept. of Computer Science, UMCS-92-4-4, 1992, p. 8.

104. J. McCarthy, "Towards a Mathematical Science of Computation," in *Information Processing 1962: Proc. IFIP Congress 62*, C.M. Popplewell, ed., North Holland, Amsterdam, 1963, pp. 21-28.

105. P. Naur, "Proof of Algorithms by General Snapshots," *BIT*, Vol. 6, 1966, pp. 310-316.

106. R.W. Floyd, "Assigning Meanings to Programs," *Mathe-*

matical Aspects of Computer Science: Proc. Symposia in Applied Mathematics, Vol. 19, 1967, pp. 19-32.

107. C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," *Comm. ACM*, Vol. 12, No. 10, 1969, pp. 576-580 and 583.

108. J. McCarthy and J. Painter, "Correctness of a Compiler for Arithmetic Expressions," in *Mathematical Aspects of Computer Science: Proc. Symposia in Applied Mathematics*, Vol. 19, 1967, pp. 33-41.

109. R.L. London, "Computer Programs Can Be Proved Correct," in *Theoretical Approaches to Non-Numerical Problem Solving: Proc. Fourth Systems Symp. at Case Western Reserve Univ.*, Springer, Berlin, 1970, pp. 281-302.

110. J.C. King, *A Program Verifier*, PhD thesis, Carnegie Mellon Univ., 1969, p. 166.

111. King, *Program Verifier*.

112. D.I. Good, *Toward a Man-Machine System for Proving Program Correctness*, PhD thesis, Univ. of Wisconsin, 1970.

113. D.I. Good, personal communication.

114. D.E. Bell and L.J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, Mitre Corp., Bedford, Mass., MTR-2547, 1973.

115. D.I. Good, "Mechanical Proofs about Computer Programs," *Philosophical Trans. of the Royal Soc. of London*, Series A, Vol. 312, 1984, pp. 389-409, quotation on p. 389.

116. W.W. Bledsoe and P. Bruell, "A Man-Machine Theorem-Proving System," in *Advance Papers of Third Int'l Joint Conf. Artificial Intelligence*, W.W. Bledsoe, ed., Vol. 5, Part 1, 1974, pp. 51-72.

117. D.I. Good, "Proof of a Distributed System in Gypsy," in *Formal Specification: Proc. Joint IBM/Univ. of Newcastle upon Tyne Seminar, 7-10 Sept., 1983*, M.J. Elphick, ed., Univ. of Newcastle upon Tyne, Computing Laboratory, 1983, pp. 44-89, quotation on p. 61.

118. D.I. Good, interviewed by E. Peláez, Austin, Tex., May 16, 1991.

119. Good, "Proof of a Distributed System in Gypsy."

120. M. Schaefer, "Symbol Security Condition Considered Harmful," *Proc. 1989 IEEE Symp. Security and Safety*, IEEE CS Press, Los Alamitos, Calif., 1989, pp. 20-46, especially p. 26.

121. Department of Defense, *Trusted Computer System Evaluation Criteria*, National Computer Security Center, Fort Meade, Md., DOD 5200.28-STD, 1985.

122. M.H. Cheheyl et al., "Verifying Security," *Computing Surveys*, Vol. 13, No. 3, 1981, pp. 279-339.

123. C.L. Gold, S.E. Goodman, and B.G. Walker, "Software: Recommendations for an Export Control Policy," *Comm. ACM*, Vol. 23, No. 4, 1980, pp. 199-207, quotations on pp. 199 and 203.

124. G. Pottinger, *Proof Requirements in the Orange Book: Origins, Implementation, and Implications*, typescript, Feb. 11, 1994.

