Formal Proof

Thomas C. Hales

There remains but one course for the recovery of a sound and healthy condition—namely, that the entire work of the understanding be commenced afresh, and the mind itself be from the very outset not left to take its own course, but guided at every step; and the business be done as if by machinery.

—F. Bacon, 1620
Novum Organum

Daily, we confront the errors of computers. They crash, hang, succumb to viruses, run buggy software, and harbor spyware. Our tabloids report bizarre computer glitches: the library patron who is fined US$40 trillion for an overdue book, because a barcode is scanned as the size of the fine; or the dentist in San Diego who was delivered over 16,000 tax forms to his doorstep when he abbreviated "suite" in his address as "su".

On average, a programmer introduces 1.5 bugs per line while typing. Most are typing errors that are spotted at once. About one bug per hundred lines of computer code ships to market without detection. Bugs are an accepted part of programming culture. The book that describes itself as the "bestselling software testing book of all time" states that "testers shouldn't want to verify that a program runs correctly" [17]. Another book on software testing states "Don't insist that every bug be fixed ... When the programmer fixes a minor bug, he might create a more serious one." Corporations may keep critical bugs off the books to limit legal liability. Only those bugs should be corrected that affect profit. The tools designed to root out bugs are themselves full of bugs. "Indeed, test tools are often buggier than comparable (but cheaper) development tools" [18]. As for hardware reliability, former Intel President Andy Grove himself said "I have come to the conclusion that no microprocessor is ever perfect; they just come closer to perfection ..." [20, p. 221].

Bugs can be far-reaching. The bug causing the explosion of the Ariane 5 rocket cost hundreds of millions of dollars. As long ago as 1854, Thoreau wrote that "by the error of some calculator the vessel often splits upon a rock that should have reached a friendly pier." Last year, the *New York Times* reported Shamir's warning that even a small math error in a widely used computer chip could be exploited to defeat cryptography and would place "the security of the global electronic commerce system at risk ..." [27].

Mathematical Certainty

By contrast, philosophers tell us that mathematics consists of analytic truths, free of all imperfection. We prove that $1 + 1 = 2$ by recalling the definition of 1 as the successor of 0, 2 as the successor of 1, and then invoking twice the recursive definition of addition:

$$1 + 1 = 1 + S(0) = S(1 + 0) = S(1) = 2.$$

If only all proofs were so simple. Mathematical error is as old as mathematics itself. Euclid's very first proposition asks, "on a given straight line to construct an equilateral triangle." Euclid's construction makes the implicit assumption—not justified by the axioms—that two circles, each passing through the other's center, must intersect. We revere Euclid, not because he got everything right, but because he set us on the right path.

We have entered an era of proofs of extraordinary complexity. Take, for example, F. Almgren's masterpiece in geometric measure theory, called appropriately enough the "Big Paper". The preprint is 1728 pages long. Each line is a chore. He spent over a decade writing it in the 1970s and early 1980s. It was not published until 2000. Yet the theorem is fundamental. It establishes the regularity of minimizing rectifiable currents, up to codimension two; in basic terms, it shows that

Thomas C. Hales is Mellon Professor of Mathematics at the University of Pittsburgh. His email address is hales@pitt.edu.
The author's research on the Formal Foundations of Discrete Geometry has been supported by NSF grant 0503447. He thanks Mark Adams, K. Parkinson, B. Casselman, and F. Wiedijk for helpful comments. This article is dedicated to N. G. de Bruijn.

higher dimensional soap bubbles are smooth rather than jagged—just as one would naturally expect. How am I to develop enough confidence in the proof that I am willing to cite it in my own research? Do the stellar reputations of the author and editors suffice, or should I try to understand the details of the proof? I would consider myself very fortunate if I could work through the proof in a year.

Computer proofs, which are sprouting up in many fields, compound the complexity: the nonexistence of a projective plane of order 10, the proof that the Lorenz equations have a strange attractor, the double-bubble problem for minimizing soap bubbles enclosing two equal volumes, the optimality of the Leech lattice among 24-dimensional lattice packings, hyperbolic 3-manifolds, and the one that got it all started: the four-color theorem. What assurance of correctness do complex computer proofs provide?

Three Early Milestones

1954 – M. Davis programs the Presburger algorithm for additive arithmetic into the "Johniac" computer at the Institute for Advanced Study. Johniac proves that the sum of two even numbers is even, to usher in the era of computer proof.

1956 – The automation of Russell and Whitehead's *Principia Mathematica* begins [26]. By the end of 1959, Wang's procedure had generated proofs of every theorem of the *Principia* in the predicate calculus [30].

1968 – N.G. de Bruijn designs the first computer program to check the validity of general mathematical proofs. His program Automath eventually checked every proposition in a primer that Landau had written for his daughter on the construction of real numbers as Dedekind cuts.

N. G. de Bruijn

On April 24, 2008, F. Wiedijk and I visited N. G. de Bruijn at his home in Nuenen, shortly before his ninetieth birthday. (Nuenen is the Dutch town where Vincent van Gogh lived when he painted *The Potato Eaters*.) We discussed Automath, Brouwer, Heyting, and some of his coauthors (Knuth and Erdös). De Bruijn has contributed to many fields of mathematics, including analytic number theory, Penrose tilings, quasicrystals, and optimal control.

De Bruijn indices give a notation that eliminates all dummy variables from formulas with quantifiers: $\forall\, x.\ P(x)$ becomes $(\forall\ P\ 1)$. This notation solves the problem of free variable capture.

De Bruijn observed that the ratio of lengths of a formal proof to the corresponding conventional proof is remarkably stable across different proofs. The ratio, called the de Bruijn factor, has become the standard benchmark to measure the overhead of a formal proof.

