The Mechanization of Mathematics

Michael Beeson[1]

San José State University, San Jose, CA 95192, USA

Summary. The *mechanization of mathematics* refers to the use of computers to find, or to help find, mathematical proofs. Turing showed that a complete reduction of mathematics to computation is not possible, but nevertheless the art and science of automated deduction has made progress. This paper describes some of the history and surveys the state of the art.

1 Introduction

In the nineteenth century, machines replaced humans and animals as physical laborers. While for the most part this was a welcome relief, there were occasional pockets of resistance. The folk song *John Henry* commemorates an occasion when a man and a machine competed at the task of drilling railroad tunnels through mountains. The "drilling" was done by hammering a steel spike. The machine was steam-powered. The man was an ex-slave, a banjo player with a deep singing voice and a reputation for physical strength and endurance. He beat the machine, drilling fourteen feet to its nine, but it was a Pyrrhic victory, as he died after the effort.

Even before the first computers were developed, people were speculating about the possibility that machines might be made to perform intellectual as well as physical tasks. Alan Turing was the first to make a careful analysis of the potential capabilities of machines, inventing his famous "Turing machines" for the purpose. He argued that if any machine could perform a computation, then some Turing machine could perform it. The argument focuses on the assertion that any machine's operations could be simulated, one step at a time, by certain simple operations, and that Turing machines were capable of those simple operations. Turing's first fame resulted from applying this analysis to a problem posed earlier by Hilbert, which concerned the possibility of mechanizing mathematics. Turing showed that in a certain sense, it is impossible to mechanize mathematics: We shall never be able to build an "oracle" machine that can correctly answer all mathematical questions presented to it with a "yes" or "no" answer. In another famous paper [101], Turing went on to consider the somewhat different question, "Can machines think?". It is a different question, because perhaps machines can think, but they might not be any better at mathematics than humans are; or perhaps they might be better at mathematics than humans are, but not by thinking,

just by brute-force calculation power. These two papers of Turing lie near the roots of the subjects today known as *automated deduction* and *artificial intelligence*.[1]

Although Turing had already proved there were limits to what one could expect of machines, nevertheless machines began to compete with humans at intellectual tasks. Arithmetic came first, but by the end of the century computers could play excellent chess, and in 1997 a computer program beat world champion Garry Kasparov. The New York Times described the match: "In a dazzling hourlong game, the Deep Blue IBM computer demolished an obviously overwhelmed Garry Kasparov and won the six-game man-vs.-machine chess match."[2]

In 1956, Herb Simon, one of the "fathers of artificial intelligence", predicted that within ten years, computers would beat the world chess champion, compose "aesthetically satisfying" original music, and prove new mathematical theorems.[3] It took forty years, not ten, but all these goals were achieved—and within a few years of each other! The music composed by David Cope's programs [33–35] cannot be distinguished, even by professors of music, from that composed by Mozart, Beethoven, and Bach.[4]

In 1976, a computer was used in the proof of the long-unsolved "four color problem'.[5] This did not fulfill Simon's prediction, because the role of the computer was simply to check by calculation the 1476 different specific cases to which the mathematicians had reduced the problem [2,3]. Today this would not cause a ripple; but in 1976 it created quite a stir, and there was serious discussion about whether such a "proof" was acceptable! The journal editors required an independent computer program to be written to check the result. The use of computer calculations to provide "empirical" evidence

[1] One controversy concerns the question whether the limiting theorems about Turing machines also apply to human intelligence, or whether human intelligence has some quality not imitable by a Turing machine (a vital force, free will, quantum indeterminacy in the synapses?) These questions were already taken up by Turing, and were still under discussion (without agreement) by scientific luminaries at the end of the twentieth century [79,80].

[2] After the game, IBM retired Deep Blue, "quitting while it was ahead." Some said that Kasparov lost only because he got nervous and blundered. No rematch was held. In October, 2002, another champion played another computer program: This time it was a draw.

[3] This prediction is usually cited as having been made in 1957, but I believe it was actually first made in 1956 at Simon's inaugural address as President of the Operations Research Society of America.

[4] That level of performance was not demanded by Simon's prediction, and his criterion of "aesthetically satisfying" music was met much earlier. It is interesting that Simon set a lower bar for music than for mathematics and chess, but music turned out to be easier to computerize than mathematics.

[5] This problem asks whether it is possible to color any map that can be drawn on a plane using at most four colors, in such a way that countries with a common border receive different colors.

for mathematical claims has led to "experimental mathematics" and even to reports of the "death of proof" [53]. As Mark Twain said, "the reports of my death are greatly exaggerated".

On December 10, 1996, Simon's prediction came true. The front page of the New York Times carried the following headline: *Computer Math Proof Shows Reasoning Power*. The story began:

> Computers are whizzes when it comes to the grunt work of mathematics. But for creative and elegant solutions to hard mathematical problems, nothing has been able to beat the human mind. That is, perhaps, until now. A computer program written by researchers at Argonne National Laboratory in Illinois has come up with a major mathematical proof that would have been called creative if a human had thought of it. In doing so, the computer has, for the first time, got a toehold into pure mathematics, a field described by its practitioners as more of an art form than a science.

The theorem was proved by the computer program EQP, written by Bill McCune. Before it was proved, it was known as the *Robbins Conjecture*, and people seem reluctant to change the name to "EQP's theorem". It is about certain algebras. An algebra is a set with two operations, written as we usually write addition and multiplication, and another operation called "complement" and written $n(x)$. If an algebra satisfies certain nice equations it is called a Boolean algebra. Robbins exhibited three short simple equations and conjectured that these three equations can be used to axiomatize Boolean algebras; that is, those three equations imply the usual axioms for Boolean algebras. A complete, precise statement of the Robbins conjecture is given in Fig. 1.

EQP solved this problem in a computer run lasting eight days, and using 30 megabytes of memory. The proof it produced, however, was only fifteen lines long and fits onto a single page or computer screen. You sometimes have to shovel a lot of dirt and gravel to find a diamond.[6] Since the proof was easily checkable by humans, there was no flurry of discussion about the acceptability of the proof, as there had been about the four-color problem. (There was, however, a bit of discussion about whether humans had really given this problem their best shot— but indeed, Tarski studied it, and none of the humans who were tempted to be critical were able to find a proof, so these discussions were generally short-lived.) An amusing sidelight: The job was just running in the background and its successful completion was not noticed until a day later!

[6] In 1966 (within ten years of Simon's prediction), a computer program was involved in the solution of an open problem. The user was guiding an interactive theorem prover known as SAM to a proof of a known theorem, and noticed that an equation that had been derived led directly to the answer to a related open question [47]. This event is "widely regarded as the first case of a new result in mathematics being found with help from an automated theorem prover", according to [72], p. 6.

Fig. 1. What exactly is the Robbins Conjecture?

A Boolean algebra is a set A together with binary operations $+$ and $\cdot$ and a unary operation $-$, and elements 0, 1 of A such that the following laws hold: commutative and associative laws for addition and multiplication, distributive laws both for multiplication over addition and for addition over multiplication, and the following special laws: $x + (x \cdot y) = x$, $x \cdot (x + y) = x$, $x + (-x) = 1$, $x \cdot (-x) = 0$. This definition, and other basic information on the subject, can be found in [73]. The Robbins conjecture says that any algebra satisfying the following three equations is a Boolean algebra.

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

Previous work had shown that it is enough to prove the *Huntington equation*:

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

That is, if this equation is satisfied, then the algebra is Boolean. What EQP actually did, then, is come up with a proof that the three Robbins equations imply the Huntington equation. Take out your pencil and paper and give it a try before reading on. You don't need a Ph. D. in mathematics to understand the problem: Just see if the three Robbins equations imply the Huntington equation. It is important to understand the nature of the game: You do not need to "understand" the equations, or the "meaning" of the symbols n, $+$ and $\cdot$. You might be happier if you could think of $+$ as "or", $\cdot$ as "and", and n as "not", but it is completely unnecessary, as you are not allowed to use any properties of these symbols except those given by the equations.

It seems, however, that the intellectual triumph of the computer is by no means as thorough as the physical triumph of the steam drill. The computer has yet to beat a human chess champion reliably and repeatedly, and the number of mathematical theorems whose first proof was found by a computer is still less than 100, though there is some fuzziness about what counts as a theorem and what counts as a computer proof. No graduate student today chooses not to become a mathematician for fear that the computer will prove too difficult a competitor. The day when a computer produces a five hundred page proof that answers a famous open question is not imminent.

Another analogy, perhaps closer than the steam drill, is to mechanizing flight. With regard to mechanizing mathematics, are we now at the stage of Leonardo da Vinci's drawings of men with wings, or at the stage of the Wright brothers? Can we expect the analog of jetliners anytime soon? Airplanes fly, but not quite like birds fly; and Dijkstra famously remarked that the question whether machines can think is like the question, "Can submarines swim?". Since people have no wings, the prospect of machines flying did not create the anxieties and controversies that surround the prospect of machines thinking.

But machines do mathematics somewhat in the way that submarines swim: ponderously, with more power and duration than a fish, but with less grace and beauty.[7]

Acknowledgments. I am grateful to the following people, who read drafts and suggested changes: Nadia Ghamrawi, Marvin Jay Greenberg, Mike Pallesen, and Larry Wos.

2 Before Turing

In this section we review the major strands of thought about the mechanization of mathematics up to the time of Turing. The major figures in this history were Leibniz, Boole, Frege, Russell, and Hilbert. The achievements of these men have been discussed in many other places, most recently in [39], and twenty years ago in [38]. Therefore we will keep this section short; nevertheless, certain minor characters deserve more attention.

Gottfried Leibniz (1646-1716) is famous in this connection for his slogan *Calculemus*, which means "Let us calculate." He envisioned a formal language to reduce reasoning to calculation, and said that reasonable men, faced with a difficult question of philosophy or policy, would express the question in a precise language and use rules of calculation to carry out precise reasoning. This is the first reduction of reasoning to calculation ever envisioned. One imagines a roomful of generals and political leaders turning the crank of Leibniz's machine to decide whether to launch a military attack. It is interesting that Leibniz did not restrict himself to theoretical speculation on this subject—he actually designed and built a working calculating machine, the *Stepped Reckoner*. He was inspired by the somewhat earlier work of Pascal, who built a machine that could add and subtract. Leibniz's machine could add, subtract, divide, and multiply, and was apparently the first machine with all four arithmetic capabilities.[8] Two of Leibniz's Stepped Reckoners have survived and are on display in museums in Munich and Hanover.

George Boole (1815-1864) took up Leibniz's idea, and wrote a book [26] called *The Laws of Thought*. The laws he formulated are now called Boolean

[7] This is the fine print containing the disclaimers. In this paper, "mechanization of mathematics" refers to getting computers to *find* proofs, rather than having them *check* proofs that we already knew, or *store* proofs or papers in a database for reference, or *typeset* our papers, or *send* them conveniently to one another, or *display* them on the Web. All these things are indeed mechanizations of mathematics, in a broader sense, and there are many interesting projects on all these fronts, but we shall limit the scope of our discussions to events in the spirit of John Henry and Big Blue. Moreover, we do not discuss past and present efforts to enable computer programs to make conjectures, or to apply mechanized reasoning to other areas than mathematics, such as verification of computer programs or security protocols, etc.

[8] The abacus does not count because it is not automatic. With Leibniz's machine, the human only turned the crank.

Algebra–yes, the same laws of concern in the Robbins conjecture. Like Leibniz, Boole seems to have had a grandiose vision about the applicability of his algebraic methods to practical problems– his book makes it clear that he hoped these laws would be used to settle practical questions. William Stanley Jevons (1835-1882) heard of Boole's work, and undertook to build a machine to make calculations in Boolean algebra. He successfully designed and built such a machine, which he called the *Logical Piano*, apparently because it was about the size and shape of a small piano. This machine and its creator deserve much more fanfare than they have so far received: This was the first machine to do mechanical inference. Its predecessors, including the Stepped Reckoner, only did arithmetic. The machine is on display at the Museum of Science at Oxford. The design of the machine was described in a paper, *On the Mechanical Performance of Logical Inference*, read before the British Royal Society in 1870.[9]

Gottlob Frege (1848-1925) created modern logic including "for all", "there exists", and rules of proof. Leibniz and Boole had dealt only with what we now call "propositional logic" (that is, no "for all" or "there exists"). They also did not concern themselves with rules of proof, since their aim was to reach truth by pure calculation with symbols for the propositions. Frege took the opposite tack: instead of trying to reduce logic to calculation, he tried to reduce mathematics to logic, including the concept of number. For example, he defined the number 2 to be the class of all classes of the form $\{x, y\}$ with $x \neq y$. Loosely speaking, 2 is the class of all classes with two members; but put that way, the definition sounds circular, which it is not. His major work, the *Begriffschrift* [43], was published in 1879, when Frege was 31 years old. He described it as a symbolic language of pure thought, modeled upon that of arithmetic.

Bertrand Russell (1872-1970) found Frege's famous error: Frege had overlooked what is now known as the Russell paradox.[10] Namely, Frege's rules allowed one to define the class of x such that $P(x)$ is true for any "concept" P. Frege's idea was that such a class was an object itself, the class of objects "falling under the concept P". Russell used this principle to define the class R of concepts that do not fall under themselves. This concept leads to a contradiction known as Russell's Paradox. Here is the argument: (1) if R falls under itself then it does not fall under itself; (2) this contradiction shows that it *does not* fall under itself; (3)therefore by definition it *does* fall under itself after all.

[9] In December 2002, an original copy of this paper was available for purchase from a rare book dealer in New York for a price exceeding $2000.

[10] Russell was thirty years old at the time–about the same age that Frege had been when he made the error. Russell's respectful letter to Frege with the bad news is reprinted in [102], p. 124, along with Frege's reply: "Your discovery of the contradiction caused me the greatest surprise and, I would almost say, consternation, since it has shaken the basis on which I intended to build arithmetic."

Russell (with co-author Whitehead) wrote *Principia Mathematica* [91] to save mathematics from this contradiction. They restricted the applicability of Frege's class-definition principle, thus blocking Russell's paradox, and showed (by actually carrying out hundreds of pages of proofs) that the main lines of mathematics could still be developed from the restricted principle. This work was very influential and became the starting point for twentieth-century logic; thirty years later, when Gödel needed a specific axiom system for use in stating his incompleteness theorem, the obvious choice was the system of *Principia*.

David Hilbert (1862-1943) was one of the foremost mathematicians of the early twentieth century. He contributed to the development of formal logic (rules for reasoning), and then became interested in a two-step reductionist program that combined those of Leibniz and Frege: he would first reduce mathematics to logic, using formal languages, and *then* reduce logic to computation. His plan was to consider the proofs in logic as objects in their own right, and study them as one would study any finite structure, just as mathematicians study groups or graphs. He hoped that we would then be able to give algorithms for determining if a given statement could be proved from given axioms, or not. By consideration of this research program, he was led to formulate the "decision problem" for logic, better known by its German name, the "Entscheidungsproblem". This problem was published in 1928 in the influential logic book by Hilbert and Ackermann [51]. This was the problem whose negative solution made Turing famous; the next section will explain the problem and its solution.

3 Hilbert and the Entscheidungsproblem

The Entscheidungsproblem asks whether there exists a decision algorithm such that:

- It takes two inputs: a finite set of axioms, and a conjecture.
- It computes for a finite time and outputs either a proof of the conjecture from the axioms, or "no proof exists".
- The result is always correct.

Part of the reason for the historical importance of this problem is that it was a significant achievement just to state the problem precisely. What are *axioms* ? What is a *proof*? What is an *algorithm*? Progress on the first two of those questions had been made by Russell and by Hilbert himself. There was an important difference in their approaches, however. Russell worked with proofs and axioms in order to find axioms that were evidently true, and would therefore enable one to derive true (and only true) mathematical theorems. He had in mind one fixed interpretation of his axioms–that is, they were about the one true mathematical universe of classes, if they were about anything at all. In the many pages of *Principia Mathematica*, Russell and Whitehead

never discussed the question of what we would today call the interpretations of their formal theory. Hilbert, on the other hand, understood very well that the same axioms could have more than one interpretation. Hilbert's most well-known work on axiomatization is his book *Foundations of Geometry* [50]. This book provided a careful axiomatic reworking of Euclid from 21 axioms. Hilbert emphasized the distinction between correct reasoning (about points, lines, and planes) and the facts about points, lines, and planes, by saying that if you replace "points, lines, and planes" by "tables, chairs, and beer mugs", the reasoning should still be correct. This seems obvious to today's mathematicians, because the axiomatic approach to mathematics proved so fruitful in the rest of the twentieth century that every student of mathematics is today steeped in this basic idea. But, at the dawn of the twentieth century, this idea seemed radical. The mathematician Poincaré understood Hilbert's point very clearly, as one can see in the following quotation [78], but he thought it antithetical to the spirit of mathematics:

> Thus it will be readily understood that in order to demonstrate a theorem, it is not necessary or even useful to know what it means. We might replace geometry by the reasoning piano imagined by Stanley Jevons, or ... a machine where we should put in axioms at one end and take out theorems at the other, like that legendary machine in Chicago where pigs go in alive and come out transformed into hams and sausages.