125. Boyer and Moore, "Theorem Prover for a Computational Logic," p. 1.

126. R.S. Boyer, interviewed by A.J. Dale, Austin, Tex., Apr. 8, 1994.

127. J.S. Moore, interviewed by A.J. Dale, Austin, Tex., Apr. 7, 1994.

128. R.S. Boyer and J.S. Moore, *A Computational Logic*, Academic Press, New York, 1979, p. xiii.

129. R. Burstall, "Proving Properties of Programs by Structural Induction," *Computer J.*, Vol. 12, 1969, pp. 41-48.

130. R.S. Boyer and J.S. Moore, "Proving Theorems about LISP Functions," *J. ACM*, Vol. 22, No. 1, 1975, pp. 129-144.

131. Boyer and Moore, *A Computational Logic*, p. xiv.

132. Boyer, interviewed by Dale.

133. Boyer and Moore, "Theorem Prover for a Computational Logic," p. 4.

134. Boyer, interviewed by Peláez.

135. R. Pollack, interviewed by A.J. Dale, Edinburgh, Nov. 15, 1993.

136. Boyer and Moore, "Theorem Prover for a Computational Logic," p. 3.

137. R.S. Boyer and J.S. Moore, *A Computational Logic Handbook*, Academic Press, San Diego, 1988, p. xiii.

138. N. Shankar, *Proof-checking Metamathematics*, PhD thesis, Univ. of Texas at Austin, 1986.

139. R. Milner, interviewed by A.J. Dale, Edinburgh, Nov. 17 and 30, 1993.

140. R. Milner, *Logic for Computable Functions: Description of a Machine Implementation*, Stanford Univ., Stanford Artificial Intelligence Project, AIM-169, 1972.

141. M. Gordon, R. Milner, and C. Wadsworth, *Edinburgh LCF*, Univ. of Edinburgh, Dept. of Computer Science, CSR-11-77, 1977.

142. Stuart Anderson, personal communication.

143. R. Milner, *LCF: A Way of Doing Proofs with a Machine*, Univ. of Edinburgh, Dept. of Computer Science, CSR-41-79, 1979, p. 3.

144. Gordon, Milner, and Wadsworth, *Edinburgh LCF*, p. 5.

145. L.C. Paulson, *Logic and Computation: Interactive Proof with Cambridge LCF*, Cambridge Univ. Press, Cambridge, 1987, p. 209.

146. W.P. van Stigt, *Brouwer's Intuitionism*, North-Holland, Amsterdam, 1990.

147. R.L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System*, Prentice Hall, Englewood Cliffs, N.J., 1986, especially p. 13.

148. F.K. Hanna and N. Daeche, "Specification and Verification Using Higher-Order Logic: A Case Study," in *Formal Aspects of VLSI Design*, G.J. Milne and P.A. Subrahmanyam, eds., North-Holland, Amsterdam, pp. 179-213, quotation on p. 180.

149. P.B. Andrews et al., "Automating Higher-Order Logic," *Contemporary Mathematics*, Vol. 29, 1984, pp. 169-192.

150. M. Gordon, "HOL: A Machine Oriented Formulation of Higher-Order Logic," Univ. of Cambridge, Computer Laboratory, Tech. Report 103, 1985.

151. A. Cohn, "A Proof of Correctness of the Viper Microprocessor: The First Level," Univ. of Cambridge, Computer Laboratory, Tech. Report No. 104, 1987.

152. M. Gordon, interviewed by D. MacKenzie, Cambridge, Jan. 10, 1991.

153. Loveland, "Automated Theorem Proving."

154. W.W. Bledsoe, "A New Method for Proving Certain Presburger Formulas," *Advance Papers of Fourth Int'l Joint Conf. Artificial Intelligence*,Tbilisi, 3-8 Sept. 1975, pp. 15-21.

155. D. Loveland, interviewed by D. MacKenzie, Edinburgh, Aug. 25, 1993.

156. Loveland, "Automated Theorem Proving."

157. A.J. Nevins, "A Human Oriented Logic for Automatic Theorem-Proving," *J. ACM*, Vol. 21, No. 4, 1974, pp. 606-621.

158. Robinson, "Machine-Oriented Logic Based on the Resolution Principle."

159. L. Wos, "Automated Reasoning Answers Open Questions," *Notices of the American Mathematical Soc.*, Vol. 40, No. 1, 1993, pp. 15-26, quotation on p. 16.