Formal Proof

Traditional mathematical proofs are written in a way to make them easily understood by mathematicians. Routine logical steps are omitted. An enormous amount of context is assumed on the part of the reader. Proofs, especially in topology and geometry, rely on intuitive arguments in situations where a trained mathematician would be capable of translating those intuitive arguments into a more rigorous argument.

A formal proof is a proof in which every logical inference has been checked all the way back to the fundamental axioms of mathematics. All the intermediate logical steps are supplied, without exception. No appeal is made to intuition, even if the translation from intuition to logic is routine. Thus, a formal proof is less intuitive, and yet less susceptible to logical errors.

There is a wide gulf that separates traditional proof from formal proof. For example, Bourbaki's Theory of Sets was designed as a purely theoretical edifice that was never intended to be used in the proof of actual theorems. Indeed, Bourbaki declares that "formalized mathematics cannot in practice be written down in full" and calls such a project "absolutely unrealizable". The basic trouble with various foundational systems is that meta-mathematical arguments (for example, abbreviations that are external to the system or inductions over the syntactical form of an expression) are usually introduced early on, and without these simplifying meta-arguments, the vehicle stalls, never making it up the steep incline from primitive notions to high-level concepts. The gulf can be extreme: A. Matthias has calculated that to expand the definition of the number "1" fully in terms of Bourbaki primitives requires over 4 trillion symbols. In Bourbaki's view, the foundations of mathematics are roped-off museum pieces to be silently appreciated, but not handled directly.

There is an opposing view that regards the foundational enterprise as unfinished until it is realized in practice and written down in full. This article sketches the current state of this endeavor. It has been necessary to commence afresh, and to retool the foundations of mathematics for practical efficiency, while preserving its reliability and austere beauty. For anything beyond a trivial proof, the number of logical inferences is so large that a computer is used to ensure that no steps

Year	Theorem	Proof System	Formalizer	Traditional Proof
1986	First Incompleteness	Boyer-Moore	Shankar	Gödel
1990	Quadratic Reciprocity	Boyer-Moore	Russinoff	Eisenstein
1996	Fundamental - of Calculus	HOL Light	Harrison	Henstock
2000	Fundamental - of Algebra	Mizar	Milewski	Brynski
2000	Fundamental - of Algebra	Coq	Geuvers et al.	Kneser
2004	Four-Color	Coq	Gonthier	Robertson et al.
2004	Prime Number	Isabelle	Avigad et al.	Selberg-Erdös
2005	Jordan Curve	HOL Light	Hales	Thomassen
2005	Brouwer Fixed Point	HOL Light	Harrison	Kuhn
2006	Flyspeck I	Isabelle	Bauer-Nipkow	Hales
2007	Cauchy Residue	HOL Light	Harrison	classical
2008	Prime Number	HOL Light	Harrison	analytic proof

Table 1. Examples of formal proofs.

are omitted. This raises basic questions about trust in computers. This article also places formal proofs within a broader context of automating more general mathematical tasks.

As the art is currently practiced, each formal proof starts with a traditional mathematical proof, which is rewritten in a greatly expanded form, where all the assumptions are made explicit and all cases are treated in full. For example, a traditional mathematical proof might show that a graph is planar by drawing the graph on a sheet of paper. The expanded form of the proof replaces the picture by careful argument. From the expanded text, a computer script is prepared, which generates all the logical inferences of the proof. The transcription of a single traditional proof into a formal proof is a major undertaking.

Examples

Computer proof assistants have been under development for decades (see Box "Early Milestones"), but only recently has it become a practical matter to prove major theorems formally. The most spectacular example is Gonthier's formal proof of the four-color theorem. His starting point is the second-generation proof by Robertson et al. Although the traditional proof uses a computer and Gonthier uses a computer, the two computer processes differ from one another in the same way that a traditional proof differs from a formal proof. They differ in the same way that adding $1 + 1 = 2$ on a calculator differs from the mathematical justification of $1 + 1 = 2$ by definitions, recursion, and a rigorous construction of the natural numbers. In short, a large logical gulf separates them. As a result of Gonthier's formalization, the proof of the four-color theorem has become one of the most meticulously verified proofs in history.

In recent years, several other significant theorems have been formally verified. See Table 1. The table lists the theorems, which proof assistant was used (there are many to choose from), the person who produced a formal proof, and the mathematicians who produced the original proof. The Prime Number Theorem, asserting that the number of primes less than n is asymptotic to $n/\log n$, has two essentially different proofs: the elementary proof of Selberg and Erdös and the analytic proof of Hadamard and de la Vallée Poussin. Formal versions of both proofs have been produced. More ambitious projects are in store: Gonthier's team is now formalizing the Feit-Thompson odd order theorem, and the leading problem of the document *Ten Challenging Research Problems for Computer Science* is the formalization of the proof of Fermat's Last Theorem [3].

The Formal Jordan Curve Theorem

$$\begin{aligned}&\forall C.\ \mathit{simple\_closed\_curve\ top2}\ C \Rightarrow\\ &\quad (\ \exists A\,B.\ \mathit{top2}\ A \wedge \mathit{top2}\ B \wedge\\ &\qquad \mathit{connected\ top2}\ A \wedge \mathit{connected\ top2}\ B \wedge\\ &\qquad A \neq \emptyset \wedge B \neq \emptyset \wedge\\ &\qquad A \cap B = \emptyset \wedge A \cap C = \emptyset \wedge B \cap C = \emptyset \wedge\\ &\qquad A \cup B \cup C = \mathit{euclid}\ 2\ )\end{aligned}$$

The box above displays the statement of the Jordan Curve theorem, in computer readable form, as it appears in the formal proof. The complete specification of the theorem should also list all definitions, all the way back to the primitives. Without giving the detailed definitions here, we note that *top2* refers to the standard topology on the plane; *top2 A* indicates that A is an open set in the plane; *euclid* 2 is the Euclidean plane; and

connected top2 A means that A is a connected set in the plane.