The date of that quotation is 1908, almost a decade after *Foundations of Geometry*. But the concept of "proof" was still a bit unclear. The distinction that was still lacking was what we call today the distinction between a *first-order* proof and a *second-order* proof. The axioms of geometry in Hilbert's book included the "continuity axiom", which says that if you have two subsets A and B of a line L, and all the points of A lie to the left[11] of all the points of B, then there exists a point P on L to the right of all points of A not equal to P, and to the left of all points of B not equal to P. This axiom is intended to say that there are no "holes" in a line. For example, if L is the x-axis, and if A is the set of points with $x^2 < 2$, and if B is the set of points with $x > 0$ and $x^2 > 2$, then the axiom guarantees the existence of $x = \sqrt{2}$. But the statement of the axiom mentions not only points, lines, and planes (the objects of geometry) but also *sets* of points. Remember that *Foundations of Geometry* was written before the discovery of Russell's paradox and *Principia*, and apparently Hilbert did not see the necessity of careful attention to the axioms for sets as well as to the axioms for points, lines, and planes. A *second-order* theory or axiomatization is one that, like Hilbert's axiomatization of geometry, uses variables for sets of objects as well as variables for objects. Peano's axioms for number theory are another famous

[11] Hilbert's axioms use a primitive relation "x is between y and z". We can avoid the informal term "lie to the left" using this relation.

example of a second-order axiomatization.[12] Incidentally, Peano's publication [75] was a pamphlet written in Latin, long after Latin had been displaced as the language of scholarship, so that the publication has been viewed as an "act of romanticism". Peano, originally a good teacher, became an unpopular teacher because he insisted on using formal notation in elementary classes; nevertheless, his work eventually became influential, and it is his notation that is used today in logic, not Frege's.

In both these two famous examples, the theories achieve their aim: They uniquely define the structures they are trying to axiomatize. Every system of objects satisfying Hilbert's axioms for plane geometry is isomorphic to the Euclidean plane. Even if we begin by assuming that the system consists of tables, chairs, and beer mugs, it turns out to be isomorphic to the Euclidean plane. Every system of objects satisfying Peano's axioms is isomorphic to the natural numbers. But the second-order nature of these axiom systems is essential to this property. The technical term for this property is that the theory is *categorical*. These are *second-order categorical* theories. The concept of second-order theory versus first-order theory is not easy to grasp, but is very important in understanding the theoretical basis of the mechanization of mathematics, so here goes:

If we require a first-order version of the continuity axiom, then instead of saying "for all sets A and $B\ldots$", the axiom will become many axioms, where A and B are replaced by many different first-order formulas. In other words, instead of being able to state the axiom for *all* sets of points, we will have to settle for *algebraically definable* sets of points. We will still be able to define $\sqrt{2}$, but we will not be able to define π, because π cannot be defined by algebraic conditions. Another way of looking at this situation is to consider systems of "points" that satisfy the axioms. Such systems are called "models". In the case at hand, we have the "real plane" consisting of all points (x, y), and on the other hand, we have the smaller "plane" consisting only of the numbers (x, y) where x and y are solutions of some polynomial equation with integer coefficients. Both these satisfy the first-order axioms of geometry, but the smaller plane lacks the point $(\pi, 0)$ and hence does not satisfy the second-order continuity axiom.

Similarly, in arithmetic, if we do not use variables for sets in stating the induction axiom, we will be able only to "approximate" the axiom by including its specific instances, where the inductive set is defined in the fixed

[12] These famous axioms characterize the natural numbers N as follows: 0 is in N, and if x is in N then the successor x^+ of x is in N, and 0 is not the successor of any number, and if $x^+ = y^+$ then $x = y$. (The successor of 0 is 1, the successor of 1 is 2, etc.) To these axioms Peano added the axiom of *induction*: if X is any set satisfying these properties with X instead of N, then N is a subset of X. The induction axiom is equivalent to the statement that every non-empty set of natural numbers contains a least element, and is also equivalent to the usual formulation of mathematical induction: for sets X of natural numbers, if 0 is in X, and if whenever n is in X so is n^+, then X contains all natural numbers.

language of arithmetic. There are theorems that say a certain equation has no solution in integers, whose proofs require proving a very complicated formula P by induction, as a lemma, where the formula P is too complicated to even be stated in the language of arithmetic–perhaps it requires more advanced mathematical concepts. Just as there exist different models of first-order geometry (in which π does or does not exist), there also exist different models of first-order number theory, some of which are "non-standard", in that the "numbers" of the model are not isomorphic to the actual integers. These non-standard models are more difficult to visualize and understand than a plane that "simply" omits numbers with complicated definitions, because these models contain "numbers" that are not really numbers, but are "extra".

Using modern language, we say that a first-order theory, even one formed by restricting a second-order categorical theory to its first-order instances, generally has many models, not just one. This situation was not clearly understood in the first two decades of the twentieth century,[13] but by 1928, when Hilbert and Ackermann published their monograph on mathematical logic [51], it had become clear at least to those authors. Clarity on this point led directly to the formulation of the Entscheidungsproblem: Since a first-order theory generally has many models, can we decide (given a theory) which formulas are true in all the models? It also led directly to the formulation of the completeness problem: Are the formulas true in all the models exactly those that have proofs from the axioms? The former problem was solved by Turing and Church, the latter by Gödel, both within a few years of the publication of Hilbert-Ackermann. These developments laid the foundations of modern mathematical logic, which in turn furnished the tools for the mechanization of mathematics.

The distinction between second-order and first-order confuses people because it has two aspects: syntax and semantics. A theory which has variables for objects and for sets of those objects (for example integers and sets of integers) is syntactically second-order. We can write down mathematical induction using the set variables. But then, we can still consider this as a first-order theory, in which case we would allow models in which the set variables range over a suitable countable collection of sets of integers, and there would also be models with non-standard integers in which the set variables range over a collection of "subsets of integers" of the model. Or, we can consider it as a second-order theory, in which case we do not allow such models, but only allow models in which the set variables range over *all* subsets of the integers of the model. Whether it is second-order or first-order is determined by what we allow as a "model" of the theory, not by the language in which we express the theory.

[13] See for example [67], Part III for more details on the views of Hilbert and his contemporaries.

4 Turing's negative solution of the Entscheidungsproblem

The developments described above still left the Entscheidungsproblem somewhat imprecise, in that the concept *algorithm* mentioned in the problem had not been defined. Apparently Hilbert hoped for a positive solution of the problem, in which case it would not have been necessary to define "algorithm", as the solution would exhibit a specific algorithm. But a negative solution would have to prove that no algorithm could do the job, and hence it would be necessary to have a definition of "algorithm".

Alan Turing (1912-1954), answered the question "What is an algorithm?" in 1936 [100] by defining Turing machines.[14] He used his definition to show that there exist problems that cannot be solved by any algorithm. The most well-known of these is the halting problem–there exists no Turing machine that takes as inputs a Turing machine M and an input x for M, and determines correctly whether M halts on input x. Indeed, we don't need two variables here: no Turing machine can determine correctly whether M halts at input M.

In that same remarkable 1936 paper [100], Turing applied his new Turing machines to give a negative solution to the Entscheidungsproblem. His solution makes use of the result just mentioned, that the halting problem is not solvable by a Turing machine. We shall describe his solution to the Entscheidungsproblem now, but not the solution to the halting problem, which is covered in any modern textbook on the theory of computation. (The reader who does not already know what a Turing machine is should skip to the next section.) The solution has three steps:

- Write down axioms A to describe the computations of Turing machines.
- Turing machine M halts at input x if and only if A proves the theorem "M halts at input x".
- If we had an algorithm to determine the consequences of axioms A, it would solve the halting problem, contradiction. Hence no such algorithm exists.[15]

[14] Turing "machines" are conceptual objects rather than physical machines. They *could* be built, but in practice the *idea* of these machines is used, rather than physical examples. Such a machine can be specified by a finite list of its parts ("states") and their connections ("instructions"). They work on "inputs" that are represented by symbols on an input device, usually called a "tape". Whenever the tape is about to be used up, an attendant will attach more, so conceptually, the tape is infinite, yet the machine could still be built. Turing's key idea was that the descriptions of the machines can be given by symbols, and hence Turing machines can accept (descriptions of) Turing machines as inputs.

[15] In more detail the argument is this: Suppose some Turing machine K accepts inputs describing axiom sets S and potential theorems B, and outputs 1 or 0

The "computations" referred to in the first step can be thought of as two-dimensional tables. Each row of the table corresponds to the tape of the Turing machine at a given stage in its computation. The next row is the next stage, after one "move" of the machine. There is an extra mark (you can think of a red color) in the cell where the Turing machine head is located at that stage. When we refer to cell (i, j) we mean the j-th cell in the i-th row. The axioms say that such a table T is a computation by machine M if for all the entries in T, the contents of cell $(i+1, j)$ are related to the contents of the three cells $(i, j-1)$, (i, j), and $(i, j+1)$ according to the program of Turing machine M. Although this uses natural numbers (i, j) to refer to the cells of T, only a few basic and easily axiomatizable properties of the numbers are needed for such an indexing. Of course, it takes some pages to fill in all the details of the first two steps, but the basic idea is not complicated once one understands the concepts involved.

Turing's result showed conclusively that it will never be possible to completely mechanize mathematics. We shall never be able to take all our mathematical questions to a computer and get a correct yes-or-no answer. To understand the definitiveness of Turing's result, one needs Gödel's completeness theorem. The completeness theorem identifies the two natural meanings of "logical consequence": P is a logical consequence of A, if P is true in all systems (models) that satisfy axioms A. On the other hand, P should hopefully be a logical consequence of A, if and only if there exists a proof of P from A. This turns out to be the case, and is exactly the content of Gödel's completeness theorem. Therefore, Turing's result means that we shall never be able to take all questions of the form, "does theorem P follow from axioms A?" to a computer and get a guaranteed correct yes or no answer.

5 Church and Gödel

Turing's negative solution of the Entscheidungsproblem was followed in the 1930's by other "negative" results. In 1936, Alonzo Church (1903-1995) invented the lambda-calculus (often written λ-calculus) and used it to give a definition of *algorithm* different from Turing's, and hence an independent solution of the Entscheidungsproblem [29]. He also proved the result we now

according as S proves B or does not prove B. To solve the halting problem, which is whether a given Turing machine M halts at a given input x, we construct the set of axioms A (depending on M) as in the first step. We then construct the sequence of symbols y expressing "M halts at input x". According to step 2, M halts at x if and only if A proves the theorem y. By hypothesis, we can determine this by running Turing machine K at the inputs A and y. If we get 1, then M halts at x, and if we get 0, it does not. If K behaves as we have supposed, this algorithm will solve the halting problem. Since it involves only Turing machines connected by simple steps, it can be done by another Turing machine, contradicting Turing's result on the unsolvability of the halting problem. Hence no such machine K can exist.

summarize in the statement, "Arithmetic is undecidable". Since Peano's axioms are not first-order, the Entscheidungsproblem does not directly apply to them, and one can ask whether there could be an algorithm that takes a first-order statement about the natural numbers as input, and correctly outputs "true" or "false". The Entscheidungsproblem does not apply, since there exists no (finite first-order) system of axioms A whose logical consequences are the statements true in the natural numbers. Church showed that, nevertheless, there is no such algorithm. Church's student Kleene proved the equivalence of the Turing-machine and the λ-calculus definitions of *algorithm* in his Ph. D. thesis, later published in [60].[16]

In 1931, Kurt Gödel [45] proved his famous "incompleteness theorem", which we can state as follows: Whatever system of axioms one writes down in an attempt to axiomatize the truths about the natural numbers, either some false statement will be proved from the axioms, or some true statement will not be proved. In other words, if all the axioms are true, then some true fact will be unprovable from those axioms. Gödel used neither Turing machines nor λ-calculus (neither of which was invented until five years later), but in essence gave a third definition of *algorithm*.[17] The bulk of Gödel's paper is devoted, not to his essential ideas, but to the details of coding computations as integers; although he did not use Turing machines, he still had to code a different kind of computation as integers. Nowadays, when "Ascii codes" used by computers routinely assign a number to each alphabetic character, and hence reduce a line of text to a very long number, using three digits per character, this seems routine. For example, 'a' has the Ascii code 97, 'b' is assigned 98, 'c' gets 99, and so on. Thus "cat" gets the number 099097116. Such encodings can also be used to show that Turing machine computations can be encoded in numbers.

Making use of Turing machines, it is not very difficult to understand the main idea of Gödel's proof. The technical details about coding can be used to construct a number-theoretical formula $T(e, x, y)$ that expresses that e is a code for a Turing machine (a finite set of instructions), and y is a code for a complete (halting) computation by machine e at input x. In other words, "machine e halts at input x" can be expressed by "there exists a y such that $T(e, x, y)$." Now suppose that we had a correct and complete axiomatization A of the true statements of arithmetic. We could then solve the halting problem by the following algorithm: we simultaneously try to prove "machine e does not halt at input e" from the axioms A, and we run

[16] Kleene went on to become one of the twentieth century's luminaries of logic; his [61] is probably the most influential logic textbook ever written, and he laid the foundations of "recursion theory", which includes the subject now known as the theory of computation.

[17] Gödel's definition seemed at the time rather specialized, and (unlike Turing five years later) he made no claim that it corresponded to the general notion of "computable", though that turned out to be true.

machine e at input e to see if it halts. Here "simultaneously" can be taken to mean "in alternating steps." At even-numbered stages, we run e at input e for one more step, and, at odd-numbered stages, we make one more deduction from the axioms A. If e halts at input e, we find that out at some even-numbered stage. Otherwise, by the assumed completeness and correctness of the axioms A, we succeed at some odd-numbered stage to find a proof that e does not halt at input e. But since the halting problem is unsolvable, this is a contradiction; hence no such set of axioms A can exist. That is Gödel's incompleteness theorem.

6 The Possible Loopholes

The results of Turing, Church, and Gödel are commonly called "negative" results in that they show the impossibility of a complete reduction of mathematics or logic to computation. Hilbert's program was a hopeless pipe dream. These famous results seem to close the doors on those who would hope to mechanize mathematics. But we are not completely trapped; there are the following possible "loopholes", or avenues that may still prove fruitful.

- Maybe there exist interesting axiom systems A such that, for that *particular* axiom system, there *does* exist a "decision procedure", that permits us to compute whether a given statement P follows from A or not.
- Maybe there exist interesting algorithms f that take an axiom system A and an input formula P and, *sometimes*, tell us that P follows from A. Even if f is not *guaranteed* to work on *all* P, if it would work on *some* P for which we did not know the answer before, that would be quite interesting.
- Even if such an f worked only for a particular axiom system A of interest, it still might be able to answer mathematical questions that we could not answer before.

These loopholes in the negative results of the thirties allow the partial mechanization of mathematics. It is the pursuit of these possibilities that occupies the main business of this paper.

7 The first theorem-provers

When the computer was still newborn, some people tried to write programs exploiting the loopholes left by Church and Gödel. The first one exploited the possibility of decision procedures. There was already a known decision procedure for arithmetic without multiplication. This is essentially the theory of linear equations with integer variables, and "for all" and "there exists". This theory goes by the name of "Presburger arithmetic", after M. Presburger,

who first gave a decision procedure for it in [82]. It cried out for implementation, now that the computer was more than a thought experiment. Martin Davis took up this challenge [37], and in 1954 his program proved that the sum of two even numbers is even. This was perhaps the first theorem ever proved by a computer program. The computer on which the program ran was a vacuum tube computer known as the "johnniac", at the Institute for Advanced Study in Princeton, which had a memory of 1024 words. The program could use a maximum of 96 words to hold the generated formulas.

In 1955, Newell, Shaw, and Simon wrote a program they called the *Logic Theorist*[74]. This program went through another loophole: it tried to find proofs, even though according to Turing it must fail sometimes. It proved several propositional logic theorems in the system of *Principia Mathematica*. The authors were proud of the fact that this program was "heuristic", by which they meant not only that it might fail, but that there was some analogy between how it solved problems and how a human would solve the same problems. They felt that a heuristic approach was necessary because the approach of systematically searching for a proof of the desired theorem from the given axioms seemed hopeless. They referred to the latter as the "British Museum" algorithm, comparing it to searching for a desired item in the British Museum by examining the entire contents of the museum. According to [38], Alan Newell said to Herb Simon on Christmas 1955, about their program, "Kind of crude, but it works, boy, it works!". In one of Simon's obituaries [66] (he died in 2001 at age 84), one finds a continuation of this story:

> The following January, Professor Simon celebrated this discovery by walking into a class and announcing to his students, "Over the Christmas holiday, Al Newell and I invented a thinking machine." A subsequent letter to Lord Russell explaining his achievement elicited the reply : "I am delighted to know that 'Principia Mathematica' can now be done by machinery. I wish Whitehead and I had known of this possibility before we wasted 10 years doing it by hand." [18]

In 1957, the year of publication of Newell, Shaw, and Simon's report [74], a five week Summer Institute for Symbolic Logic was held at Cornell, attended by many American logicians and some researchers from IBM. At this meeting, Abraham Robinson introduced the idea of Skolem functions [explained below], and shortly after the meeting a number of important advances were made. Several new programs were written that searched more systematically for proofs than the *Logic Theorist* had done. The problem was clearly seen as "pruning" the search, i.e. eliminating fruitless deductions as early as possible. Gelernter's geometry prover [44] used a "diagram" to prune false goals. The mathematical logician Hao Wang wrote a program [103] based on a logical system known as "natural deduction". Wang's program proved all 400 pure predicate-calculus theorems in *Principia Mathematica*. Davis and Putnam

[18] Russell may have had his tongue firmly in cheek.