160. D.B. Lenat, *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*, PhD thesis, Stanford Univ., 1976.

161. H. Wang, "Computer Theorem Proving and Artificial Intelligence," *Contemporary Mathematics*, Vol. 29, 1984, pp. 49-70, quotation on pp. 49-50.

162. Simon, interviewed by Dale.

163. Wang, "Toward Mechanical Mathematics," p. 3.

164. J.A. Robinson, "Formal and Informal Proofs," in *Automated Reasoning: Essays in Honor of Woody Bledsoe*, R.S. Boyer, ed., Kluwer, Dordrecht, 1991, pp. 267-282.

165. Wang, "Computer Theorem Proving and Artificial Intelligence," p. 60.

166. H.A. Simon and A. Newell, "Heuristic Problem Solving: The Next Advance in Operations Research," *Operations Research*, 6, 1958, pp. 1-10, quotation on p. 7.

167. E.A. Feigenbaum and J. Feldman, eds., *Computers and Thought*, McGraw-Hill, New York, 1963.

168. Anon., "The Talk of the Town," *The New Yorker*, June 11, 1966, pp. 27-28.

169. H.L. Dreyfus, *What Computers Can't Do: The Limits of Artificial Intelligence*, Harper & Row, New York, 1979, p. 293.

170. Wang, "Toward Mechanical Mathematics," p. 3.

171. Dreyfus, *What Computers Can't Do*, p. 300.

172. Wang, "Computer Theorem Proving and Artificial Intelligence," p. 65.

173. Lighthill, "Artificial Intelligence," pp. 10 and 20.

174. Robinson, "Formal and Informal Proofs," p. 269.

175. D. Hilbert, "The Foundations of Mathematics," in *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*, J. van Heijenoort, ed., Harvard Univ. Press, Cambridge, Mass., 1967, pp. 464-479, quotation on p. 465.

176. D. Gorenstein, "The Enormous Theorem," *Scientific American*, Vol. 253, No. 6, Dec. 1985, pp. 92-103, quotation on p. 92.

177. R. DeMillo, interviewed by A.J. Dale, Chicago, Sept. 1994.

178. R. Lipton, interviewed by A.J. Dale, Princeton, N.J., May 4, 1994.

179. R. DeMillo, R. Lipton, and A. Perlis, "Social Processes and Proofs of Theorems and Programs," *Comm. ACM*, Vol. 22, 1979, pp. 271-280, quotation on p. 275.

180. K. Appel and W. Haken, "Every Planar Map is Four Colorable," *Illinois J. Mathematics*, Vol. 21, 1977, pp. 429-567.

181. F.F. Bonsall, "A Down-to-Earth View of Mathematics," *American Mathematical Monthly*, Vol. 89, 1982, pp. 8-14, quotations on pp. 13-14.

182. D.I.A. Cohen, "The Superfluous Paradigm," in *The Mathematical Revolution Inspired by Computing*, J.H. Johnson and M.J. Loomes, eds., Clarendon, Oxford, 1991, pp. 323-329, quotation on p. 328.

183. D.I.A. Cohen, interviewed by A.J. Dale, New York, May 2, 1994.

184. M. Schaefer, interviewed by Garrel Pottinger, Rockville, Md., Mar. 23, 1993.

185. D. MacKenzie, "The Fangs of the VIPER," *Nature*, Vol. 352, No. 6335, Aug. 8, 1991, pp. 467-468.

Donald MacKenzie holds a personal chair in sociology at the University of Edinburgh and researches the sociology and social history of science and technology. His previous work has covered mathematical statistics, inertial guidance and navigation, and supercomputing. In 1992, his work on the last of these fields was awarded the IEEE Life Members' Prize in Electrical History. He has a BSc in applied mathematics and a PhD in science studies, both from the University of Edinburgh.

MacKenzie's address is Department of Sociology, University of Edinburgh, 18 Buccleuch Place, Edinburgh EH8 9LN, Scotland, e-mail: D.MacKenzie@edinburgh.ac.uk.