A large library is maintained of all previously established proofs in the system, and anyone may use any result that has been previously established. Although every step of every proof is always checked, as researchers contribute to the system, interaction with the system gradually moves away from the primitive foundations towards something more closely resembling the high-level practice of mathematicians. The hope is the system will eventually become sufficiently user-friendly to become a familiar part of the mathematical workplace, much as email, TEX, computer algebra systems, and Web browsers are today.

HOL Light

This section gives a brief introduction to one foundational system designed for doing mathematical proofs on a computer. The system is called HOL Light (an acronym for a lightweight implementation of Higher Order Logic). I have singled it out because of its simple design and because it is the system that I understand the best. Some understanding of the design of a simple system is helpful before turning to questions of soundness in the next section. HOL Light by itself is only a small part of the overall formal-theorem-proving landscape. There are several competing systems to choose from, built on various logical foundations, and with their own powerful features. People argue about the relative merits of the different systems much in the same way that people argue about the relative merits of operating systems, political loyalties, or programming languages. To some extent, preferences show a geographical bias: HOL in the UK, Mizar in Poland, Coq in France, and Isabelle in Germany and the UK.

The basic components of the HOL Light system are its types, terms, theorems, rules of inference, and axioms. Each is briefly described in turn. The HOL Light System box (next page) gives a summary of the entire system.

Types

Much day-to-day mathematics is written at a level of abstraction that is indifferent to its exact representation as sets. For example, it does not matter how an ordered pair is encoded as a set, as long as the ordered pair has the characteristic property

$$(x, y) = (x', y') \quad \Leftrightarrow \quad x = x' \text{ and } y = y'.$$

It is bad style to break the abstraction to write $2 \in (0, 1)$. This layer of abstraction is good news, because it allows us to shift from Zermelo-Fraenkel-Choice (ZFC) set theory to a different foundational system with equanimity and ease.

Many proof assistants are based on types. Types are familiar to computer programmers. In a typed computer language, 3 is an integer and $[1.0; 2.0; 3.0]$ is an array of floating point numbers. An attempt to add 3 to this array results in a type mismatch error, and the computer program will not compile. The type checking mechanism of programming languages conveniently detects many bugs at the time of compilation.

ZFC set theory has no such type checking mechanism. As de Bruijn puts it, "Theoretically, it seems perfectly legitimate to ask whether the union of the cosine function and the number e (the basis of natural logarithms) contains a finite geometry" [6]. Mathematicians have the good sense not to ask such questions. However, when moving mathematics to a computer, which is lacking in common sense, it is useful to introduce types into the foundations to prevent this kind of nonsense. By convention, a colon is written before the name of a type. For instance, we write the type of the real number e as $:\mathbb{R}$, or simply $e : \mathbb{R}$, to indicate that e is a real number. The cosine function has a different type $:\mathbb{R} \to \mathbb{R}$, or $\cos : \mathbb{R} \to \mathbb{R}$. The type of the union operator forces its two arguments to have the same type, so that an attempt to take the union of the cosine function with e is then flat out rejected.

HOL Light is a new axiomatic foundation with types, different from the usual ZFC. The types are presented in the HOL Light System box. There are only two primitive types, the boolean type *:bool* and an infinite type *:ind*. The rest are formed with type variables joined by arrows. A mechanism is also provided for creating a new type that is in bijection with a nonempty subset of an existing type, allowing the system to be extended with types for ordered pairs, integers, rational numbers, real numbers, and so forth.

Terms

Terms are the basic mathematical objects of the HOL Light system. The syntax is based on Church's λ-calculus, which uses the notation

$$\lambda x.\, f(x)$$

to represent the function that takes x to $f(x)$, what a mathematician would write as $f : \mathbb{N} \to \mathbb{N}$, $x \mapsto f(x)$. The name λ-calculus is derived from the use of the letter λ to mark function arguments. The HOL Light System box lays out the construction of terms.

In ZFC set theory, there is a bijection of sets

$$Z^{X \times Y} \simeq (Z^Y)^X.$$

In other words, a function $(x, y) \mapsto f(x, y)$ from the Cartesian product $X \times Y$ to Z can be viewed as a function on X that maps x to a function $f(x, \cdot) : Y \to Z$. The right-hand side of this bijection is called the curried form of the function (named after the logician Haskell Curry). In typed

The HOL Light System

HOL Light (Lightweight Higher Order Logic) is a foundational system designed for doing mathematical proofs on a computer. The notation is based on a typed λ-calculus.

1. Types: The collection of types is freely generated from *type variables* $:A, :B, \ldots$ and *type constants* $:bool$ (boolean), $:ind$ (infinite type), joined by *arrows* ($\to$). The colon is used as a notational device to indicate a type. For example, $:bool$, $:bool \to A$, and $:(bool \to A) \to (ind \to B)$ are types.

2. Terms: The collection of terms is freely generated from *variables* $x, y, \ldots$ and *constants* $0, \ldots$ using *abstraction* ($\lambda x.t$ where x is a variable and t a term) and *application* ($f(x)$ for compatibly typed terms x and f). Each term has a type. The notation $x{:}A$ indicates that the type of term x is $:A$. Variables and constants are assigned a type at the moment of creation; the types of abstractions and applications are defined recursively: the type of $\lambda x.t$ is $:A \to B$ when $x{:}A$ and $t{:}B$; the type of $f(x)$ is $:B$ if $f{:}A \to B$ and $x{:}A$.