[40] published a paper that coupled the use of Skolem functions and conjunctive normal form with a better algorithm to determine satisfiability. Over the next several years, these strands of development led to the invention of fundamental algorithms that are still in use. We shall discuss three of these tools: Skolemization, resolution, and unification.

Skolem functions are used to systematically eliminate "there exists". For instance, "for every x there exists y such that $P(x, y)$" is replaced by $P(x, g(x))$, where g is called a "Skolem function". When we express the law that every nonzero x has a multiplicative inverse in the form $x \neq 0 \rightarrow x \cdot x^{-1} = 1$, we are using a Skolem function (written as x^{-1} instead of $g(x)$). Terms are built up, using function and operation symbols, from variables and constants; usually letters near the beginning of the alphabet are constants and letters near the end are variables (a convention introduced by Descartes). Certain terms are distinguished as "propositions"; intuitively these are the ones that should be either true or false if the variables are given specific values. The use of Skolem functions and elementary logical manipulations enables us to express every axiom and theorem in a certain standard form called "clausal form", which we now explain. A *literal* is an atomic proposition or its negation. A *clause* is a "disjunction of literals"; that is, a list of literals separated by "or". Given some axioms and a conjectured theorem, we negate the theorem, and seek a proof by contradiction. We use Skolem functions and logical manipulations to eliminate "there exists", and then we use logical manipulations to bring the axioms and negated goal to the form of a list of clauses, where "and" implicitly joins the clauses. This process is known as "Skolemization." The clausal form contains no "there exists", but it does contain new symbols for the (unknown) Skolem functions. The original question whether the axioms imply the goal is equivalent to the more convenient question whether the resulting list of clauses is contradictory or not.

In automated deduction, it is customary to use the vertical bar to mean "or", and the minus sign to mean "not". An *inference rule* is a rule for deducing theorems from previously-deduced theorems or axioms. It therefore has "premisses" and "conclusions". As an example of an inference rule we mention the rule *modus ponens*, which is already over 2000 years old: from p and "if p then q" infer q. In clausal notation that would be, from p and $-p|q$ infer q. *Resolution* generalizes this rule. In its simplest form it says, from $p|r$ and $-p|q$, infer $r|q$. Even more generally, r and q can be replaced with several propositions. For example, from $p|r|s$ and $-p|q|t$, we can infer $r|s|q|t$. The rule can be thought of as "cancelling" p with $-p$. The cancelled term p does not have to be the first one listed. If we derive p and also $-p$, then resolution leads to the "empty clause", which denotes a contradiction.

The third of the three tools we mentioned is the *unification algorithm*. This was published by J. A. Robinson[89]. Robinson's publication (which contained more than "just" unification) appeared in 1965, but at that time unification was already in the air, having been implemented by others as early

as 1962. See [38] for this history. The purpose of the unification algorithm is to find values of variables to make two terms match. For example: given $f(x, g(x))$ and $f(g(c), z)$, we find $x = g(c)$, $z = g(g(c))$ by applying unification. The input to the algorithm is a pair of terms to be unified. The output is a substitution; that is, an assignment of terms to variables. We shall not give the details of the unification algorithm here; they can be found in many books, for example in [25], Ch. 17, or [5], pp. 453 *ff.*

Combining resolution and unification, we arrive at the following rule of inference: Suppose that p and s can be unified. Let $*$ denote the substitution found by the unification algorithm. Then from $p|q$ and $-s|r$ infer $q^*|r^*$ provided $p^* = s^*$. This rule is also commonly known as "resolution"–in fact, resolution without unification is only of historical or pedagogical interest. Resolution is *always* combined with unification. J. A. Robinson proved [89] that this rule is *refutation complete*. That means that if a list of clauses is contradictory, there exists a proof of the empty clause from the original list, using resolution as the sole rule of inference.[19]

The basic paradigm for automated deduction then was born: Start with the axioms and negated goal. Perform resolutions (using unification) until a contradiction is reached, or until you run out of time or memory. The modern era in automated deduction could be said to have begun when this paradigm was in place.[20] One very important strand of work in the subject since the sixties has been devoted to various attempts to prevent running out of time or memory. These attempts will be discussed in the section "Searching for proofs" below.[21]

[19] We have oversimplified in the text. The resolution rule as we have given it does not permit one to infer $p(z)$ from $p(x)|p(y)$. Either the resolution rule has to be stated a bit more generally, as Robinson did, or we have to supplement it with the rule called *factoring*, which says that if A and B can be unified, and $*$ is the substitution produced by the unification algorithm, we can infer $A*$.

[20] There were several more attempts to write programs that proved theorems "heuristically", to some extent trying to imitate human thought, but in the end these programs could not compete with an algorithmic search.

[21] It is true that several other approaches have been developed, and have succeeded on some problems. We note in particular the successes of ACL2 [20] and RRL [59] on problems involving mathematical induction, and regret that our limited space and scope do not permit a fuller discussion of alternative approaches. The author is partial to approaches derived from the branch of mathematical logic known as "proof theory"; in the USSR this approach was followed early on, and an algorithm closely related to resolution was invented by Maslov at about the same time as resolution was invented. A theorem-prover based on these principles was built in Leningrad (1971). See [68] for further details and references.

8 Kinds of Mathematical Reasoning

In this section, we abandon the historical approach to the subject. Instead, we examine the mechanization of mathematics by taking inventory of the mathematics to be mechanized. Let us make a rough taxonomy of mathematics. Of course librarians and journal editors are accustomed to classifying mathematics by subject matter, but that is not what we have in mind. Instead, we propose to classify mathematics by the *kind of proofs* that are used. We can distinguish at least the following categories:

- Purely logical
- Simple theory, as in geometry (one kind of object, few relations)
- Equational, as in the Robbins problem, or in group or ring theory.
- Uses calculations, as in algebra or calculus
- Uses natural numbers and mathematical induction
- Uses definitions (perhaps lots of them)
- Uses a little number theory and simple set theory (as in undergraduate algebra courses)
- Uses inequalities heavily (as in analysis)

Purely logical theorems are more interesting than may appear at first blush. One is not restricted to logical systems based on resolution just because one is using a theorem-prover that works that way. There are hundreds of interesting logical systems, including various axiom systems for classical propositional logic, multi-valued logic, modal logic, intuitionistic logic, etc. All of these can be analyzed using the following method. We use a predicate $P(x)$ to stand for "x is provable". We use $i(x, y)$ to mean x implies y. Then, for example, we can write down $-P(x)|-P(i(x, y))|P(y)$ to express "if x and $i(x, y)$ are provable, so is y." When (a commonly-used variant of) resolution is used with this axiom, it will have the same effect as an inference rule called "condensed detachment" that has long been used by logicians. We will return to this discussion near the end of the paper, in the section on "Searching for proofs".

Euclidean geometry can be formulated in a first-order theory with a simple, natural set of axioms. In fact, it can be formulated in a theory all of whose variables stand for points; direct references to lines and planes can be eliminated [97]. But that is not important—we could use unary predicates for points, lines, and planes, or we could use three "sorts" of variables. What we cannot do in such a theory is mention arbitrary sets of points; therefore, the continuity axiom (discussed above) cannot be stated in such a theory. We can state some instances of the continuity axiom (for example, that a line segment with one end inside a circle and one end outside the circle must meet the circle); or we could even consider a theory with an *axiom schema* (infinitely many axioms of a recognizable form) stating the continuity axiom for all first-order definable sets. But if we are interested in Euclid's propositions,

extremely complex forms of the continuity axiom will not be necessary–we can consider a simple theory of geometry instead. It will not prove all the theorems one could prove with the full first-order continuity axiom, but would be sufficient for Euclid. On the other hand, if we wish to prove a theorem about all regular n-gons, the concept of natural number will be required, and proofs by mathematical induction will soon arise. In first-order geometry, we would have one theorem for a square, another for a pentagon, another for a hexagon, and so on. Of course not only Euclidean, but also non-Euclidean geometry, can be formulated in a first-order theory. I know of no work in automated deduction in non-Euclidean geometry, but there exists at least one interesting open problem in hyperbolic geometry whose solution might be possible with automated deduction.[22]

Another example of a simple theory is ring theory. Ring theory is a subject commonly taught in the first year of abstract algebra. The "ring axioms" use the symbols $+$ and $*$, and include most of the familiar laws about them, except the "multiplicative inverse" law and the "commutative law of multiplication", $x * y = y * x$. Many specific systems of mathematical objects satisfy these laws, and may or may not satisfy additional laws such as $x * y = y * x$. A system of objects, with two given (but possibly arbitrarily defined) operations to be denoted by the symbols $+$ and $*$, is called a *ring* if all the ring axioms hold when the variables range over these objects and $+$ and $*$ are interpreted as the given operations. In ring theory, one tries to prove a theorem using only the ring axioms; if one succeeds, the theorem will be true in all rings. However, in books on ring theory one finds many theorems about rings that are not formulated purely in the language of ring theory. These theorems have a larger context: they deal with rings and subrings, with homomorphisms and isomorphisms of rings, and with matrix rings. Homomorphisms are functions from one ring to another that preserve sums and products; isomorphisms are one-to-one homomorphisms; subrings are subsets of a ring that are rings in their own right; matrix rings are rings whose elements are matrices with coefficients drawn from a given ring. Thus passing from a ring R to the ring of n by n matrices with coefficients in R is a method of constructing one ring from another. If, however, we wish to consider such rings of matrices for any n, then the concept of natural number enters again, and we are beyond the simple theory level. Also, if we wish to formulate theorems about arbitrary subrings of a ring, again we have a theory that (at least on the face of it) is second-order. A recent master's thesis [54] went through a typical

[22] The open problem is this: Given a line L and a point P not on L, prove that there exist a pair of *limiting parallels* to L through P. The definition of limiting parallel says that K and R form a pair of limiting parallels to L through P if one of the four angles formed at P by K and R does not contain any ray that does not meet L. It is known that limiting parallels exist, but no first-order proof is known, and experts tell me that producing a first-order proof would be worth a Ph. D.

algebra textbook [56], and found that of about 150 exercises on ring theory, 14 could be straightforwardly formalized in first-order ring theory. One more could be formulated using a single natural-number variable in addition to the ring axioms. The rest were more complex. The 14 first-order exercises, however, could be proved by the theorem-proving program Otter. (Otter is a well-known and widely used modern theorem prover, described in [70], and readily available on the Web.)

A great many mathematical proofs seem to depend on calculations for some of the steps. In fact, typically a mathematical proof consists of some parts that are calculations, and some parts that are logical inferences. Of course, it is possible to recast calculations as logical proofs, and it is possible to recast logical proofs as calculations. But there is an intuitive distinction: a calculation proceeds in a straightforward manner, one step after another, applying obvious rules at each step, until the answer is obtained. While performing a calculation, one needs to be careful, but one does not need to be a genius, once one has figured out what calculation to make. It is "merely a calculation." When finding a proof, one needs insight, experience, intelligence–even genius–to succeed, because the search space is too large for a systematic search to succeed.

It is not surprising that a good deal of progress has been made in mechanizing those parts of proof that are calculations. It may be slightly surprising that methods have been found for automatically discovering new rules to be used for calculations. Furthermore, the relations between the computational parts of proofs and the logical parts have been explored to some extent. However, there is still some work to be done before this subject is finished, as we will discuss in more detail below.

One aspect of mathematics that has not been adequately mechanized at the present time is *definitions*. Let me give a few examples of the use of definitions in mathematics. The concept "f is continuous at x", where f is a real-valued function, has a well-known definition: "for every $\epsilon > 0$ there exists $\delta > 0$ such that for all y with $|y - x| < \delta$, we have $|f(x) - f(y)| < \epsilon$." One important virtue of this definition is that it sweeps the quantifiers "for every" and "there exists" under the rug: We are able to work with continuity in a quantifier-free context. If, for example, we wish to prove that $f(x) = (x + 3)^{100}$ is a continuous function, the "easy way" is to recognize that f is a composition of two continuous functions and appeal to the theorem that the composition of two continuous functions is continuous. That theorem, however, has to be proved by expanding the definitions and using ϵ and δ. This kind of argument does not mesh well with the clausal form paradigm for automated reasoning, because when the definition is expanded, the result involves quantifiers. Theorem-proving programs usually require clausal form at input, and do not perform dynamic Skolemization. Theorems that have been proved about continuity have, therefore, had the definition-expansion and Skolemization performed by hand before the automated deduction pro-

gram began, or have used another paradigm (Gentzen sequents or natural deduction), that does not suffer from this problem, but is not as well-suited to searching for proofs. Merely recognizing $f(x) = (x + 3)^{100}$ as a composition of two functions is beyond the reach of current theorem-provers–it is an application of the author's current research into "second-order unification".

One might well look, therefore, for the *simplest* example of a definition. Consider the definition of a "commutator" in group theory. The notation usually used for a commutator is $[x, y]$, but to avoid notational complexities, let us use the notation $x \otimes y$. The definition is $x \otimes y = x^{-1}y^{-1}xy$, where as usual we leave the symbol $*$ for the group operation unwritten, and assume that association is to the right, i.e. $abc = a(bc)$. We can find problems in group theory that mention commutators but do not need second-order concepts or natural numbers for their formulation or solution. Here we have a single definition added to a simple theory. Now the point is that sometimes we will need to recognize complicated expressions as being actually "nothing but" a commutator. Long expressions become short ones when written using the commutator notation. On the other hand, sometimes we will not be able to solve the problem without using the definition of $x \otimes y$ to eliminate the symbol $\otimes$. That is, sometimes the definition of $x \otimes y$ will be needed in the left-to-right direction, and sometimes in the right-to-left direction. Existing theorem-provers have no method to control equations with this degree of subtlety. Either $\otimes$ will *always* be eliminated, or *never*. This example definition also serves to bring out another point: definitions can be explicit, like the definition of $x \otimes y$ given above, or implicit. Cancellative semigroups are systems like groups except that inverse is replaced by the cancellation law, $xy = xz$ implies $y = z$. We can define $x \otimes y$ in the context of cancellative semigroups by the equation $xy = yx(x \otimes y)$. This is an "implicit definition". If the law holds in a semigroup S, for some operation $\otimes$, we say "S admits commutators."

Consider the following three formulas, taken from [41], and originally from [64].

$$(x \otimes y) \otimes z = x \otimes (y \otimes z) \qquad (1)\ \textit{commutator is associative}$$
$$(x * y) \otimes z = (x \otimes z) * (y \otimes z) \qquad (2)\ \textit{commutator distributes over product}$$
$$(x \otimes y) * z = z * (x \otimes y) \qquad (3)\ \textit{semigroup is nilpotent class 2}$$

These three properties are equivalent in groups (in fact, in cancellative semigroups that admit commutators). One of the points of considering this example is that it is not clear (to the human mathematician) whether one ought to eliminate the definition of $x \otimes y$ to prove these theorems, or not. Otter is able to prove (1) implies (2), (2) implies (3), and (3) implies (1), in three separate runs, in spite of not having a systematic way to handle definitions; but the proofs are not found easily, and a lot of useless clauses are generated along the way.[23]

[23] An example of the use of a definition to help Otter find a proof that it cannot find without using a definition is the proof of the "HCBK-1 problem" found recently

Another interesting problem involving commutators is often an exercise in an elementary abstract algebra course: Show that in a group, the commutator subgroup (consisting of all $x \otimes y$) is a normal subgroup. For the part about normality, we have to show that for all a,b, and c, $c^{-1}(a \otimes b)c$ has the form $u \otimes v$ for some u and v. Otter can find several proofs of this theorem, but the u and v in the first few proofs are not the ones a human would find—although it does eventually find the human proof—and Otter does a fairly large search, while a human does very little searching on this problem.

In mathematics up through calculus, if we do not go deeply into the foundations of the subject but consider only what is actually taught to students, there is mostly calculation. In abstract algebra, most of the work in a one-semester course involves some first-order axioms (groups, rings, etc.), along with the notions of subgroup, homomorphism, isomorphism, and a small amount of the theory of natural numbers. The latter is needed for the concept of "finite group" and the concept of "order of a group". Number theory is needed only (approximately) up to the concept of "a divides b" and the factorization of a number into a product of primes. One proves, for example, the structure theorem for a finite abelian group, and then one can use it to prove the beautiful theorem that the multiplicative group of a finite field is cyclic. These theorems are presently beyond the reach of automated deduction in any honest sense, although of course one could prepare a sequence of lemmas in such a way that the proof could ultimately be found.