3. Theorems: A theorem is a *sequent* $\{p_1, \ldots, p_k\} \vdash q$, where $p_1, \ldots, p_k, q$ are terms of type $:bool$. The terms $p_1, \ldots, p_k$ are called the assumptions and q is called the conclusion of the sequent. The design of the system prevents the construction of theorems except through inferences from existing theorems, new definitions, and axioms.

4. Inference Rules: The system has ten inference rules and a mechanism for defining new constants and types. Each inference rule is depicted as a fraction; the inputs to the rule are listed in the numerator, and the output in the denominator. The inputs to the rules may be terms or other theorems. In the following rules, we assume that p and p' are equal, up to a renaming of bound variables, and similarly for b and b'. (Such terms are called α-equivalent.)

On first reading, ignore the assumption lists Γ and Δ. They propagate silently through the inference rules, but are really not what the rules are about. When taking the union $\Gamma \cup \Delta$, α-equivalent assumptions should be considered as equal.

Equality is reflexive:

$$\frac{}{\vdash a = a}$$

The application of the function $x \mapsto a$ to x gives a:

$$\frac{}{\vdash (\lambda x.\ a)\, x = a}$$

Equality is transitive:

$$\frac{\Gamma \vdash a = b;\ \Delta \vdash b' = c}{\Gamma \cup \Delta \vdash a = c}$$

Assume p, then conclude p:

$$\frac{p{:}bool}{p \vdash p}$$

Equal functions applied to equals are equal:

$$\frac{\Gamma \vdash f = g;\ \Delta \vdash a = b}{\Gamma \cup \Delta \vdash f\, a = g\, b}$$

An "equality-based" rule of modus ponens holds:

$$\frac{\Gamma \vdash p;\ \Delta \vdash p' = q}{\Gamma \cup \Delta \vdash q}$$

The rule of abstraction holds. Equal terms give equal functions:

$$\frac{x;\ \Gamma \vdash a = b}{\Gamma \vdash \lambda x.\ a\ = \lambda x.\ b}\ \text{(if } x \text{ is not free in } \Gamma)$$

If the assumption q gives conclusion p and the assumption p gives q, then they are equivalent:

$$\frac{\Gamma \vdash p;\ \Delta \vdash q}{(\Gamma \setminus q) \cup (\Delta \setminus p) \vdash p = q}$$

Type variable substitution holds. If arbitrary types are substituted in parallel for type variables in a sequent, a theorem results. Term variable substitution holds. If arbitrary terms are substituted in parallel for term variables in a sequent, a theorem results.

5. Mathematical Axioms: There are only three mathematical axioms.

Axiom of Extensionality: $\forall f.\quad (\lambda x.f\,x) = f.$

Axiom of Infinity: $\exists f{:}ind \to ind.\ (\text{ONE\_ONE}\, f) \wedge \neg(\text{ONTO}\, f).$

Axiom of Choice: $\forall P\, x.\quad Px \Rightarrow P(\varepsilon P).$

Extensionality asserts that every function is determined by its input-output relation. Dedekind's axiom of infinity asserts the existence of a function that is one-to-one but not onto. The Hilbert choice operator ε applied to a predicate P chooses a term that satisfies the predicate, provided the predicate is satisfiable.

systems, the curried form of multivariate functions is generally preferred. Treating X, Y, Z as types, we write the type of the curried function as $f : X \to (Y \to Z)$, or simply $f : X \to Y \to Z$.

The system has only two primitive constants. One of them[1] is the equality symbol $(=)$ of type $:A \to A \to bool$. That is, equality is a curried function that takes two arguments of the same type and returns the boolean type.

Axioms, Inference, and Theorems

There are three mathematical axioms: an axiom of extensionality that asserts that a function is determined by the values that it takes on all inputs, an axiom of infinity that asserts that the type $:ind$ is not finite, and an axiom of choice. The system has ten rules of inference, as described in the HOL Light System box. For example, the first two state that equality is reflexive and transitive. The final two rules of inference allow one to substitute new terms for the free variables in a theorem and allow one to substitute new types for the type variables in a theorem. Beyond these ten rules of inference are mechanisms for defining new constants and new types. A theorem is expressed in *sequent* form; that is, as a set of assumptions, followed by a conclusion.

Extending the Primitive System

This primitive system lacks the customary logical operators. There are no symbols for "and", "or", "not", and "implies." There are no universal or existential quantifiers. The set membership operator is absent. It is remarkable none of this is needed to express the rules of inference.

Logical operators are defined later. For example, the boolean constant T (true) can be defined as the conclusion of any theorem that has no assumptions. The most accessible yet jarringly iconoclast theorem comes from the reflexive law applied to equality itself:

$$\vdash (=)(=)(=).$$

Each new definition becomes a theorem. So then $\vdash T = ((=)(=)(=))$. Conjunction $(\wedge)$ is roundaboutly defined as the curried function that on boolean inputs p and q returns $(\lambda f.\ f\ p\ q) = (\lambda f.\ f\ T\ T)$; that is, conjunction yields true exactly when no curried function f is able to distinguish (p, q) from (T, T). The other logical operations are built with similar tricks.

[1] *The second constant is the Hilbert choice operator* (ε). *Recall that every term that is not a variable, a function application, or* λ*-abstraction is a constant. "Constancy" is thus a broader notion here than in first-order logic, and includes terms such as equality that take arguments. Parentheses are drawn around the equality symbol* $(=)$ *to denote the prefixed curried form, with* $(=)\,x\,x$ *an alternative syntax for* $x = x$.

The inference rules and axioms become bits of data that are processed by other computer procedures. For example, to give a formal proof that

$$2682440^4 + 15365639^4 + 18796760^4 = 20615673^4$$

a human is not required to type each primitive inference. An automated procedure takes any arithmetic identity as input, generates the inferences, and produces the theorem as output. A large number of such small decision procedures have been programmed into the system to handle routine tasks such as polynomial simplification, basic tautologies in logic, and decidable fragments of arithmetic. Procedures that automatically search for steps in a proof are also programmed into the computer. New procedures may be contributed by any user at any time to automate further tasks. The design of the kernel of the system prevents a rogue user from writing computer code that could compromise the soundness of the system.

All the basic theorems of mathematics up through the Fundamental Theorem of Calculus are proved from scratch on the user's laptop in about two minutes every time the system loads, so that the casual user does not need to be concerned with the low-level details. Basic facts of logic and elementary mathematics are simply there in the system to be used as needed.

Soundness

HOL Light is both an axiomatic system for doing mathematics and a computer program that implements the system. How trustworthy is it?

If the computer is set aside for a moment, and the axiomatic system alone analyzed, it is known to be consistent relative to ZFC. That is, an inconsistency in the HOL Light system would imply the inconsistency of ZFC.

Computer Implementation

> *You've got to prove the theorem-proving program correct. You're in a regression aren't you?*
>
> —*A. Robinson* [20, p. 288].

The more pressing question is the soundness and reliability of the computer program that implements the logic. An earlier section reported that a typical software program has approximately one bug per 100 lines of computer code. The most reliable software ever created, for example mission-critical software written for the space shuttle, has fewer than one bug per 10,000 lines of computer code. Various proof assistants vary widely in reliability, ranging from some of the world's most carefully crafted code at the upper end, to rubbish at the lower end. I confine my attention to the upper end.

The computer code that implements the axioms and rules of inference is referred to as the kernel of the system. It takes fewer than 500 lines of computer code to implement the kernel of HOL Light. (By contrast, a Linux distribution contains approximately 283 million lines of computer code.)

A bug anywhere in the kernel of this system might have fatal consequences. For example, if one of the axioms is incorrectly typed, it might lead to an inconsistent system.

Yes, it is a regress; but a rather manageable regress. The kernel is a tiny amount of computer code, but hundreds of thousands of lines of code are verified by the kernel. Eventually, there may be many millions of lines that are verified by this small kernel. The same kernel verifies everything from the prime number theorem to the correctness of hardware designs.

Since the kernel is so small, it can be checked on many different levels. The code has been written in a friendly programming style for the benefit of a human audience. The source code is available for public scrutiny. Indeed, the code has been studied by eminent logicians. By design, the mathematical system is spartan and clean. The computer code has these same attributes. A powerful type-checking mechanism within the programming language prevents a user from creating a theorem by any means except through this small fixed kernel. Through type-checking, soundness is ensured, even after a large community of users contributes further theorems and computer code. I wish to see a poster[2] of the lines of the kernel, to be taught in undergraduate courses, and published throughout the world, as the bedrock of mathematics. It is math commenced afresh as executable code.

Experience from other top-tier theorem-proving systems has been that about three to five bugs have been found in each system over a period of 15-20 years of use. After decades of use on many different systems, to my knowledge, only one proof has ever had to be retracted as a result of a bug in a theorem-proving system, and this in a system that I do not rank in the top-tier: in 1995 a heap overflow error led to the false claim that the theorem-prover REVEAL had solved the Robbins conjecture. We can assert with utmost confidence that the error rates of top-tier theorem-proving systems are orders of magnitude lower than error rates in the most prestigious mathematical journals. Indeed, since a formal proof starts with a traditional proof, then does strictly more checking even at the human level, it would be hard for the outcome to be otherwise.

As an extra check, J. Harrison gave what can almost be described as a formal proof in HOL Light of its own soundness [15]. To get around the self-referential limitations imposed by Gödel, he gave two separate proofs. In the first proof, a weakened version of HOL Light is created, without the axiom of infinity. The standard version is used to give a formal proof of the soundness of the weakened version. In the second proof, a strengthened version of HOL Light is created, with an additional axiom giving a large cardinal. The strengthened version then proves the standard version sound. These proofs go beyond traditional relative consistency proofs in logic in two respects. First of all, they are formal proofs, rather than conventional proofs. Second, the proofs establish not only the soundness of the logic, but also the underlying soundness of the computer code implementing the logic.[3]

[2] *A T-shirt has already been made!*

Export

In the past few years, a number of programs have been written to automatically translate a proof written in one system into a proof in another system. If a proof in one system is incorrect because of an underlying flaw in the theorem-proving program itself, then the export to a different system fails, and the underlying flaw is exposed. (Except of course, unless the second theorem-proving program also has a bug that is perfectly aligned with the bug in the first system. Since these systems are largely independently designed and implemented, the events of failure in different systems are treated as nearly independent, so that the probability of a perfect alignment of failures across n systems, goes to zero roughly as p^n, where p is the individual failure rate.)