However, there is a natural family of mathematical theories that is just sufficient for expressing most undergraduate mathematics. Theories of this kind include a simple theory as discussed above (simple axioms about a single kind of object), and in addition parameters for subsets (but not arbitrary quantification over subsets), variables for natural numbers and mathematical induction, and functions from natural numbers into the objects of the simple theory, so that one can speak about sequences of the objects. These additional features, plus definitions, will encompass most of the proofs encountered in the first semester of abstract algebra. If we add inequalities and calculations to this mix, we will encompass undergraduate analysis, complex analysis, and topology as well.[24]

Of course, there exist branches of mathematics that go beyond this kind of mathematics (e.g. Galois theory or algebraic topology). We propose to not even think about automated deduction in these areas of mathematics. Dealing with the challenges of second-order variables (without quantification),

by Robert Veroff. Although it is too technical to discuss here, the problem is listed as an open problem (which previously had a model-theoretic proof, but no first-order proof) in Appendix 3 of [72] (which also lists other challenges to theorem-proving programs). The solution can be found on Veroff's web page.

[24] There is an obvious objection to the above taxonomy: Functions can be reduced to sets, and numbers can be reduced to sets, so that all of mathematics can be formalized in set theory. This objection will be taken up in the last section of the paper.

definitions, calculations, incorporating natural numbers, sequences, and induction, should keep researchers busy for at least a generation. At that point computers should have more or less the capabilities of an entering Ph. D. student in mathematics. Now, in 2003, they are at approximately freshman level. I do not mean that this progress is inevitable—it will require resources and effort that may not be forthcoming. But it is *possible*.

9 Computer Algebra

"Computer algebra", while a common and descriptive term, is a bit misleading since the subject encompasses calculus and to some extent combinatorics, as well as algebra. Originally computers were viewed as numerical calculators. In fact, when the first checkers-playing program was written in 1948, there was no printer at the IBM research lab that could print anything but numbers, so the output of the checkers playing program had to be coded numerically. But by the late 1950s, realization was spreading that there was such a thing as "symbolic computation", and algebra and calculus were among the first areas attacked. Programs for elementary calculus were soon written, incorporating rules of thumb ("heuristic" is the ten-dollar word for a rule of thumb) for elementary integration. John McCarthy invented LISP, a computer language designed for programming symbolic tasks, and the field of computer algebra began to develop soon afterwards. The subject had, and still has, two sides. One side is the theoretical development of algorithms to solve problems of specific classes. The other side is the efficient, practical implementation of those algorithms in useful programs.

In the sixties and seventies, there were several important theoretical developments in the subject. Although we cannot afford a thorough introduction to computer algebra, we will mention three developments. Consider the following Mathematica input: $Factor[x^{119}-1]$. Mathematica responds instantly with

$$\begin{aligned}
&(-1+x)(1+x+x^2+x^3+x^4+x^5+x^6)(1+x+x^2+x^3+x^4+x^5\\
&+x^6+x^7+x^8+x^9+x^{10}+x^{11}+x^{12}+x^{13}+x^{14}+x^{15}+x^{16})\\
&(1-x+x^7-x^8+x^{14}-x^{15}+x^{17}-x^{18}+x^{21}-x^{22}+x^{24}-x^{25}\\
&+x^{28}-x^{29}+x^{31}-x^{32}+x^{34}-x^{36}+x^{38}-x^{39}+x^{41}-x^{43}+x^{45}\\
&-x^{46}+x^{48}-x^{50}+x^{51}-x^{53}+x^{55}-x^{57}+x^{58}-x^{60}+x^{62}-x^{64}\\
&+x^{65}-x^{67}+x^{68}-x^{71}+x^{72}-x^{74}+x^{75}-x^{78}+x^{79}-x^{81}+x^{82}\\
&-x^{88}+x^{89}-x^{95}+x^{96})
\end{aligned}$$

This was not done by trial and error. It uses algorithms for polynomial factorization[25] that first factor the polynomial mod p for various small primes

[25] For readers unfamiliar with mod p, this means that numbers are always replaced with their remainders after division by p. For example, 3 times 5 is 1 mod 7,

p, and then put the results together cleverly, using a 19th-century theorem known as Hensel's lemma. The factorization mod p can be done quickly, using an algorithm discovered by Berlekamp in 1967. The application of Hensel's lemma to this problem was discovered by Zassenhaus in 1969. Full historical and mathematical details can be found in [57] and [63].

Our second example concerns the integration of elementary functions. An *elementary function* is one that you might encounter in freshman calculus: it is defined using multiplication, addition, subtraction, division, trig functions, exponents, and logarithms. Much effort in freshman calculus goes into rules and methods for computing elementary integrals of elementary functions. However, not every elementary function has an elementary integral. For example, $\int e^{x^2}\,dx$ cannot be expressed in elementary form. Risch [95,96] discovered in 1969 that the trial-and-error methods you may have studied in freshman calculus, such as integration by substitution and integration by parts, can be replaced by a single, systematic procedure, that always works if the integral has *any* elementary answer. A complete exposition of the theory is in [21].

Our third example concerns sets of simultaneous polynomial equations. Say, for example, that you wish to solve the equations

$$\begin{aligned} z + x^4 - 2x + 1 &= 0 \\ y^2 + x^2 - 1 &= 0 \\ x^5 - 6x^3 + x^2 - 1 &= 0 \end{aligned}$$

If you ask Mathematica to solve this set of three equations in three unknowns, it answers (immediately) with a list of the ten solutions. Since the solutions do not have expressions in terms of square roots, they have to be given in the form of algebraic numbers. For example, the first one is $x = \alpha, y = \alpha - 1, z = -1 + 2\alpha$, where α is the smallest root of $-1 + \alpha^2 - 6\alpha^3 + \alpha^5 = 0$. This problem has been solved by constructing what is known as a "Gröbner basis" of the ideal generated by the three polynomials in the original problem. It takes too much space, and demands too much mathematical background, to explain this more fully; see [106], Chapter 8 for explanations. (This example is Exercise 4, p. 201). Although methods (due to Kronecker) were known in the nineteenth century that in principle could solve such problems, the concept of a Gröbner basis and the algorithm for finding one, known as "Buchberger's algorithm", have played an indispensable role in the development of modern computer algebra. These results were in Buchberger's Ph. D. thesis in 1965. Thus the period 1965-70 saw the theoretical foundations of computer algebra laid.

It took some time for implementation to catch up with theory, but as the twenty-first century opened, there were several well-known, widely available programs containing implementations of these important algorithms, as well

because 15 has remainder 1 after division by 7. So $(x+3)(x+5) = x^2 + x + 1 \mod 7$.

as many others. Symbolic mathematics up to and including freshman calculus can thus be regarded as completely mechanized at this point. While one cannot say that the field is complete–every year there is a large international conference devoted to the subject and many more specialized conferences–on the whole the mechanization of computation has progressed much further than the mechanization of proof.

In addition to the well-known general-purpose symbolic computation programs such as Maple, Mathematica, and Macsyma, there are also a number of special-purpose programs devoted to particular branches of mathematics. These are programs such as MAGMA, PARI-GP (algebraic number theory), SnapPea (topology), GAP (group theory), Surface Evolver (differential geometry), etc. These are used by specialists in those fields.

What is the place of computer algebra in the mechanization of mathematics? Obviously there are some parts of mathematics that consist mainly of computations. The fact is that this part of mathematics includes high-school mathematics and first-year calculus as it is usually taught, so that people who do not study mathematics beyond that point have the (mis)-impression that mathematics consists of calculations, and they imagine that advanced mathematics consists of yet more complicated calculations. That is not true. Beginning with the course after calculus, mathematics relies heavily on proofs. Some of the proofs contain some steps that can be justified by calculation, but more emphasis is placed on precisely defined, abstract concepts, and the study of what properties follow from more fundamental properties by logical implication.

10 Decision Procedures in Algebra and Geometry

The "first loophole" allows the possibility that *some* branches of mathematics can be mechanized. An algorithm which can answer any yes-no question in a given class of mathematical questions is called a "decision procedure" for those questions. We will give a simple example to illustrate the concept. You may recall studying trigonometry. In that subject, one considers "trigonometric identities" such as $\cos(2x) = \cos^2 x - \sin^2 x$. The identities considered in trigonometry always have only linear functions in the arguments of the trig functions; for example, they never consider $\sin(x^2)$, although $\sin(2x+3)$ would be allowed. Moreover, the coefficients of those linear functions are always integers, or can be made so by a simple change of variable. The question is, given such an equation, determine whether or not it holds for all values of x (except possibly at the points where one side or the other is not defined, e.g. because a denominator is zero.) You may be surprised to learn that there is a decision method for this class, which we now give. First, use known identities to express everything in terms of sin and cos. If necessary, make a change of variable so that the linear functions in the arguments of sin and cos have integer coefficients. Even though everything is in now in

terms of sin and cos, there could still be different arguments, for example $\sin(2x) - \sin x$. If so, we next use the identities for $\sin(x+y)$ and $\cos(x+y)$ to express everything in terms of $\sin x$ and $\cos x$. The equation is now a rational function of $\sin x$ and $\cos x$. Now for the key step: Make the "Weierstrass substitution" $t = \tan(x/2)$. Then $\sin x$ and $\cos x$ become rational functions of t. Specifically, we have $\sin x = 2t/(1+t^2)$ and $\cos x = (1-t^2)/(1+t^2)$. After this substitution, the equation becomes a polynomial identity in one variable, and we just have to simplify it to "standard form" and see if the two sides are identical or not. All that suffering that you went through in trigonometry class! and a computer can do the job in an instant.

The question is, then, exactly where the borderline between mechanizable theories and non-mechanizable theories lies. It is somewhere between trig identities and number theory, since by Turing and Church's results, we cannot give a decision procedure for number theory. The borderline is in some sense not very far beyond trig identities, since a result of Richardson [85] shows that there is no algorithm that can decide the truth of identities involving polynomials, trig functions, logarithms, and exponentials (with the constant π allowed, and the restriction that the arguments of trig functions be linear removed).[26] Nevertheless, there are many examples of decision procedures for significant bodies of mathematics. Perhaps the most striking is one first explored by Alfred Tarski (1902-1983). The branch of mathematics in question is, roughly speaking, elementary algebra. It is really more than elementary algebra, because "for all" and "there exists" are also allowed, so such questions as the following are legal:

- Does the equation $x^3 - x^2 + 1 = 0$ have a solution between 0 and 1?
- For which values of a and b does the equation $x^4 - ax^3 + b$ take on only positive values as x varies?

The first question has an implicit "there exists an x", and the second has an implicit "for all x". We will call this part of mathematics "Tarski algebra."

"For all" and "there exists" are called "quantifiers". A formula without quantifiers is called "quantifier-free". For example, '$x^2 + 2 = y$' is quantifier-free. A quantifier-free formula might have the form

$$f(x_1, \ldots, x_n) = 0 \ \& \ g(x_1, \ldots, x_n) \geq 0,$$

where f and g are polynomials. More generally, you might have several inequalities instead of just one. Using simple identities, one can show that any quantifier-free formula is equivalent to one in the form indicated. That is, if such formulas are combined with "not", "and", or "or", the result can be equivalently expressed in the standard form mentioned. Tarski's idea is called

[26] The exact borderline for classes of identities still is not known very accurately. For example, what if we keep the restriction that the arguments of trig functions should be linear with integer coefficients, but we allow logarithms and exponentials?

Fig. 2. What is Tarski algebra?

The technical name of this branch of mathematics is the theory of *real-closed fields*. The language for this branch of mathematics has symbols for two operations $+$ and $\cdot$, the inverse operations $-x$ and x^{-1}, the additive and multiplicative identity elements 0 and 1, the ordering relation $<$, and the equality relation $=$. The axioms include the usual laws for $+$ and $\cdot$, and axioms relating $<$ to the operations $+$ and $\cdot$. Defining $0 < x$ as $P(x)$ (P for "positive"), those axioms say that the sum of positive elements is positive and the product of positive elements is positive. These are the axioms of *ordered fields*. The axioms for real-closed fields specify in addition that all positive elements have a square root, and all polynomials of odd degree have a root. One will, of course, need infinitely many axioms to express this without mentioning the concept of "natural number", one axiom for each odd degree. The classical theory of real-closed fields is developed in most algebra textbooks, for example in [65], pp. 273 *ff.*

Tarski algebra escapes the negative results of Church and Gödel because it does not have variables for natural numbers. The variables range over "real numbers"–these are the numbers that correspond to points on a line and are used for coordinates. Even though the variables of Tarski algebra are meant to stand for such numbers, not all individual numbers can be defined in Tarski algebra. In this language, one cannot directly write integers in decimal notation such as 3. Instead of 3, one officially has to write $1 + 1 + 1$. Aside from the inconvenience, one can in effect write any rational number; for example 2/3 is $(1 + 1)(1 + 1 + 1)^{-1}$. But one does not, for example, have a name for π.

elimination of quantifiers. He showed in [97] that every formula in Tarski algebra is equivalent to one without any quantifiers. For example, the question whether $x^2 + bx + c = 0$ has a solution x with $0 \leq x$ appears to involve "there exists an x", but from the quadratic formula we find that the answer can be expressed by a condition involving only b and c, namely, $b^2 - 4c \geq 0$ and either $b \leq 0$ or $c \leq 0$. The quantifier "there exists x" has been eliminated. Several classical results of algebra have a similar flavor. For example, Sturm's theorem from the 1830s [65], p. 276, counts the number of roots of a polynomial in an interval in terms of the alternations of signs in the coefficients. Another classical result is the existence of the *resultant*: If we are given polynomials $f(a, x)$ and $g(a, x)$, we can compute another polynomial $R(a, b)$ called the resultant of f and g, such that $R(a, b) = 0$ if and only if a common solution x can be found for the equations $f(a, x) = 0$ and $g(a, x) = 0$. Again the quantifier "there exists x" has been eliminated. Tarski showed that algebraic methods can always be applied to eliminate "there exists" from algebraic formulas, even ones involving inequalities. The elimination of one quantifier depends essentially on the fact that a polynomial has only finitely many roots, and we can compute the number, the maximum size, and some information about the location of the roots from the coefficients of the polynomial. Applying this procedure again and again, we can strip off one quantifier after another

(from the inside out), eliminating all the quantifiers in a formula with nested quantifiers. We need only deal with "there exists" because "for all" can be expressed as "not there exists x not" . Tarski's procedure is a decision procedure for Tarski algebra, because if we start with a formula that has only quantified variables (so it makes an assertion that should be true or false), after we apply the procedure we get a purely numerical formula involving equations and inequalities of rational numbers, and we can simply compute whether it is true or false.

Descartes showed several centuries earlier that geometry could be reduced to algebra, by the device of coordinates. This reduction, known as analytic geometry, coupled with Tarski's reduction of algebra with quantifiers to computation, yields a reduction of geometry (with quantifiers) to computation. In more technical words: a decision procedure for Euclidean geometry. Thus Hilbert's program, to reduce mathematics to computation, might seem to be achieved for the mathematics of the classical era, algebra and geometry. Tarski's student Szmielew made it work for non-Euclidean (hyperbolic) geometry too [27]. Since the Weierstrass substitution reduces trigonometry to algebra, a decision method for real-closed fields also applies to trigonometry, as long as the arguments of the trig functions are linear.

Tarski's result is regarded as very important. Hundreds of researchers have pursued, and continue to pursue, the lines of investigation he opened. There are two reasons for that: First, his results contrast sharply with Church's and Gödel's, and show that the classical areas of algebra and geometry are not subject to those limiting theorems. Second, there are plenty of open and interesting problems that can be formulated in the theory of real-closed fields, and this has raised the hope that decision procedures implemented on a computer might one day routinely answer open questions. Our purpose in this section is to investigate this possibility.

First, let us give an example of an open problem one can formulate in Tarski algebra. Here is an example from the theory of sphere-packing. This example, and many others, can be found in [32]. The "kissing problem" asks how many n-dimensional spheres can be packed disjointly so that they each touch the unit sphere centered at origin. For $n = 2$ the answer is six (2-spheres are circles). For $n = 3$ the answer is 12. For $n = 4$ the answer is either 24, or 25, but nobody knows which! The problem can be formulated in the theory of real-closed fields, using 100 variables for the coordinates of the centers of the spheres. We simply have to say that each center is at distance 2 from the origin and that each of the 300 pairs of points are at least 2 units apart. Explicitly, we wish to know if there exist $x_1, \ldots, x_{25}, y_1, \ldots, y_{25}, z_1, \ldots, z_{25}$, and $w_1, \ldots, w_{25}$ such that $x_i^2 + y_i^2 + z_i^2 + w_i^2 = 4$ and $(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 + (w_i - w_j)^2 \geq 4$ for $i \neq j$. All we have to do is run Tarski's algorithm on that formula, and the open problem will be answered.[27]

[27] Another interesting sphere-packing problem was open for centuries, until it was solved in 1998. Namely, what is the densest packing of spheres into a large cube

Well then, why is this problem still open? The suspicion may be dawning on you that it isn't so easy to run this procedure on a formula with 100 quantified variables! In the half-century since Tarski's work, researchers have found more efficient algorithms for quantifier elimination, but on the other hand, they have also proved theorems showing that *any* algorithm for quantifier elimination must necessarily run slowly when large numbers of variables are involved. These lines of research now almost meet: the best algorithms almost achieve the theoretical limits. However, it seems that the edge of capability of algorithms is close to the edge of human capability as well, so the possibility that decision procedures might settle an open question cannot be definitively refuted. We therefore review the situation carefully.