Consider what happens when the proof of the soundness of HOL Light is exported. (This has not happened yet, but should happen soon.) The exported proof is a formal proof within a second theorem-prover that the HOL Light logic and implementation are sound. It will soon be within reach for several systems to give proofs of one another's soundness. When this is achieved, the probability of a false certification of a pseudo-proof is pushed an order of magnitude closer to zero. With a computer—indeed with any physical artifact, whether a codex, transistor, or a flash drive made of proteins from salt-marsh bacteria—it is never a matter of achieving philosophical certainty. It is a scientific knowledge of the regularity of nature and human technology, akin to the scientific evidence that Planck's constant $\hbar$ lies reliably within its experimental range. Technology can push the probability of a false certification ever closer to zero: 10^{-6}, 10^{-9}, 10^{-12}, ... The intent is that one

[3] *The soundness of the computer code is considered relative to a semantic model of the underlying programming language. This model may differ from the real-world behavior of the programming language, a reminder that the task of verification is never complete.*

day a system will store a million proofs without so much as a misplaced semicolon.

A bug in the compiler, operating system, or underlying hardware has the potential to compromise a formal proof. To minimize such bugs, formal proofs can be made about the correctness of the ambient computational environment. Indeed, verification of hardware design, compilers, and computer languages has long been one of the principal aims of formal methods. HOL itself was initially created for hardware verification. As early as 1989, a simple computer system from high-level language down to microprocessor was "formally specified and mechanically verified" [4]. Today, the semantics of various high-level programming languages have been defined with complete mathematical rigor [23]. In recent work that is nothing short of spectacular, X. Leroy has developed a formally verified compiler for the C programming language [19]. (When the target of a formal verification is a piece of computer code, rather than a standard mathematical text, the formalization checks that the computer code conforms to a precise specification of the algorithm; certifying that the computer code is bug free.)

Full Automation

Formal proofs are part of a larger project of automating all mechanizable mathematical tasks, from conjecture making to concept formation. This section touches on the problem of fully automated proofs—the discovery of proofs entirely by computer without any human intervention. The next section briefly describes the ultimate challenge of producing an automated mathematician. Progress has been gradual. Fifty years ago it was famously predicted that within a decade "a digital computer will discover and prove an important new mathematical theorem." This did not happen as scheduled.

Most success has been with the development of algorithms to solve special classes of problems. The WZ algorithm gives automated proofs of identities of hypergeometric sums. Gröbner basis methods solve ideal membership problems. Wu's geometry algorithm proves theorems such as Pappus' theorem and Pascal's theorem on the ellipse. Tarski's algorithm solves problems that can be formulated in the first-order language of the real numbers. The list of specialized algorithms is in fact enormous.

The most widely acclaimed example of a fully automated computer proof is the solution of the Robbins conjecture in 1996. The conjecture asserts that an alternative definition is equivalent to the usual definition of a Boolean algebra. It is remarkable because the solution does not involve any human assistance, specialized algorithms, or software designed with this particular problem in mind. Just type the problem into W. McCune's general purpose theorem prover *EQN*, hit return, and wait eight days for the solution to appear [21], [22].

Yet the story is only a qualified success. It has remained almost an isolated example, rather than the first in a torrent of results. The conjecture itself has the rather special form of a word problem in an abstractly defined algebraic system—a type of problem particularly suited for computer search. The proof that was found by computer can be expressed as a short yet non-obvious sequence of substitutions. (See box on next page.)

Overall, the level today of fully automated computer proof (lying outside special purpose algorithms) remains that of undergraduate homework exercises: a group in which every element has order two is necessarily abelian; Cantor's theorem asserting that a set is not in bijection with its powerset; if some iterate of a function has a unique fixed point, then the function has a fixed point; the base e for natural logarithms is irrational [1], [2]. Because of current limitations, fully automated proof tools generally serve to fill in intermediate steps of a larger formal proof. They are not ready to take on the Riemann hypothesis.

Automated Discovery

What happens if one sets aside rigor, and lets a computer explore? A groundbreaking project was D. Lenat's 1976 Stanford thesis. His computer program AM (for Automated Mathematician) was designed to discover new mathematical concepts. When AM was set loose to explore in the wild, it discovered the concepts of natural number, addition, multiplication, prime numbers, Pythagorean triples, and even the fundamental theorem of arithmetic. The thesis touched off a firestorm of criticism and praise.

To put AM in context, consider a hypothetical program that is instructed to discover new concepts by deleting conditions from the list of axioms defining a finite abelian group. The computer will then immediately discover the concepts of infinite group, nonabelian group, monoid, and so forth because these concepts all arise as subsets of the axioms. These discoveries could be sensationalized: *A program in Artificial Intelligence has made the ultimate leap from the finite to infinite, and from the abelian to the nonabelian, rediscovering fundamental concepts in seconds that mathematicians have grappled with for centuries.* There are nagging questions about the emptiness of AM's discoveries; a suggestive representation of the problem gives the answer away.

More recent projects stir the imagination, even if the field is still young. Computer programs have generated over one thousand conjectures in graph theory, expressing numerical relationships between different graph invariants. One

Full Automation of the Robbins Conjecture

Let S be a nonempty set with an associative commutative binary operation $(x, y) \mapsto xy$ and a unary operation $x \mapsto [x]$ (which, for convenience, we write synonymously as $x \mapsto \bar{x}$). The Robbins conjecture (in Winker form) asserts that the general Robbins identity

$$[[ab][a\bar{b}]] = a$$

implies the existence of $c, d \in S$ such that $[cd] = \bar{c}$. Here is the original proof that EQN discovered, as reconstructed in [10].

Proof. A solution is $c = x^3u, d = xu$, where $u = [x\bar{x}]$ and x is arbitrary. Abbreviate $j = [cd], e = u[x^2]\bar{c}$. Over the equality sign, a prime indicates a direct application of the Robbins identity; a superscript indicates a substitution of the numbered line; no superscript indicates a rewriting of abbreviations c, d, e, j, u.

$$\begin{aligned}
0 : [u[x^2]] \quad & = [[x\bar{x}][xx]] =' x.\\
1 : [xu[xu[x^2]\bar{c}]] \quad & =' [[[xux^2][xu[x^2]]][xu[x^2]\bar{c}]] = [[\bar{c}[xu[x^2]]][\bar{c}xu[x^2]]] =' \bar{c}.\\
2 : [u\bar{c}] \quad & = [u[x^2ux]] =^0 [u[x^2u[u[x^2]]]] =' [[[ux^2][u[x^2]]][x^2u[u[x^2]]]]\\
& =' [u[x^2]] =^0 x.\\
3 : [ju] \quad & = [[xcu]u] =' [[xcu][[uc][u\bar{c}]]] =^2 [[xcu][x[cu]]] =' x\\
4 : [x[x[x^2]u\bar{c}]] \quad & =' [[[x[u\bar{c}]][xu\bar{c}]][x[x^2]u\bar{c}]] =^2 [[[x^2][xu\bar{c}]][[x^2]xu\bar{c}]] =' [x^2]\\
5 : [x\bar{c}] \quad & =^1 [x[xu[xu[x^2]\bar{c}]]] =^0 [[u[x^2]][xu[xu[x^2]\bar{c}]]]\\
& = [[u[x^2]][ux[xe]]] =^4 [[u[x[xe]]][ux[xe]]] =' u\\
6 : [jx] \quad & =' [j[[xc][x\bar{c}]]] =^5 [j[[xc]u]] = [[uxc][u[xc]]] =' u\\
7 : [cd] \quad & = j =' [[j[x\bar{c}]][jx\bar{c}]] =^5 [[ju][jx\bar{c}]] =^3 [x[jx\bar{c}]] =^2 [[\bar{c}u][\bar{c}jx]]\\
& =^6 [[\bar{c}[jx]][\bar{c}jx]] =' \bar{c}.
\end{aligned}$$

□

open conjecture is described in the box "An Open Computer-Generated Conjecture". No technological barriers prevent us from unleashing conjecturing machines in all branches of mathematics, to see what moonshine they reveal.

Flyspeck

My interest in formal proofs grows out of a practical desire for a thorough verification of my own research that goes beyond what the traditional peer review process has been able to provide. A few years ago, I launched a project called *Flyspeck* to give a formal proof of the Kepler conjecture, asserting that no packing of congruent balls in three-dimensional Euclidean space can have density greater than the density of the face-centered cubic packing (also known as the cannonball arrangement). The name Flyspeck, which quite appropriately can mean to scrutinize, is derived from the acronym FPK, for the Formal Proof of the Kepler conjecture.

The original proof of this theorem was unusually difficult to check. In a letter of qualified acceptance for publication in the *Annals of Mathematics*, an editor described the process, "The referees put a level of energy into this that is, in my experience, unprecedented. They ran a seminar on it for a long time. A number of people were involved, and they worked hard. They checked many local statements in the proof, and each time they found that what you claimed was in fact correct. Some of these local checks were highly non-obvious at first, and required weeks to see that they worked out…They have not been able to certify the correctness of the proof, and will not be able to certify it in the future, because they have run out of energy to devote to the problem." In addition to a 300-page text, the proof relies on about forty thousand lines of custom computer code. To the best of my knowledge, the computer code was never carefully examined by the referees. The policy of the *Annals of Mathematics* states, "The human part of the proof, which reduces the original mathematical problem to one tractable by the computer, will be refereed for correctness in the traditional manner. The computer part may not be checked line-by-line, but will be examined for the methods by which the authors have eliminated or minimized possible sources of error…"

Ultimately, the mathematical corpus is no more reliable than the processes that assure its quality. A formal proof attains a much higher level of quality control than can be achieved by "local checks" and an "examination of methods".

Flyspeck may take as many as twenty work-years to complete. S. Obua and G. Bauer have already defended Ph.D. theses on the project. Together with the work of their advisor T. Nipkow (one of the principal architects of the Isabelle proof assistant),

nearly half of the computer code used in the proof of the Kepler conjecture is now certified.

An Open Computer-Generated Conjecture

Let G be a finite graph with the following properties:

(1) It has at least two vertices.
(2) The graph is simple; that is, there are no loops or multiple joins.
(3) It is regular; that is, every vertex has the same degree.
(4) The graph is connected.

For example, the complete graph (the graph with an edge between every two vertices) on n vertices has these properties, when $n \geq 2$. Define the *total domination number* of G to be the size of the smallest subset of vertices such that every vertex of G is adjacent to some vertex in the subset. The *path covering number* is the size of the smallest partition of the vertices into subsets, such that there exists a path confined to each subset S that steps through each vertex of S exactly once (that is, the induced graph on S has a Hamiltonian path).

The computer program Graffiti.pc conjectures that *the total domination number of G is at least twice the path covering number of G*. For example, the complete graph on n vertices has path covering number one, because it has a Hamiltonian path. Its total domination number is two (take any two vertices). The conjecture is sharp in this case by these direct observations [9].

QED

The Flyspeck project is a minute speck in the overarching Q.E.D. project (an anonymous manifesto declaring that all significant mathematical results should be preserved in a vast library of formal proofs). The labor required to realize such a library would be staggering. In the *Notices* in 1991, de Bruijn proposed an assembly line to turn mathematical ideas into formally verified proofs [7]. The standard benchmark for the human labor to transcribe one printed page of textbook mathematics into machine-verified formal text is one week, or US$150 per page at an outsourced wage. To undertake the formalization of just 100,000 pages of core mathematics would be one of the most ambitious collaborative projects ever undertaken in pure mathematics, the sequencing of a mathematical genome. One might imagine a massive wiki collaboration that settles the text of the most significant theorems in contemporary mathematics from Poincaré to Sato-Tate.