First we review the worst-case analyses that show quantifier elimination must run slowly. Fischer and Rabin showed [42] that any algorithm for quantifier elimination in real-closed fields will necessarily require exponential time for worst-case input formulas; that is, time of the order of 2^{dn} where n is the length of the input formula, and d is a fixed constant. This is true even for formulas involving only addition (not multiplication). Later [104,36] a stronger lower bound was proved: sometimes quantifier elimination will require time (and space) of the order 2^{2^n} (double exponential). (See [84] for a survey of results on the complexity of this problem.) Taking $n = 64$ we get a number with more than 10^{18} decimal digits. No wonder the kissing number problem is still open.

Tarski's original algorithm, which was never implemented, was in principle much slower even than double exponential. Tarski's method eliminates one quantifier at a time, and the formula expands in length by a double exponential each time, so the running time cannot be bounded by any tower of exponents. Fischer and Rabin's result was obtained in the fall of 1972, but not published until 1974. In the interim, not knowing that efficient quantifier elimination is impossible, George Collins invented an improved quantifier-elimination method known as *cylindric algebraic decomposition* (CAD) [31]. Actually, according to the preface of [28], Collins had been working on quantifier elimination since 1955, but the 1973 work generalized his method to n variables and hence made it a general quantifier elimination method. Collins's method runs in double exponential time, much better than Tarski's method, and almost best-possible [36]. We knew from Fischer and Rabin's lower bound that there was no hope of a really efficient quantifier elimination algorithm, but the CAD method is much faster than Tarski's or Cohen's methods. The worst case, when the algorithm takes time 2^{2^n}, arises only when there are lots of variables. The algorithm is double exponential in the number of variables,

in 3-space? Kepler conjectured that it is the usual packing used by grocers for stacking oranges, but this was difficult to prove. It can, however, easily be formulated in the theory at hand, so in principle, "all we have to do" is quantifier elimination.

but for a fixed number of variables, the time increases only as some power of the length of the input.[28]

'Moore's law" is the observation, made in 1965 by Gordon Moore, co-founder of Intel, that data density in computers (bits per square centimeter) has been growing exponentially, doubling every 12-18 months, ever since the integrated circuit was invented in 1962. Perhaps incorrectly, many people also use "Moore's law" to refer to the exponential increase in computer speed.[29] One should clearly understand that Moore's law cannot help us much with an algorithm whose running time is double exponential. If the running time is 2^{2^n} and we want to increase n by one, we need a computer that runs 2^{2^n} times faster, as a short calculation will show you: take the ratio of the new running time, $2^{2^{n+1}}$, to the old running time 2^{2^n}. You will get 2^{2^n} when you simplify that ratio. It takes 2^n Moore's law doubling periods just to increase n by one. The import of the double-exponential running time theorems about quantifier elimination is therefore almost as grim as the import of Gödel's theorem. It seems that Fischer, Rabin, Weispfenning, and Davenport have destroyed Tarski's dream as thoroughly as Gödel destroyed Hilbert's.

But people never give up! Maybe there is an escape route. It was discovered in 1992 by Grigorev [46] that if we restrict attention to formulas that only have "there exists", and no "for all", then we can escape the dreaded double-exponential. He gave a decision procedure for this class of formulas which is "only" exponential in the number of variables. This is an important difference, since with an exponential algorithm, if it doesn't run today, perhaps our children will be able to run it; while with a double-exponential running time, our posterity is also doomed to failure. Further improvements since 1992 are described in [8].

A Web search shows that dozens, if not hundreds, of researchers are working on quantifier elimination these days. Although we know that quantifier elimination will take "forever" on large problems, there still might be some interesting open problems within reach—a tantalizing possibility. Hong [52] made improvements to the CAD algorithm, calling his enhanced version "partial CAD", and implemented it in a program called qepcad (quantifier elimination by partial CAD). This program has subsequently been improved upon by many other people, and is publicly available on the Web [22]. At least some of its functionality has been included with *Mathematica* versions 4.1 and 4.2, in the *Experimental* package. But to the best of my knowledge, the algorithms in [46] and [8] have not been implemented.

[28] The interested reader should pursue the CAD algorithm in [28]; we cannot take the space here even to correctly define what a CAD is, let alone describe the original algorithm and its recent improvements in full.

[29] Moore's paper contains a prophetic cartoon showing a "happy home computer" counter between "notions" and "cosmetics". Bill Gates was ten years old at the time.

It seems fair to ask, then, what are the present-day limits of quantifier elimination in algebra? The first interesting example, often used as a benchmark for quantifier elimination, is to find the conditions on a, b, c, d such that the fourth-degree polynomial $x^4 + ax^3 + bx^2 + cx + d$ is positive for all x. That is, to eliminate the quantifier "for all x" in that statement. The answer is a surprisingly complex polynomial in a,b,c, and d. Qepcad does this five-variable problem almost instantaneously. The curious reader can find the answer at the qepcad examples web page [22].

It was fairly easy to create a *Mathematica* notebook with functions defined to facilitate asking simple questions about sphere packing.I defined *TwoSpheres*[M], which uses *InequalityInstance* to ask whether there exist two disjoint spheres of radius 1 in a cube of side $2M$. This is a seven-variable problem, counting M and the coordinates of the centers of the spheres. To make it a six-variable problem, we can put in specific values of M: *Mathematica* answers the query, *TwoSpheres*[7/4] with *True*; and with a suitable variant of the question, it could even exhibit an example of two such spheres. *TwoSpheres*[3/2] returns *False*. The time required is less than a second. The seven-variable problem, with M variable, seems to be too difficult for Mathematica's *CylindricalAlgebraicDecomposition* function. I also tried these problems with the version of qepcad available from [22]; this program was able to express *TwoSpheres*[M] as $M \geq 1$ & $3M^2 - 6M + 2 \geq 0$. The least such M is $1 + 1/\sqrt{3}$, which is the value of M one finds with pencil and paper if one assumes that the centers of the spheres are on the main diagonal of the cube. But the program did not make that assumption–it *proved* that this is the best possible arrangement. Similar queries *ThreeSpheres*[M], for various specific values of M, never returned answers. After several hours I stopped Mathematica; in the version of qepcad from [22], the jobs failed (after several hours) because they ran out of memory. Nine variables is too many in 2003.

Using a double-exponential algorithm, we could expect that with running time varying as 2^{2^n}, if $n = 7$ corresponds to one second, then $n = 8$ should correspond to 2^{128} seconds, or more than 10^{35} years, so a sharp cutoff is to be expected. As calculated above, to increase n from 7 to 8, we need 2^7, or 128, doublings of computer speed. But the kissing problem needs only existential quantifiers, so as discussed above, it sneaks under the wall: we can solve it in "only" exponential time. In that case if 2^7 corresponds to one second, then 2^8 is only two seconds; but to attack the kissing problem we need 100 variables, and 2^{100} corresponds to 2^{93} seconds–about 10^{24} years. [Physicists know a convenient coincidence: that to three significant digits, there are $\pi \times 10^7$ seconds per year.] Even when the work of Grigorev is implemented, it still won't solve the kissing problem. Nevertheless, there may well be open questions with fewer than fifteen variables, so it seems the jury is still out on the potential usefulness of quantifier elimination.

Quantifier elimination has not been the only decision method used in geometry. In 1978, Wu Wen-Tsen pioneered the reduction of geometry to

polynomial ideal theory, introducing "Wu's method" [115]. The idea here is that an important class of geometric theorems can be stated in algebraic language *without using inequalities*. If the theorem can be stated using only conjunctions of equations in the hypothesis and an equation in the conclusion, then it reduces to asking if a certain polynomial lies in the ideal generated by a finite set of other polynomials. Since that time, other methods, based on Gröbner bases, have also been applied to geometric theorem proving. This work has reduced geometric theorem-proving to computer algebra, and when it is applicable, it seems to be more efficient than quantifier elimination. Many interesting theorems in classical plane and solid geometry have been proved this way.

11 Equality Reasoning

The usual axioms for equality, as given in mathematics textbooks, are

$$\begin{aligned} x = x & \qquad \text{reflexivity} \\ x = y \,\&\, y = z \to x = z & \qquad \text{transitivity} \\ x = y \to y = x & \qquad \text{symmetry} \\ x = y \,\&\, \phi(x) \to \phi(y) & \qquad \text{substitutivity} \end{aligned}$$

In this form, these axioms are useless in automated deduction, because they will (in fact even just the first three will) generate a never-ending stream of useless deductions. The "right method" for dealing with equality was discovered three times in the period 1965-1970, independently in [87], [47], and [62]. The approaches had slightly different emphases, although the kernel of the methods is the same. We will first explain the Knuth-Bendix method.

By an "oriented equation" $p = q$ we simply mean a pair of terms separated by an equality sign, so that $p = q$ is not considered the same oriented equation as $q = p$. The idea is that an oriented equation is to be used from left to right only. The oriented equation $x(y + z) = xy + xz$ can be used to change $3 * (4 + 5)$ to $3 * 4 + 3 * 5$, but not vice-versa. The variables can be matched to complicated expressions, although this example shows them matched to constants. Another name for "oriented equation" is "rewrite rule", which conveys an intuition about how rewrite rules are to be used.

Suppose one is given a set E of oriented equations. Given an expression t, we can rewrite t or its subterms using (oriented) equations from E until no more rules can be applied. If this happens, the resulting term is called a "normal form" of t. It need not happen: for example, if E includes the equation $xy = yx$, then we have $ab = ba = ab = \ldots$ *ad infinitum*. If one sequence of rewrites does terminate in a normal form, it still does not guarantee that every such sequence terminates (different subterms can be rewritten at different stages). If, no matter what subterm we rewrite and no matter what equation from E we use, the result always terminates in a normal form, and

if this happens no matter what term t we start with, then E is called *terminating*.

Even this does not guarantee the uniqueness of the normal form of t. That would be guaranteed by the following desirable property of E, known as *confluence*. E is called *confluent* if whenever a term t can be rewritten (using one or more steps) in two different ways to r and s, then there exists another term q such that both r and s can be rewritten to q. This property clearly ensures the uniqueness of normal forms because, if r and s were distinct normal forms of t, it would be impossible to rewrite them as q.

These concepts will be made clear by considering the example of group theory. Consider the usual three axioms of group theory:

$$\begin{aligned} e * x &= x \\ i(x) * x &= e \\ (x * y) * z &= x * (y * z) \end{aligned}$$

This set is not confluent. For example, the term $(i(a) * a) * b)$ can be rewritten to $e * b$ and then to b, but it can also be rewritten to $i(a) * (a * b)$, which cannot be rewritten further. Does there exist a terminating confluent set of equations E extending these three, and such that each of the equations in E is a theorem of group theory? This is an interesting question because if there is, it would enable us to solve the *word problem for group theory*: given an equation $t = s$, does it follow from the three axioms of group theory? If we had a complete confluent set E, we could simply rewrite t and s to their respective unique normal forms, and see if the results are identical. If so, then the equation $t = s$ is a theorem of group theory. If not, it is not a theorem.

The answer for group theory is a set of ten equations. These are the original three, plus the following seven:

$$\begin{aligned} i(x) * (x * y) &= y \\ x * e &= x \\ i(e) &= e \\ i(i(x)) &= x \\ x * i(x) &= e \\ x * (i(x) * y) &= y \\ i(x * y) &= i(y) * i(x) \end{aligned}$$

We call this set of ten equations "complete" because it proves the same equations as the original three axioms, i.e., all the theorems of group theory, but it can do so by using the ten equations only left-to-right, while the original three must be used in both directions to prove the same theorems. In technical language: the ten equations constitute a complete confluent set. That set happens to contain the original three axioms, but that can be viewed as accidental. We would not have cared if the original axioms had themselves

simplified somewhat in the final ten. (Of course, the original axioms of group theory were chosen to be as simple as possible, so it is not really accidental that they are among the ten.)

This solution of the word problem for groups can be vastly generalized. Donald Knuth invented an algorithm, which was implemented by his student Bendix (in FORTRAN IV for the IBM 7094), and has become known as the Knuth-Bendix algorithm since they were the joint authors of [62]. This algorithm was published in 1970, but the work was done considerably earlier. The input is a set E of (unoriented or oriented) equations. The output (if the algorithm terminates) is a set Q of oriented equations (rewrite rules) that is confluent and terminating, and has the same (unoriented) equations as logical consequences as the original set E. However, in general there is no guarantee of termination. One can run this algorithm with the three axioms of group theory as input and obtain the ten-equation system given above as output.

The Knuth-Bendix method is (or can be with appropriate commands) used by most modern theorem-provers. It is integrated with the other methods used in such theorem-provers. Here is an example of an interesting theorem proved by this method: In a ring suppose $x^3 = x$ for all x. Then the ring is commutative. This is proved by starting out with the set E containing the ring axioms and the axiom $xxx = x$. Then the Knuth-Bendix algorithm is run until it deduces $xy = yx$. When that happens, a contradiction will be found by resolution with the negated goal $ab \neq ba$, so the Knuth-Bendix algorithm will not go off *ad infinitum* using the commutative law (as it would if running by itself.) The resulting proof is 52 steps long. Up until 1988 it took ten hours to find this proof; then the prover RRL [59] was able to reduce this time to two minutes. Actually, the hypothesis $x^3 = x$ can be replaced by $x^n = x$ for any natural number $n \geq 2$, and RRL could also do the cases $n = 4, 6, 8 \ldots$, and many other even values of n [117], but it still takes a human being to prove it for all n, because the (only known) proof involves induction on n and the theory of the Jacobsen radical (a second-order concept). The odd cases are still quite hard for theorem provers.

The use of a set of oriented equations to rewrite subterms of a given term is called "demodulation" in the automated theorem proving community, and "rewriting" in an almost separate group of researchers who study rewrite rules for other reasons. A set of oriented equations can be specified by the user of Otter as "demodulators". They will be used to "reduce" (repeatedly rewrite) all newly-generated clauses. What the Knuth-Bendix algorithm does, in addition to this, is to use the existing demodulators at each stage to generate new demodulators dynamically. The method is simple: Find a subterm of one of the left-hand sides that can be rewritten in two different ways by different demodulators. Reduce the left-hand side as far as possible after starting in these two different ways. You will obtain two terms p and q. If they are different, then the set of existing demodulators is manifestly not confluent,

and the equation $p = q$ is a candidate for a new demodulator. The pair p, q is called a *critical pair*. Also $q = p$ is a candidate, so the difficulty is which way the new equation should be oriented. The solution is to put the "heaviest" term on the left. In the simplest case, "heaviest" just means "longest", but if the algorithm does not halt with that definition of "weight", other more complicated definitions might make the algorithm converge. In short, the Knuth-Bendix algorithm depends on a way of orienting new equations, and many papers have been written about the possible methods.

Because commutativity is important in many examples, but makes the Knuth-Bendix algorithm fail to converge, some effort has been expended to generalize the algorithm. If the matching for rewrites is done using "associative-commutative unification" instead of ordinary unification, then the algorithm still works, and one can simply omit the commutative and associative axioms [5]. This was the method employed in McCune's theorem-prover EQP to settle the Robbins Conjecture [71].

Returning to the late 1960s, we now describe the contribution of George Robinson and Larry Wos. They defined the inference rule they called *paramodulation*. This is essentially the rule used to generate new critical pairs at each step of the Knuth-Bendix algorithm. But in [87], it was not restricted to theories whose only relation symbol is equality. Instead, it was viewed as a general adjunct to resolution. One retains the reflexivity axiom $x = x$ and replaces transitivity, symmetry, and substitutivity with the new inference rule. They used this method to find proofs of theorems that were previously beyond the reach of computer programs. For example, with $(x \otimes y)$ defined as the commutator of x and y, they proved that in a group, if $x^3 = 1$ then $(x \otimes y) \otimes y = 1$. Although this example is purely equational, the rule of paramodulation is generally applicable, whatever relation symbols may occur in addition to equality. Robinson and Wos proved [88] the refutation-completeness of this method, i.e., any theorem has a proof by contradiction using resolution and paramodulation, with the axiom $x = x$. On the other hand, Robinson and Wos did not introduce the concept of confluence or of a complete confluent set of rules, so, for example, the deduction of the ten group-theory theorems given above escaped their notice.

For theories relying exclusively on equality, no serious distinction should made between the Knuth-Bendix method and paramodulation. They are essentially the same thing.[30] Nevertheless, as mentioned before, there is a community of researchers in "rewrite rules" and an almost disjoint community of researchers in "automated deduction", each with their own conferences and journals. The challenge for today's workers in equality reasoning is to connect the vast body of existing work with the work that has been done in

[30] The four differences listed on p. 20 of [72] are actually differences in the way the technique is used in Otter and the way it would be used in a program that implemented only Knuth-Bendix.

computer algebra, so that proofs involving computation can begin to be done by computer. This task has hardly been begun.

12 Proofs Involving Computations

There have always been two aspects of mathematics: logical reasoning and computation. These have historical roots as far back as Greece and Babylonia, respectively. Efforts to mechanize mathematics began with computation, and as discussed above, the machines of Pascal and Leibniz preceded the Logical Piano. In our time, the mechanization of computation via computer has been much more successful than the mechanization of logical reasoning. The mechanization of *symbolic* computation (as opposed to numerical computation) began in the fifties, as did the mechanization of logic. What is interesting, and surprising to people outside the field, is that the mechanization of logic and the mechanization of computation have proceeded somewhat independently. We now have computer programs that can carry out very elaborate computations, and these programs are used by mathematicians "as required". We also have "theorem-provers", but for the most part, these two capabilities do not occur in the same program, and these programs do not even communicate usefully.

Part of the problem is that popular symbolic computation software (such as Mathematica, Maple, and Macsyma) is logically incorrect. For example: Set $a = 0$. Divide both sides by a. You get $1 = 0$, because the software thinks $a/a = 1$ and $0/a = 0$. This kind of problem is pervasive and is not just an isolated "bug", because computation software applies transformations without checking the assumptions under which they are valid. Alternately, if transformations are not applied unless the assumptions are all checked, then computations grind to a halt because the necessary assumptions are not verifiable. The author's software MathXpert [11], which was written for education rather than for advanced mathematics, handles these matters correctly, as described in [10]. Later versions of Mathematica have begun attacking this problem by restricting the applicability of transformations and allowing the user to specify assumptions as extra arguments to transformations, but this is not a complete solution. Buchberger's *Theorema* project [23] is the best attempt so far to combine logic and computation, but it is not intended to be a proof-finder, but rather a proof-checker, enabling a human to interactively develop a proof. The difficulty here is that when the underlying computational ability of Mathematica is used, it is hard to be certain that all error has been excluded, because Mathematica does not have a systematic way of tracking or verifying the pre-conditions and post-conditions for its transformations.

Another program that was a pioneer in this area is *Analytica* [30]. This was a theorem-prover written in the Mathematica programming language. "Was" is the appropriate tense, since this program is no longer in use or under development. *Analytica* was primarily useful for proving identities, and made

a splash by proving some difficult identities from Ramanujan's notebooks. It could not deal with quantified formulas and did not have state-of-the-art searching abilities.

On the theorem-proving side of the endeavor, efforts to incorporate computation in theorem-provers have been restricted to two approaches: using rewrite rules (or demodulators), and calling external decision procedures for formulas in certain specialized forms. The subject known as "constraint logic programming" (CLP) can be considered in this latter category. Today there are a few experiments in linking decision-procedure modules to proof-checkers (e.g. qepcad to PVS), but there is little work in linking decision-procedure modules to proof-finding programs.[31]

The author's software MathXpert contains computational code that properly tracks the preconditions for the application of mathematical transformations. After publishing MathXpert in 1997, I then combined some of this code with a simple theorem prover I had written earlier [9], and was therefore in a unique position to experiment with the automated generation of proofs involving computation. I named the combined theorem-prover Weierstrass because the first experiments I performed involved epsilon–delta arguments. These are the first proofs, other than simple mathematical inductions, to which students of mathematics are exposed. I used Weierstrass in 1988-1990 to find epsilon-delta proofs of the continuity of specific functions such as powers of x, square root, log, sine and cosine, etc. Before this, the best that could be done was the continuity of a linear function [19]. These proofs involve simple algebraic laws (or laws involving sine, cosine, log, and the like), but, what is more, they involve combining those computations with inequality reasoning.

I then moved from analysis to number theory, and considered the proof of the irrationality of e. Weierstrass was able, after several improvements, to automatically generate a proof of this theorem [13]. The proof involves inequalities, bounds on infinite series, type distinctions (between real numbers and natural numbers), a subproof by mathematical induction, and significant

[31] Possible exceptions: if the set of demodulators is confluent and complete, then demodulation could be regarded as a decision procedure for equations in that theory. Bledsoe's rules [19], [25] (Ch. 8) for inequalities could be regarded as a decision procedure for a certain class of inequalities. Theorem provers such as EQP and RRL that have AC-unification could be regarded as having a decision procedure for linear equations. *Theorema* does contain decision procedures, but it is primarily a proof-checker, not a proof-finder. An extension of the prover RRL called Tecton [1] has a decision procedure for Presburger arithmetic.

mathematical steps, including correct simplification of expressions involving factorials and summing an infinite geometrical series.[32] [33]

Inequalities played a central role in both the epsilon-delta proofs and the proof of the irrationality of e. Inequalities certainly play a central role in classical analysis. Books and journal articles about partial differential equations, for example, are full of inequalities known as "estimates" or "bounds", that play key roles in existence proofs. Classically, mathematics has been divided into algebra and analysis. I would venture to call algebra the mathematics of equality, and analysis the mathematics of inequality.

The mechanization of equality reasoning has made more progress than the mechanization of inequality reasoning. We have discussed the "first loophole" above, which allows for the complete mechanization of certain subfields of mathematics by a "decision procedure" that algorithmically settles questions in a specific area. The mechanization of equality reasoning has benefited from the discovery of decision procedures with surprisingly wide applicability. In particular, a decision procedure has been found for a class including what are usually called *combinatorial identities*. Combinatorial identities are those involving sums and binomial coefficients, often in quite complicated algebraic forms. To illustrate with a very simple example,

$$\sum_{j=0}^{n} \binom{n}{j}^2 = \binom{2n}{n}.$$

In 1974 it was recognized by Gosper, who was at that time involved in the creation of Macsyma, that almost all such identities are special cases of a few identities involving *hypergeometric functions*, an area of mathematics initiated, like so many others, by Gauss. In 1982, Doron Zeilberger realized that recurrence relations for such identities can be generated automatically. This realization is the basis for "Zeilberger's paradigm" (see [81], p. 23). This "paradigm" is a method for proving an identity of the form $\sum_k summand(n,k) = answer(n)$. Namely: (i) find a recurrence relation satisfied by the sum; (ii) show that the proposed answer satisfies the same recurrence; (iii) check that "enough" initial values of both sides are equal. Here "enough" depends on the rational functions involved in the recurrence relation. The key to automating proofs of combinatorial identities is to automate

[32] Two things particularly amused me about this piece of work: First, one of the referees said "Of course it's a stunt." Second, audiences to whom I lectured were quite ready to accept that next I might be proving the irrationality of Euler's constant γ or solving other open problems. People today are quite jaded about the amazing latest accomplishments of computers! What the referee meant was that the "stunt" was not going to be repeated any time soon with famous open problems of number theory.

[33] It was difficult for others to build upon this work in that the code from MathXpert could not be shared, because it is part of a commercial product no longer under the author's control. In the future, similar features should be added to an existing, widely-used theorem prover, whose source code is accessible, such as Otter.

the discovery of the appropriate recurrence relation. In [81], one can learn how this is done, using methods whose roots lie in Gosper's algorithm for the summation of hypergeometric series, and in yet earlier work by Sister Mary Celine on recurrence relations. The appendix of [81] contains pointers to Mathematica and Maple implementations of the algorithms in question. In addition to verifying proposed identities, some of these algorithms can, given only the left-hand sum, determine whether there exists an "answer" in a certain form, and if so, find it. The algorithms presented in [81] are noteworthy because, unlike either proof-search or quantifier elimination for the reals, they routinely perform at human level or better in finding and proving combinatorial identities.

13 Searching for Proofs

In essence, automated deduction is a search problem. We have a list of axioms, a few "special hypotheses" of the theorem to be proved, and the negation of its conclusion, and some inference rules. These inputs determine a large space of possible conclusions that can be drawn. We must search that space to see if it contains a contradiction. In some approaches to automated deduction (that do not use proof by contradiction), we might not put the negation of the conclusion in, and then search the possible deductions for the conclusion, instead of a contradiction. Either way, a search is involved. To the extent that calculation is involved, the search can be limited–when we are calculating, we "know what we are doing". But the logical component of mathematics involves, even intuitively, a search. We "find" proofs, we do not "construct" them.

This search appears to be fundamentally infeasible. Let us see why by considering a straightforward "breadth-first search", as a computer scientist would call it. Suppose we start with just 3 axioms and one rule of inference. The three axioms we call "level 0". Level $n+1$ is the set of formulas that can be deduced in one step from formulas of level n or less, at least one of which has level exactly n. The "level saturation strategy" is to generate the levels, one by one, by applying the inference rule to all pairs of formulas of lower levels. It is difficult to count the size of the levels exactly because we cannot tell in advance how many pairs of formulas can function as premisses of the inference rule. But for a worst-case estimate, if L_n is the number of formulas in level n or less, we would have $L_{n+1} = L_n + L_n(L_n - L_{n-1})$. To make a tentative analysis, assume that L_{n-1} can be neglected compared to the much larger L_n. Then the recursion is approximately $L_{n+1} = L_n^2$, which is solved by $L_n = 2^{2^n}$. When $n = 7$ we have 2^{128}, a number that compares with the number of electrons in the universe (said to be 10^{44}). Yet proofs of level 30 are often found by Otter (according to [109], p. 225). Of course, we have given a worst-case estimate, but in practice, level saturation is not a feasible way to organize proof search.

Intuitively, the difficulty with level saturation is this: what we are doing with level saturation (whether or not the negation of the conclusion is thrown in) is developing the entire theory from the axioms. Naturally there will be many conclusions that are irrelevant to the desired one. Whole books may exist filled with interesting deductions from these axioms that are irrelevant today in spite of being interesting on another day, and there will of course be even more uninteresting conclusions. What we need, then, are techniques to

- prevent the generation of unwanted deduced clauses,
- discard unwanted clauses before they are used to generate yet more unwanted clauses,
- generate useful clauses sooner,
- use useful clauses sooner than they would otherwise be used.

Methods directed towards these objectives are called "strategies".

In 1962, when none of the strategies known today had yet been invented, the following problem was too difficult for automated theorem proving: In a group, if $x*x = e$ for every x, then the group is commutative, i.e. $z*y = y*z$ for every y and z. Today this is trivial (for both humans and computers). It was consideration of this example that led Larry Wos to invent the "set of support" strategy [107], which is today basic to the organization of a modern theorem-prover.

Here is an explanation of (one version of) this strategy. Divide the axioms into two lists, usable and set of support (sos). Normally, sos contains the negation of the desired theorem (that is, it contains the "special hypothesis" of the theorem and the negation of the conclusion of the theorem). The axioms of the theory go into usable. To generate new clauses, use resolution (or a variant of resolution) with one parent from sos and one parent from usable. Specifically, pick one "given clause" from sos. Move the given clause from sos to usable. Then make all possible inferences using the given clause as one parent, with the other parent chosen from usable. Add the new conclusions (possibly after some post-processing) to the sos list. Continue, choosing a new given clause, until the set of support becomes empty or a contradiction is derived.

The following fragment of an Otter input file illustrate the choice of sos and usable in the example mentioned above. (Here `f` means the group operation, and `g` is the inverse.)

```
list(usable).
x = x.                          \% equality
f(e,x) = x.                     \% identity
f(g(x),x) = e.                  \% inverse
f(f(x,y),z) = f(x,f(y,z)).      \%associativity
end_of_list.

list(sos).
```

```
f(x,x) = e.                 \% special hypothesis
f(a,b) != f(b,a).           \% Denial of conclusion
end_of_list.
```

Otter finds a 6-step proof, of level 4, for this problem. Wos, George Robinson, and Carson proved (acknowledging invaluable assistance from J. A. Robinson) [107] that the appropriate use of this strategy still preserves the refutation-completeness property; that is, if there exists a proof of contradiction from the formulas in usable and sos together, then in principle it can be found by this strategy, if we do not run out of space or time first. The hypothesis of this theorem is that the usable list must itself be satisfiable, i.e. not contain a contradiction. That will normally be so because we put the denial of the conclusion into sos.

Another way of trying to generate useful formulas sooner, or to avoid generating useless formulas, is to invent and use new rules of inference. Quite a number of variations of resolution have been introduced and shown to be useful, and various theorems have been proved about whether refutation completeness is preserved using various combinations of the rules. For an overview of these matters, see [109]. For additional details and many examples, see [108]. Nowadays, the user of a theorem-prover can typically specify the inference rules to be used on a particular problem, and may try various choices; while there may be a default selection (Otter has an "autonomous mode"), expertise in the selection of inference rules is often helpful.

Another common way of trying to generate useful formulas sooner is to simply throw out "useless" formulas as soon as they are generated, instead of putting them in sos for further processing. For example, if a formula is a substitution instance of a formula already proved, there is no use keeping it. If you feel (or hope) that the proof will not require formulas longer than 20 symbols, why not throw out longer formulas as soon as they are generated? More generally, we can assign "weights" to formulas. The simplest "weight" is just the length (total number of symbols), but more complex weightings are possible. Then we can specify the maximum weight of formulas to be retained. Of course, doing so destroys refutation completeness, but it may also enable us to find a proof that would otherwise never have been produced in our lifetimes. If we do not find a proof, we can always try again with a larger maximum weight.

The description of the sos strategy above leaves several things imprecise: how do we "select" a formula from sos to be the next given formula? What is the nature of the "post-processing"? These questions have interesting answers, and the answers are not unique. There are different strategies addressing these questions. Otter has many user-controllable parameters that influence these kinds of things. There are so many parameters that running Otter is more of an art than a science. For a more detailed description of the basic algorithm of Otter, see [109], p. 94, where the program's main loop is summarized on a single page.

It has now been nearly forty years since the invention of the set of support strategy, and the general approach to theorem proving described above has not changed, nor has any competing approach met with as much success. Over that forty years, the approach has been refined by the development of many interesting strategies. The skillful application of these strategies has led to the solution of more and more difficult problems, some of which were previously unsolved. An impressive list of such problems solved just in the last couple of years is given in [41].[34] These problems are in highly technical areas, so it is difficult to list and explain them in a survey article. To give a taste of this kind of research, we shall explain just one of the areas involved: propositional logic. You may think that propositional logic is trivial. After all, you know how to decide the validity of any proposition by the method of truth tables. Therefore it is first necessary to convince you that this is an area with interesting questions. We write $i(x, y)$ for "x implies y", and $n(x)$ for "not x". Since "and" and "or" can be defined in terms of implication and negation, we will restrict ourselves to the connectives i and n. The Polish logician Jan Łukasiewicz (1878-1956) introduced the following axioms for propositional logic:

$$i(i(x, y), i(i(y, z), i(x, z))) \tag{1}$$

$$i(i(n(x), x), x) \tag{2}$$

$$i(x, i(n(x), y)) \tag{3}$$

To work with these axioms in Otter, we use the predicate $P(x)$ to mean "x is provable". We then put into the usable list,

```
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(n(x),x),x)).
P(i(x,i(n(x),y))).
-P(x) | -P(i(x,y)) | P(y).
```

Now to ask, for example, whether $i(x, x)$ is a theorem, we put `-P(i(c,c))` into list(sos). That is, we put in the negation of the assertion that $i(c, c)$ is provable. The steps taken by resolution correspond to the rule of "detachment" used by logicians: To deduce a new formula from A and $i(B, C)$, make a substitution $*$ so that $A^* = B^*$. Then you can deduce C^*.[35] Why do we

[34] If the non-expert user looks at the list given in [41] of difficult problems solved using Otter, he or she will very likely not be able in a straightforward manner to get Otter to prove these theorems. He or she will have to go to the appropriate web site and get the input files prepared by the experts, specifying the inference rules and parameters controlling the search strategies. As the authors state, they do not have a single uniform strategy that will enable Otter to solve all these difficult problems, and a lot of human trial and error has gone into the construction of those input files.

[35] Technically, since a theorem prover always uses the *most general* unifier, it corresponds to the rule known as "condensed detachment", in which only most general substitutions are allowed.

need the predicate P? Because we are interested in proofs from L1–L3 using condensed detachment; P is used to force the theorem prover to imitate that rule. We are not just interested in verifying tautologies, but in finding proofs from the specific axioms L1–L3. Now, the reader is invited to try to prove $i(x, x)$ from the axioms L1–L3. This should be enough to convince you that the field is not trivial. Other axiom systems for propositional logic were given by Frege, by Hilbert, and by Łukasievich. (See the wonderful appendix in [83], where these and many other axiom systems are listed. The questions then arise about the equivalence of these axiom systems. We want proofs of each of these axiom systems from each of the others. The appendix of [109] (pp. 554-55) lists Otter proofs of the some of these systems from L1-L3. For example, the first axiom in Frege's system is $i(x, n(n(x)))$. Go ahead, John Henry: try to prove it from L1-L3 using pencil and paper.