Outsourcing is the brute force solution to the Q.E.D. manifesto. Most researchers, however, prefer beauty over brute force; we may hope for advances in our understanding that will permit us someday to convert a printed page of textbook mathematics into machine-verified formal text in a matter of hours, rather than after a full week's labor. As long as transcription from traditional proof into formal proof is based on human labor rather than automation, formalization remains an art rather than a science. Until that day of automation, we fall short of a practical understanding of the foundations of mathematics.

Recommended Reading and Software

By far the best overview of the subject is the book *Mechanizing Proof,* winner of the 2003 Merton Book Award of the American Sociological Association [20]. The Q.E.D. Manifesto can be found at [29]. Historical surveys include [5], [13], [11], and [25]. For something more comprehensive, see [14].

Several theorem proving systems are extensively documented and are available for download, including HOL Light [12], Isabelle [16], Coq [8], Mizar [24], TPS, PVS, ACL2, NuPRL, and MetaPRL. A Web-browser version of Coq allows one to experiment with a proof assistant without downloading any software [28].

References

[1] P. B. Andrews and C. Brown, Proving theorems and teaching logic with TPS and ETPS, *Bulletin of Symbolic Logic* **11**(1), March 2005.

[2] Michael Beeson, Automatic generation of a proof of the irrationality of e, *Journal of Symbolic Computation*, **32**(4) (2001), 333–349.

[3] J. Bergstra, *Nationale Onderzoeksagenda Informatie en Communicatietechnologie* (NOAG-ict) 2005–2010, Albani drukkers, Den Haag, 2005.

[4] W. R. Bevier, W. A. Hunt Jr., J. Strother Moore, and W. D. Young, An approach to ssystems verification, *Journal of Automated Reasoning* **5** (1989); 411–428, at pp. 422–423.

[5] W. W. Bledsoe and D. W. Loveland (eds.), *Automated Theorem Proving: After 25 Years*, Contemporary Mathematics, Vol. 29, AMS, Providence, RI, 1984.

[6] N. G. de Bruijn, On the Role of Types in Mathematics, `http://www.win.tue.nl/~wsdwnb/`, 1995.

[7] ———, Checking mathematics with computer assistance, *Notices of the AMS* **38**(1), January 1991.

[8] The Coq proof assistant, `http://coq.inria.fr/`.

[9] E. DeLaVina, Q. Liu, R. Pepper, B. Waller and D. B. West, On some conjectures of Graffiti.pc on total domination, *Congressus Numerantium* **185** (2007), 81–95.

[10] B. Fitelson, Using Mathematica to Understand the Computer Proof of the Robbins Conjecture, *Mathematica in Education and Research* (Winter 1998, Volume 7, No. 1).

[11] M. Gordon, From LCF to HOL: A short history, *Proof, language, and interaction: Essays in honour of Robin Milner*, 169–185, MIT, 2000.

[12] J. HARRISON, The HOL Light theorem prover, http://www.cl.cam.ac.uk/~jrh13/hol-light/index.html.
[13] _____, A short survey of automated reasoning, *Proceedings of AB 2007*, the Second International Conference on Algebraic Biology, Springer LNCS vol. 4545, pp. x–x, 2007.
[14] _____, *Handbook of Practical Logic and Automated Reasoning*, Cambridge University Press, to appear 2009, 704 pp.
[15] _____, Towards self-verification of HOL Light.
[16] Isabelle, http://isabelle.in.tum.de/.
[17] C. KANER, J. FALK, and H. NGUYEN, *Testing Computer Software*, Wiley, 1999.
[18] C. KANER, J. BACH, B. PETTICHORD, *Lessons Learned in Software Testing*, Wiley, 2001.
[19] X. LEROY, Formal certification of a compiler back-end, or: programming a compiler with a proof assistant, in *33rd ACM Symposium on Principles of Programming Languages*, ACM Press, 2006, pp. 42–54.
[20] D. MACKENZIE, *Mechanizing Proof*, MIT Press, Cambridge, MA, 2001.
[21] W. MCCUNE, Robbins Algebras are Boolean, http://www.cs.unm.edu/~mccune/papers/robbins/.
[22] _____, Solution of the Robbins Problem, *JAR* **19**(3) (1997), 263–276.
[23] R. MILNER, M. TOFTE, and R. HARPER, *The Definition of Standard ML*, MIT Press, 1990.
[24] Mizar Home Page, http://mizar.org/.
[25] R. MURAWSKI, The present state of mechanized deduction, and the present knowledge of limitations, *Studies in Logic, Grammar and Rhetoric* **9**(22) (2006), pp. 31–60.
[26] A. NEWELL, J. C. SHAW and H. A. SIMON, 1956, Empirical explorations of the Logic Theory Machine: A case study in heuristics, *Proc. Western Joint Computer Conf.*, **15**, pp. 218–239. Also in: Feigenbaum and Feldman (eds.), *Computers and Thought*, McGraw-Hill, 1963.
[27] JOHN MARKOFF, Adding math to list of security threats, *New York Times*, November 17, 2007.
[28] ProofWeb, http://prover.cs.ru.nl/login.php.
[29] The QED Manifesto, Automated deduction–CADE 12, Springer-Verlag, *Lecture Notes in Artificial Intelligence*, Vol. 814 (1994), pp. 238–251. http://www.cs.ru.nl/~freek/qed/qed.html.
[30] H. WANG, Computer Theorem Proving and Artificial Intelligence, in [5], 49–70.