One type of interesting question studied by logicians in the 1930s through 1950s–and resumed again today with the aid of automated reasoning–was this: given a theory T defined by several axioms, can we find a "single axiom" for T? That is, a single formula from which all the axioms of T can be derived. If so, what is the shortest possible such axiom? This type of question has been attacked using Otter for a large number of different systems, including various logics, group theories, and recently, lattice theory. For example, "equivalential calculus" is the logical theory of bi-implication (if and only if). It can be represented using $e(x, y)$ instead of $i(x, y)$, and treated using a "provability predicate" P as above. See [113] for an example of an Otter proof that settled a long-open question in equivalential calculus, namely, whether a certain formula XCB is a single axiom for this theory. This is perhaps the most recent example of a theorem that has been proved for the first time by a computer. Before it was proved (in April, 2002), people were not willing to give odds either way on the question.

14 Proofs Involving Sets, Functions, and Numbers

If we examine a textbook for an introductory course in abstract algebra, such as [56], we find that only about ten percent of the problems can be formulated in the first-order languages of groups, rings, etc. The rest involve subgroups, subrings, homomorphisms, and/or natural numbers. For example, one of the first theorems in group theory is Lagrange's theorem: if H is a subgroup of a finite group G, then the order of H (the number of its elements) divides the order of G. Here we need natural numbers to define the order, and a bit of number theory to define "divides"; we need the concept of subgroup, and the proof involves constructing a function to put H in one-one correspondence with the coset Ha, namely, $x \mapsto xa$. At present, no theorem-proving program has ever generated a proof of Lagrange's theorem, even though the proof is very short and simple. The obstacle is the mingling of

elements, subgroups, mappings, and natural numbers.[36] The present power of automated theorem provers has yielded results only in theories based on equality and a few operations or in other very simple theories. At least half of undergraduate mathematics should come within the scope of automated proof generation, if we are able to add in a relatively small ability to deal with sets, numbers, and functions. We do not (usually) need sets of sets, or sets of sets of sets, and the like. Nor do we usually need functions of functions, except special functions of functions like the derivative operator. If we add to a first-order theory some variables for sets (of the objects of the theory) and functions (from objects to objects), we have what is known as a second-order theory. The lambda-calculus can be used to define functions, and sets can be regarded as Boolean-valued functions. The author's current research involves adding capabilities to the existing, widely-used theorem prover Otter to assist it in handling second-order theories, without interfering with its first-order capabilities.[37] Specifically, a new second-order unification algorithm [14,15], has been added to Otter, and will be improved and applied. Preliminary results, and the direction of the research program, are described in [16].

One may object to the use of second-order logic, and indeed to the whole idea of a "taxonomy" of mathematics, on the grounds of the universality of set theory. Let us begin by stating the objection clearly. Set theory is a "simple theory", with one relation symbol for membership and a small number of axioms. True, one of the "axioms" of ZF set theory is an infinite schema, with one instance for each formula of the language; but there is another formulation of set theory, Gödel–Bernays set theory (GB), which has a small finite number of axioms. In GB, variables range over classes, and sets are defined as classes which belong to some other class. (The idea is that properties define classes, but not every class is a set—we escape the Russell paradox in GB because the Russell class is a class, but not a set.) Because of the possibility of formulating set theory in this way as a simple theory, the taxonomy given above collapses–all of mathematics is contained in a single simple first-order theory. Now for some relevant history: a century ago, this program for the foundations of mathematics was laid out, but in the middle twentieth century, the Bourbaki school prevailed, at least in practice, organizing mathematics according to the "many small theories" program. At present, most work in

[36] A proof of Lagrange's theorem developed by a human has been checked by the computer program ACL2, see [116]. That proof is not the ordinary proof, but instead proceeds by mathematical induction on the order of the group. The ordinary proof has been proof-checked using HOL [58]. That paper also presents an interesting "theory hierarchy" showing exactly what is needed. It has also been checked in Mizar [99].

[37] Of course, second-order and (why not?) higher-order theorem proving has been in existence for a long time, and there are even whole conferences devoted to the subject, e.g. TPHOL (Theorem Proving in Higher-Order Logic). It seems that most of this research is not directed towards proving new theorems, so it has not been discussed in this paper.

automated deduction is based on the "small theories" approach, although one brave man, Belinfante, has been proceeding for many years to develop computerized proofs based on GB set theory [17,18]. Following this approach, he has enabled Otter to prove more than 1000 theorems in set theory–but he still is not up to Lagrange's theorem, or even close. Belinfante built on the pioneering work of Quaife [94]. Finally: the answer to the objection is simply that it is too complex to regard numbers and functions as built out of sets. No mathematician does so in everyday practice, and neither should automated deduction when the aim is to someday prove new theorems.[38]

On the other hand, one may take the opposite view and say that, because of the difficulties of developing mathematics within set theory, one should use "higher-order logic". This has been the view of many of the "proof-checking" projects, and they have been successful in checking proofs of many fairly complicated theorems. A proper review of this work would double the length of this article, so we must forego it. The interested reader can consult [105] for a list of fifteen proof checkers and proof finders, as well as references to further information.

15 Conclusion

Alan Turing wrote a seminal paper [101] in which he raised the question "Can machines think?".[39] After discussing various examples, such as chess, musical composition, and theorem proving, he then formulated the "Turing test" as a replacement for that imprecise question. In the Turing test, a computer tries to deceive a human into thinking that the computer is human.[40] Of course in the foreseeable future it will be too difficult for a single computer to be able to reach human level in many areas simultaneously; but we might consider restricted versions of the Turing test for specific areas of endeavor. As mentioned in the introduction, the Turing test has already been passed for musical composition: David Cope has written a program EMI (pronounced "Emmy", for Experiments in Musical Intelligence) which produces music that regularly fools sophisticated audiences–at least, it did until Cope stopped conducting "The Test" at his concerts—stopped because the experts were too embarrassed. Since Cope is a composer rather than a computer scientist, he presents his results primarily at concerts rather than conferences.

[38] For the record, Belinfante agrees with this statement. His aim, however, is foundational. As a boy, he took *Principia Mathematica* from his physicist father's bookshelf and said to himself, "Someday I'm going to check if all these proofs are really right!". That spirit still animates his work.

[39] Like Stanley Jevon's paper, an original copy of this journal article now is priced at $2000.

[40] A more detailed discussion of the Turing test can be found in Turing's paper *op. cit.* or in any modern textbook on artificial intelligence; the idea of a computer trying to appear human is enough for our purposes.

I heard a seven-piece chamber orchestra perform the Eighth Brandenburg Concerto (composed by EMI). (Bach composed the first seven Brandenburg Concertos.)

In theorem-proving, as in artificial intelligence, there was initially a division between those who thought computers should be programmed to "think" like humans and those who favored a more computational approach. Should we try to find "heuristics" (rules of thumb) to guide a computer's efforts to find a proof, or play a game of chess, or compose a piece of music? Or should we just give the computer the rules and a simple algorithm and rely on the power of silicon chips? It is interesting to compare computerized chess and computerized theorem-proving in this respect. Both can be viewed as search problems: chess is a search organized by "if I make move x_1 and he makes move y_1 and then I make move x_2 and he makes move y_2 and then ..."; we search the various possibilities, up to a certain "level" or "depth", and then, for each sequence of possible moves, we score the situation. Then we pick our move, using a "max-min" algorithm. As in theorem proving, the difficulty is to "prune the search tree" to avoid getting swamped by the consideration of useless moves. In both endeavors, theorem proving and chess, one feels that expert human beings have subtle and powerful methods. Chess players analyze *far* fewer possibilities than chess programs do, and those good possibilities are analyzed deeper. One feels that the same may be true of mathematicians. In chess programs, "knowledge" about openings and end games is stored in a database and consulted when appropriate. But in the mid-game, every effort in chess programming to use more specialized chess knowledge and less search has failed. The computer time is better spent searching one move deeper. On the other hand, the game of go is played at only slightly above the beginner level by computers. The board is 19 by 19 instead of 8 by 8, and there are more pieces; the combinatorial explosion is too deadly for computers to advance much beyond this level at present.[41]

Similarly, in mathematics, so far at least, search has proved the most fruitful general technique. One can view computer algebra and computerized decision procedures, such as quantifier elimination or Wilf and Zeilberger's decision procedure for combinatorial sums, as ways of embedding mathematical knowledge in computer programs. Where they are applicable, they play an indispensable role, analogous to the role of opening books and end game databases in chess. In areas of mathematics in which it is difficult to bring knowledge to bear (such as elementary group theory or propositional logic)

[41] The game tree, searching n moves in the future, has about b^n nodes, where at a crude approximation $b = 8^2$ for chess and 19^2 for go. So the ratio is $(19/8)^{2n}$. Taking $n = 10$ we get more than 2^{20}, which is about a million: go is a million times harder than chess. On the other hand, computer speeds and memory sizes have historically increased exponentially, doubling every 18 months; so if Moore's law continues to hold, we might hope that go programs would perform well enough in 30 years, even without improvements in the programs.

because the axioms are very simple and tools from outside the subject area are not applicable, theorem-proving programs can outperform human beings, at least sometimes, just as in chess.

How did the trade-off between high-speed but simple computation and heuristics play out in the area of musical composition? The answer to this question is quite interesting, and may have implications for the future of research in theorem proving. EMI does not compose from a blank slate. To use EMI, you first decide on a composer to be imitated; let's say Bach. Then, you feed EMI several compositions by Bach. EMI extracts from these data a "grammar" of musical structures. EMI then uses this grammar to generate a new composition, which will be perceived as "in the style of Bach". The selection of the initial database calls for musical expertise and for expertise with EMI. For example, to compose the Eighth Brandenburg Concerto, Cope chose some of the original seven Brandenburg Concertos and a couple of violin concertos. When the database contained only the seven Brandenburg Concertos, the resulting composition seemed too "derivative", and even contained recognizable phrases. Yet, once the data has been digested, the program works according to specific, precise rules. There is nothing "heuristic" about the process. A result that "looks human" has been achieved by computational means.

This is at present not true of most proofs produced by most theorem-provers. To exhibit a simple example, consider the theorem that a group in which $x^2 = 1$ for all x is commutative. A human proof might start by substituting uv for x, to get $(uv)^2 = 1$. Multiplying on the left by u and on the right by v the desired result is immediate. The proof that a theorem-prover finds is much less clever. In longer proofs, the computer's inhuman style stands out even more. The most notable feature of such proofs is that theorem provers never invent concepts or formulate lemmas. A paper written by a mathematician may have a "main theorem" and twenty supporting lemmas. The proof of the main theorem may be quite short, but it relies on the preceding lemmas. Not only does this help understanding, but the lemmas may be useful in other contexts. The most powerful present-day theorem provers never find, organize, or present their proofs in this way (unless led to do so by a human after a failure to find the proof in one go).

Theorem-provers of the future should be able to invent terminology and form definitions. The basis of their ability to do this should be an underlying ability to monitor and reason about their own deduction process. As it is now, humans using a theorem prover monitor the output, and then change parameters and restart the job. In the future, this kind of feedback should be automated and dynamic, so that the parameters of a run can be altered (by the program itself) while the run is in progress. With this capability in hand, one should then be able to detect candidates for "lemma" status: short formulas that are used several times. It is then a good idea to keep an eye out for further deductions similar in form to the formulas involved in the proof

of the lemmas.[42] Giving a program the ability to formulate its own lemmas dynamically might, in conjunction with the ability to modify the criteria for keeping or using deduced formulas, enable the program to find proofs that might otherwise be beyond reach.

Such a prover might produce proofs that look more human. The investigation, the style of thought, and the development of the theory should each look more like proofs people produce. Searching would still be the basis (not heuristics), but the result would look less like it had been produced by Poincaré's famous machine that takes in pigs and produces sausages. This type of prover would be a little more like EMI than today's theorem provers. One might even be able to prime it with several proofs by the famous logician Meredith in order to have it produce proofs in propositional logic in Meredith's style, much as EMI can produce music in the style of Bach or the style of Scott Joplin. At present this is rather farfetched, as there is nothing like the "grammar" of Meredith's proofs. The closest approximation at present would be to tell the program to retain deduced clauses similar in form to the lines of the proofs used to prime the program.

We do not expect, however, that all machine-generated proofs will "look human". For example, there exists a machine-generated proof that a certain formula is a single axiom for groups satisfying $x^{19} = 1$ for all x. This proof contains a formula 715 symbols long. No human will find that proof.

Remember: the question whether machines can think is like the question whether submarines can swim. We expect machine mathematics to be different from human mathematics–but it seems a safe prediction that the twenty-first century will see some amazing achievements in machine mathematics.

References

1. Agarwal, R., Kapur, D., Musser, D. R., and Nie, X., Tecton proof system. In: Book, R. (ed.) *Proc. Fourth International Conference on Rewriting Techniques and Applications, Milan, Italy, 1991.* LNCS **488**, Springer-Verlag (1991).
2. Appel, K., and Haken, W., Every planar map is four colorable. Part I. Discharging, *Illinois J. Math.* **21** 429–490, 1977.
3. Appel, K., Haken, W. Haken, and Koch, J., Every planar map is four colorable. Part II. Reducibility, *Illinois J. Math.* **21**491–567, 1977.
4. Arnon, D., and Buchberger, B., *Algorithms in Real Algebraic Geometry*, Academic Press, London (1988). Reprinted from *J. Symbolic Computation* **5** Numbers 1 and 2, 1988.
5. Baader, F., and Snyder, W., Unification theory, in [90], pp. 435-534.

[42] There are various possible notions of "similar in form" that might be used. For example, one idea is to call formulas similar if they become the same when all variables are replaced with the same letter. This notion is behind a successful strategy called "resonance". [109], p. 457.

6. Barendregt, H., *The Lambda Calculus: Its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics **103**, Elsevier Science Ltd. Revised edition (October 1984).
7. Barendregt, H., and Geuvers, H., Proof-Assistants Using Dependent Type Systems, in: Robinson, A., and Voronkov, A. (eds.), *Handbook of Automated Reasoning, vol. II*, pp. 1151-1238. Elsevier Science (2001).
8. Basu, S., Pollack, R., and Roy, M. F., On the Combinatorial and Algebraic Complexity of Quantifier Elimination, *Journal of the ACM* **43** (6) 1002–1046, 1996.
9. Beeson, M., Some applications of Gentzen's proof theory to automated deduction, in P. Schroeder-Heister (ed.), *Extensions of Logic Programming*, Lecture Notes in Computer Science **475** 101-156, Springer-Verlag (1991).
10. Beeson, M., *Mathpert*: Computer support for learning algebra, trigonometry, and calculus, in: A. Voronkov (ed.), Logic Programming and Automated Reasoning, Lecture Notes in Artificial Intelligence 624, Springer-Verlag (1992).
11. Beeson, M., *Mathpert Calculus Assistant*. This software product was published in July, 1997 by Mathpert Systems, Santa Clara, CA. See www.mathxpert.com to download a trial copy.
12. Beeson, M., Automatic generation of epsilon-delta proofs of continuity, in: Calmet, Jacques, and Plaza, Jan (eds.) *Artificial Intelligence and Symbolic Computation: International Conference AISC-98, Plattsburgh, New York, USA, September 1998 Proceedings*, pp. 67-83. Springer-Verlag (1998).
13. Beeson, M., Automatic generation of a proof of the irrationality of e, *Journal of Symbolic Computation* **32**, No. 4 (2001), pp. 333-349.
14. Beeson, M., Unification in Lambda Calculus with if-then-else, in: Kirchner, C., and Kirchner, H. (eds.), *Automated Deduction-CADE-15. 15th International Conference on Automated Deduction, Lindau, Germany, July 1998 Proceedings*, pp. 96-111, Lecture Notes in Artificial Intelligence **1421**, Springer-Verlag (1998).
15. Beeson, M., A second-order theorem prover applied to circumscription, in: Gor, R., Leitsch, A., and Nipkow, T. (eds.), *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 2001, Proceedings*, Lecture Notes in Artificial Intelligence **2083**, Springer-Verlag (2001).
16. Beeson, M., Solving for functions, to appear in *Journal of Symbolic Computation*. A preliminary version appeared in: *LMCS 2002, Logic, Mathematics, and Computer Science: Interactions, Symposium in Honor of Bruno Buchberger's 60th Birthday*, pp. 24–38, RISC-Linz Report Series No. 02-60, Research Institute for Symbolic Computation, Linz (2002).
17. Belinfante, J., Computer proofs in Gödel's class theory with equational definitions for composite and cross, *J. Automated Reasoning* **22**, No. 3, pp. 311-339, 1988.
18. Belinfante, J., On computer-assisted proofs in ordinal number theory, *J. Automated Reasoning* **22**, No. 3, pp. 341-378, 1988.
19. Bledsoe, W. W., and Hines, L. M., Variable elimination and chaining in a resolution-based prover for inequalities
20. Boyer, R. S., and Moore, J. S., *A Computational Logic Handbook*, Academic Press, Boston (1988).
21. Bronstein, M., *Symbolic Integration I: Transcendental Functions*, Springer-Verlag, Berlin/ Heidelberg/ New York (1997).

22. Brown, C., QEPCAD-B, a program for computing with semi-algebraic sets using CADs, to appear. Preprint available at http://www.cs.usna.edu/ wcbrown/research/MOTS2002.2.pdf. The program itself is available from
http://www.cs.usna.edu/ qepcad/B/QEPCAD.html, and five example problems can be viewed at
http://www.cs.usna.edu/ qepcad/B/examples/Examples.html.
23. Buchberger, B., *et. al.* Theorema: An Integrated System for Computation and Deduction in Natural Style, in: Kirchner, C., and Kirchner, H. (eds.), *Proceedings of the Workshop on Integration of Deductive Systems at CADE-15, Lindau, Germany, July 1998*, LNAI **1421**, Springer, Berlin (1998).
24. Buchberger, B., Collins, G., and Loos, R., *Computer Algebra: Symbolic and Algebraic Manipulation*, second edition, Springer-Verlag Wien/ New York (1983).
25. Bundy, A., *The Computer Modelling of Mathematical Reasoning*, Academic Press, London (1983).
26. Boole, G., *The Laws of Thought*, Dover, New York (1958). Original edition, MacMillan (1854).
27. Borsuk, K., and Szmielew, W. *Foundations of Geometry*, North-Holland, Amsterdam, (1960).
28. Caviness, B.F., and Johnson, J.R. (eds.) *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Springer, Wien/New York (1998).
29. Church, A., An unsolvable problem of elementary number theory, *American Journal of Mathematics* **58** 345–363, 1936.
30. Clarke, E., and Zhao, X.: Analytica: A Theorem Prover in Mathematica, in: Kapur, D. (ed.), *Automated Deduction: CADE-11: Proc. of the 11th International Conference on Automated Deduction*, pp. 761–765, Springer-Verlag, Berlin/Heidelberg (1992).
31. Collins, G.E., Quantifier elimination for real closed fields by cylindrical algebraic decomposition, in: *Proc. 2nd Conf. on Automata Theory and Formal Languages*, Springer LNCS **33** 134–183. Reprinted in [28], pp. 85-121.
32. Conway, J., and Sloane, N., *Sphere Packings, Lattices and Groups*, Grundlehren Der Mathematischen Wissenschaften **290**, Springer-Verlag, Berlin/ Heidelberg/ New York (1998).
33. Cope, D., *Computers and Musical Style*, A-R Editions, Madison (1991).
34. Cope, D., *Experiments in Musical Intelligence*, A-R Editions, Madison (1996).
35. Cope, D., *The Algorithmic Composer*, A-R Editions, Madison (2000).
36. Davenport, J., and Heintz, J., Real quantifier elimination is doubly exponential, in [4], pp. 29–35.
37. Davis, M., A computer program for Presburger's algorithm, in *Summaries of Talks Presented at the Summer Institute for Symbolic Logic, 1957* Second edition, published by Institute for Defense Analysis, 1960. Reprinted in [92], pp. 41-48.
38. Davis, M. The prehistory and early history of automated deduction, in [92], pp. 1–28.
39. Davis, M., *The Universal Computer: The Road from Leibniz to Turing*, Norton (2000). Reprinted in 2001 under the title *Engines of Logic*.
40. Davis, M., and Putnam, H., A computing procedure for quantification theory, *JACM* **7** 201-215, 1960.
41. Ernst, Z., Fitelson, B., Harris, K., McCune, W., Veroff, R., Wos, L., More First-order Test Problems in Math and Logic, in: Sutcliffe, G., Pelletier, J. and Suttner,

C. (eds.) Proceedings of the CADE-18 Workshop–Problems and Problem Sets for ATP, Technical Report 02/10, Department of Computer Science, University of Copenhagen, Copenhagen (2002). The paper and associated Otter files can be accessed from http://www.mcs.anl.gov/ mccune/papers/paps-2002.
42. Fischer, M. J., and Rabin, M. O., Super-exponential complexity of Presburger arithmetic, *SIAM-AMS Proceedings*, Volume VII, pp. 27–41; reprinted in [28], pp. 122–135.
43. Frege, G., *Begriffshrift, a formula language, modeled upon that of arithmetic, for pure thought.*, English translation in [102], pp. 1–82. Original date 1879.
44. Gelernter, H., Realization of a geometry-theorem proving machine, *Proc. International Conference on Information Processing, UNESCO House* 273–282, (1959). Reprinted in Feigenbaum and Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York (1963). Reprinted again in [92], pp. 99-124.
45. Über formal unentscheidbare Sätze der *Principia Mathematica* und verwandter Systems I, in: Feferman, S., *et. al.* (eds.), *Kurt Gödel: Collected Works, Volume I, Publications 1929-1936*, pp. 144-195. (The translation, On formally undecidable propositions of *Principia Mathematica* and related systems I, appears on facing pages.) Oxford University Press, New York, and Clarendon Press, Oxford (1986).
46. Grigor'ev, D., and Vorobjov, N., Solving systems of polynomial inequalities in subexponential time. J. Symb. Comput. **5**, 37-64, 1988.
47. Guard, J., Oglesby, F., Bennett, J., and Settle, L., Semi-automated mathematics, *JACM* **16**(1) 49–62, 1969.
48. Harrison, J., and Théry, L.: Extending the HOL theorem prover with a computer algebra system to reason about the reals, in *Higher Order Logic Theorem Proving and its Applications: 6th International Workshop, HUG '93*, pp. 174–184, Lecture Notes in Computer Science **780**, Springer-Verlag (1993).
49. Harrison, J., *Theorem Proving with the Real Numbers*, Springer-Verlag, Berlin/Heidelberg/New York (1998).
50. Hilbert, D., *Grundlagen der Geometrie*, Teubner (1899). English translation (of the 10th edition, which appeared in 1962): *Foundations of Geometry*, Open Court, La Salle, Illinois (1987).
51. Hilbert, D., and Ackermann, W., 'Grundzge der theoretischen Logik', 2nd edition (1938; first edition 1928). English translation: *Principles of Mathematical Logic*, translated by Lewis M. Hammond, George G. Leckie and F. Steinhardt; edited with notes by Robert E. Luce, Chelsea (1950). The translation is to be reprinted by the A. M. S. in 2003.
52. Hong, H., Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination, in: *Proc. International Symposium on Symbolic and Algebraic Computation*, pp. 177-188, ACM, 1992. Reprinted in [28], pp. 210–210.
53. Horgan, J., The death of proof, *Scientific American*, October 1993, 74–82.
54. Huang, T., *Automated Deduction in Ring Theory*, Master's Writing Project, Department of Computer Science, San Jose State University (2002).
55. G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science* **1** (1975) 27–52.
56. Jacobsen, N., *Basic Algebra I*, (1985).
57. Kaltofen, E., Factorization of polynomials, in [24], pp. 95–114.
58. Kammüller, F., and Paulson, L., A formal proof of Sylow's theorem: an experiment in abstract algebra with Isabelle HOL, *J. Automated Reasoning* **23**, pp. 235–264, 1999.

59. Kapur, D., and Zhang, H., An overview of Rewrite Rule Laboratory (RRL), *J. of Computer and Mathematics with Applications*, **29** 2, 91–114,1995.
60. Kleene, S. C., λ-definability and recursiveness, Duke Mathematical Journal **2**, pp. 340–353, 1936.
61. Kleene, S. C., *Introduction to Metamathematics*, van Nostrand, Princeton (1952).
62. Knuth, D. E., and Bendix, P. B., Simple word problems in universal algebras, in: Leech, J. (ed)., *Computational Problems in Abstract Algebras* pp. 263–297, Pergamon Press (1970). Reprinted in [93], pp. 342–376.
63. Knuth, D. E., *Seminumerical Algorithms: The Art of Computer Programming, Volume 2*, second edition, Addison-Wesley, Reading, MA (1981).
64. Kurosh, A. G., *The Theory of Groups, volume 1*, Chelsea, New York (1955).
65. Lang, S. *Algebra*, Addison-Wesley, Reading, MA. (1965).
66. Lewis, Paul, obituary of Herb Simon, http://www.cs.oswego.edu/ blue/hx/ courses/cogsci1/s2001/section04/subsection01/main.html.
67. Mancuso, P., *From Brouwer to Hilbert*, Oxford University Press, Oxford (1998).
68. Maslov, S., Mints, G., and Orevkov, V., Mechanical proof-search and the theory of logical deduction in the USSR, in: [92], pp. 29–37.
69. Moore, G. E., Cramming more components onto integrated circuits, *Electronics* **38**, Number 8, pp. April 18, 1965. Available at http://www.intel.com/research/silicon/moorespaper.pdf.
70. McCune, W., Otter 2.0, in: Stickel, M. E. (ed.), *10th International Conference on Automated Deduction* pp. 663–664, Springer-Verlag, Berlin/Heidelberg (1990).
71. McCune, W.,Solution of the Robbins problem, *J. Automated Reasoning* **19**(3) 263–276, 1997.
72. McCune, W., and Padmanabhan, R., *Automated Deduction in Equational Logic and Cubic Curves*, LNAI **1095**, Springer, Berlin/ Heidelberg (1996).
73. Monk, J. D., The mathematics of Boolean algebra, in the *Stanford Dictionary of Philosophy*, http://plato.stanford.edu/entries/boolalg-math.
74. Newell, A., Shaw, J. C., Simon, H., Empirical explorations with the Logic Theory Machine: a case study in heuristics, *Proceedings of the Western Joint Computer Conference, Institute of Radio Engineers* 218-230, 1957. Reprinted in [92], pp. 49–74.
75. Peano, G., *Arithmetices principia, nova methodo exposita*, Bocca, Turin, (1889).
76. Pietrzykowski, T., and Jensen, D., A complete mechanization of second order logic, *J. Assoc. Comp. Mach.* **20** (2), 1971, pp. 333-364.
77. Pietrzykowski, T., and Jensen, D., A complete mechanization of ω-order type theory, *Assoc. Comp. Math. Nat. Conf.* 1972, Vol. 1, 82-92.
78. Poincaré, H., *Science and Method*, translated by Maitland from the original French edition of 1908, Dover (1952).
79. Penrose, R., *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics* , American Philological Association (1989). See also the review by McCarthy, J., in *Bulletin of the American Mathematical Society*, October, 1990.
80. Penrose, R., *Shadows of the Mind: A Search for the Missing Science of Consciousness*, Oxford University Press, Oxford (1996). See also the critical reviews by McCarthy, J., in the electronic journal *Psyche* **2** (11) (1995), http://psyche.cs.monash.edu.au/v2/psyche-2-11-mccarthy.html, and by

Feferman, S., *Psyche* **2**(7) (1995), http://psyche.cs.monash.edu.au/v2/psyche-2-07-feferman.html, and Penrose's replies to these and other critics, at http://psyche.cs.monash.edu.au/v2/psyche-2-23-penrose.html.
81. Petkovšek, M., Wilf, H., and Zeilberger, D., *A=B*, A. K. Peters, Wellesley, MA (1996).
82. Presburger, M., Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt, Sparwozdanie z I Kongresu Matematyków Krajów Slowiańskich Warszawa, pp. 92-101, 1929.
83. Prior, A. N., *Formal Logic*, second edition, Clarendon Press, Oxford (1962).
84. Renegar, J., Recent progress on the complexity of the decision problem for the reals, in: [28], pp. 220–241.
85. Richardson, D., Some unsolvable problems involving elementary functions of a real variable, J. Symbolic Logic **33** 511–520 (1968).
86. Robinson, A., Proving theorems, as done by man, machine, and logician, in *Summaries of Talks Presented at the Summer Institute for Symbolic Logic, 1957* Second edition, published by Institute for Defense Analysis, 1960. Reprinted in [92], pp. 74-76.
87. Robinson, G., and Wos, L., Paramodulation and theorem-proving in first-order theories with equality, in Meltzer and Michie (eds.), *Machine Intelligence* **4**, pp. 135-150, American Elsevier, New York (1969). Reprinted in [110], pp. 83–99.
88. Robinson, G., and Wos, L., Completeness of paramodulation, *Journal of Symbolic Logic* **34**, p. 160 (1969). Reprinted in [110], pp. 102–103.
89. Robinson, J. A., A machine oriented logic based on the resolution principle, *JACM* **12**, 23-41, 1965. Reprinted in [92], pp. 397-415.
90. Robinson, Alan, and Voronkov, A. (eds.),*Handbook of Automated Reasoning, Volume I*, MIT Press, Cambridge, and North-Holland, Ammsterdam (2001).
91. Russell, B., and Whitehead, A. N., *Principia Mathematica*, Cambridge University Press, Cambridge, England. First edition (1910), second edition (1927), reprinted 1963.
92. Siekmann, J., and Wrightson, G. (eds), *Automation of Reasoning 1: Classical Papers on Computational Logic 1957-1966*, Springer-Verlag, Berlin/Heidelberg/New York (1983).
93. Siekmann, J., and Wrightson, G. (eds), *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, Springer-Verlag, Berlin/Heidelberg/New York (1983).
94. Quaife, A., *Automated Development of Fundamental Mathematical Theories, Automated Reasoning, Vol.2*, Kluwer Academic Publishers, Dordrecht (1992).
95. Risch, R., The problem of integration in finite terms, *Transactions of the AMS* **139** 167–189, 1969.
96. Risch, R., The solution of the problem of integration in finite terms, *Bulletin of the AMS* **76** 605–608, 1970.
97. Tarski,A., A decision method for elementary algebra and geometry. Report R-109, second revised edition, Rand Corporation, Santa Monica, CA, 1951. Reprinted in [28], pp. 24–84.
98. Tarski, A., What is elementary geometry? in: Henkin, L., Suppes, P, and Tarski, A. (eds.), *Proceedings of an International Symposium on the Axiomatic Method, with Special Reference to Geometry and Physics* 16-29, North-Holland, Amsterdam (1959).

99. Trybulec, W. A., Subgroup and cosets of subgroup, *Journal of Formalized Mathematics* **2**, 1990. This is an electronic journal, published at http://mizar.uwb.edu.pl/JFM/index.html.
100. Turing, A., On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, ser. 2 **42** 230–265, 1936-7; corrections, *ibid.* **43** (1937) pp. 544-546.
101. Turing, A., Computing Machines and Intelligence, in *MIND, A Quarterly Review of Psychology and Philosophy***59**, No. 236, October, 1950, pp. 433-460.
102. van Heijenoort, J. (ed.) *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, Harvard University Press, Cambridge, MA (1967).
103. Wang, H., Toward mechanical mathematics, 1960. Reprinted in [92], pp. 244–267.
104. Weispfenning, V., The complexity of linear problems in fields, *J. Symbolic Computation* **5** 3–27 (1988).
105. Wiedijk,F., The fifteen provers of the world, to appear.
106. Winkler, F., *Polynomial Algorithms in Computer Algebra*, Springer-Verlag, Wien/ New York (1996).
107. Wos., L., Robinson, G., and Carson, D., Efficiency and completeness of the set of support strategy in theorem proving, *JACM* **12**(4) 536–541, 1965. Reprinted in [110], pp. 29–36.
108. Wos, L., *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial*, Academic Press, San Diego (1996).
109. Wos, L., and Pieper, G., *A Fascinating Country in the World of Computing*, World Scientific, Singapore (1999).
110. Wos, L., and Pieper, W., *The Collected Works of Larry Wos, Volume I: Exploring the Power of Automated Reasoning*, World Scientific, Singapore (2000).
111. Wos, L., and Pieper, W., *The Collected Works of Larry Wos, Volume II: Applying Automated Reasoning to Puzzles, Problems, and Open Questions*, World Scientific, Singapore (2000).
112. Wos, L., Reflections on Automated Reasoning at Argonne: A Personalized Fragment of History, on the CD-ROM included with [109].
113. Wos, L., Ulrich, D, and Fitelson, B., XCB, the last of the shortest single axioms for the classical equivalential calculus, to appear in *J. Automated Reasoning*
114. Wu, Wen-Tsun, On the decision problem and the mechanization of theorem-proving in elementary geometry, *Scientia Sinica* **21** (2), 1978. Reprinted in: Bledsoe, W. W., and Loveland, D. W. (eds.), *Automated Theorem Proving: After 25 Years*, AMS, Providence, RI (1983).
115. Wu, Wen-Tsun, *Mechanical Theorem Proving in Geometries: Basic Principles*, Springer-Verlag, Wien/ New York (1994).
116. Yu, Y. Computer Proofs in Group Theory, *Journal of Automated Reasoning* **6**(3) 251-286, 1990.
117. Zhang, H., Automated proof of ring commutativity problems by algebraic methods, *J. Symbolic Computation* **9** 423–427, 1990.