

DOI:10.1145/2591012

**With the help of computational proof assistants, formal verification could become the new standard for rigor in mathematics.**

BY JEREMY AVIGAD AND JOHN HARRISON

# Formally Verified Mathematics

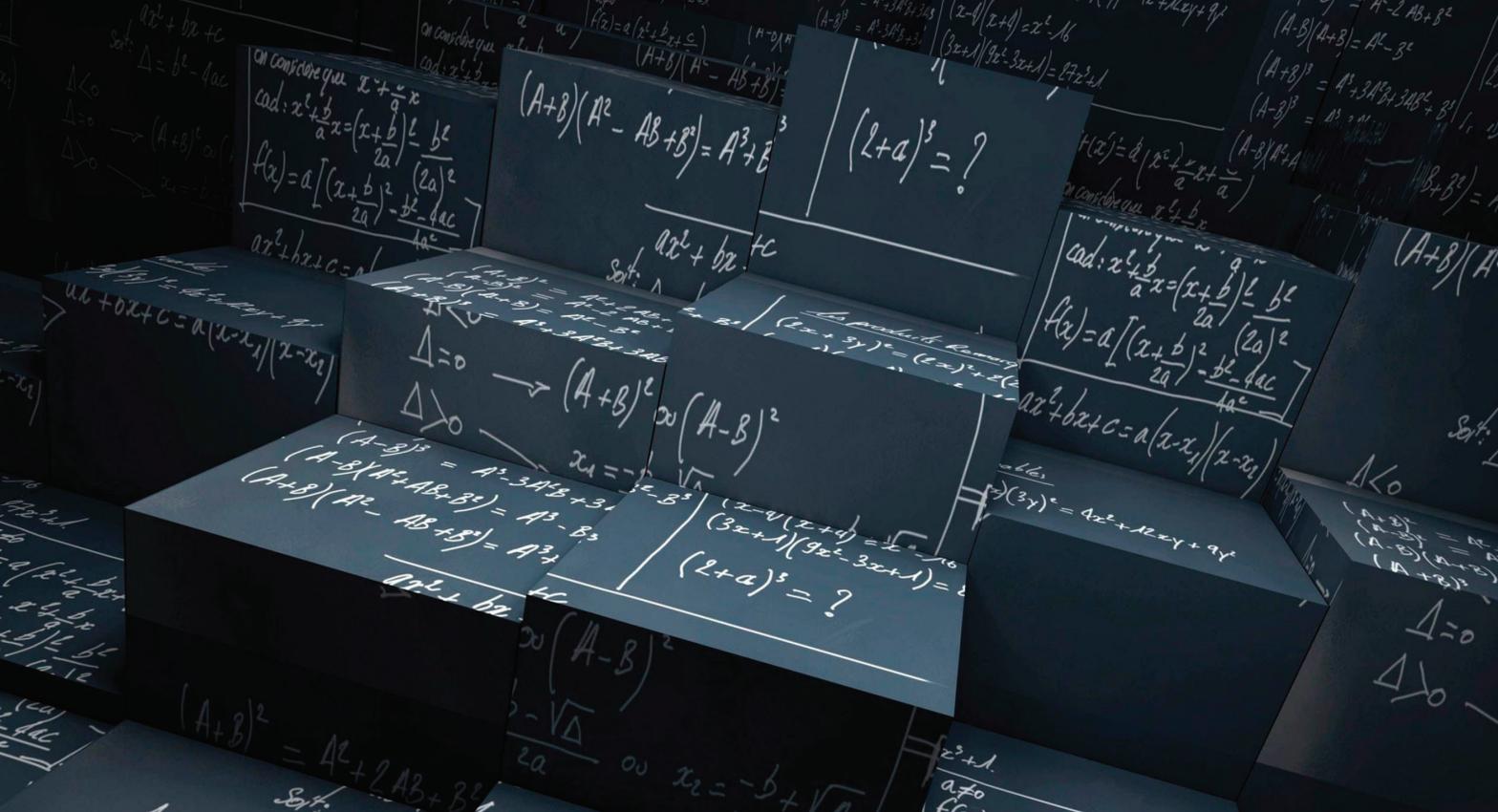
FROM THE POINT of view of the foundations of mathematics, one of the most significant advances in mathematical logic around the turn of the 20<sup>th</sup> century was the realization that ordinary mathematical arguments can be represented in formal axiomatic systems in such a way their correctness can be verified mechanically, at least in principle. Gottlob Frege presented such a formal system in the first volume of his *Grundgesetze der Arithmetik*, published in 1893, though in 1903 Bertrand Russell showed the system to be inconsistent. Subsequent foundational systems include the ramified type theory of Russell and Alfred North Whitehead's *Principia Mathematica*, published in three volumes from 1910 to 1913; Ernst Zermelo's axiomatic set theory of 1908, later extended by Abraham Fraenkel; and Alonzo Church's simple type theory of 1940. When Kurt Gödel presented his celebrated incompleteness theorems in 1931, he began with the following assessment:

“The development of mathematics toward greater precision has led, as is well known, to the formalization of large tracts of it, so one can prove any theorem using nothing but a few mechanical rules. The most comprehensive formal systems that have been set up hitherto are the system of *Principia Mathematica* on the one hand and the Zermelo-Fraenkel axiom system of set theory (further developed by J. von Neumann) on the other. These two systems are so comprehensive that in them all methods of proof used today in mathematics are formalized, that is, reduced to a few axioms and rules of inference. One might therefore conjecture that these axioms and rules of inference are sufficient to decide any mathematical question that can at all be formally expressed in these systems. It will be shown below that this is not the case...”<sup>4</sup>

Gödel was right to claim the mathematics of his day could generally be formalized in axiomatic set theory and type theory, and these have held up, to today, as remarkably robust foundations for mathematics. Indeed, set-theoretic language is now ubiquitous, and most mathematicians take the language and axioms of set theory to underwrite their arguments, in the sense that any ambiguities in a claim or proof could, in principle, be eliminated by spelling out the details in set-theoretic terms. In the mid-1930s, a group of mathematicians writing under the pen name Nicolas Bourbaki adopted set theory as the nominal foundation for a series of influential treatises aiming to provide a self-con-

## » key insights

- Among the sciences, mathematics is distinguished by its precise language and clear rules of argumentation.
- This fact makes it possible to model mathematical proofs as formal axiomatic derivations.
- Computational proof assistants make it possible to check the correctness of these derivations, thereby increasing the reliability of mathematical claims.



tained, rigorous presentation of the core branches of mathematics:

“...the correctness of a mathematical text is verified by comparing it, more or less explicitly, with the rules of a formalized language.”<sup>3</sup>

From this standpoint, the contemporary informal practice of writing a mathematical proof is an approximation to that ideal, whereby the task of a referee is to exercise professional judgment as to whether the proof could be expressed, in principle, in a way that conforms to the rules. The mathematician Saunders Mac Lane put it as follows:

“A Mathematical proof is rigorous when it is (or could be) written out in the first-order predicate language  $L(\in)$  as a sequence of inferences from the axioms ZFC, each inference made according to one of the stated rules... When a proof is in doubt, its repair is usually just a partial approximation to the fully formal version.”<sup>15</sup>

This point of view is to some extent anticipated by the 17<sup>th</sup> century philosopher Gottfried Leibniz, who called for development of a universal language (*characteristica universalis*) in which anything can be expressed and a calculus of reasoning (*calculus ratiocinator*) for deciding the truth of assertions expressed in the *characteristica*. Leibniz’s ambitions were not limited to mathematics; he dreamed

of a time when any disputants could translate their disagreement into the characteristic and say to each other “*calculemus*” (“let us calculate”). In the 20<sup>th</sup> century, however, even Bourbaki conceded complete formalization was an unattainable ideal:

“...the tiniest proof at the beginning of the Theory of Sets would already require several hundreds of signs for its complete formalization... formalized mathematics cannot in practice be written down in full... We shall therefore very quickly abandon formalized mathematics.”<sup>3</sup>

Due to developments in computer science over the past few decades, it is now possible to achieve complete formalization in practice. Working with “computational proof assistants,” users are able to verify substantial mathematical theorems, constructing formal axiomatic derivations of remarkable complexity. Our goal in this article is to describe the current technology and its motivations, survey the state of the art, highlight some recent advances, and discuss prospects for the future.

### Mathematical Rigor

The notion of proof lies at the heart of mathematics. Although early records of measurement and numeric computation predate the ancient Greeks, mathematics proper is commonly seen as having begun with de-

velopment of the deductive method, as exemplified by Euclid’s *Elements of Geometry*. Starting from axioms and postulates, a mathematical proof proceeds by a chain of incontrovertible logical steps to its conclusion. Through the ages, the method of the *Elements* was held to represent the paradigm of rigorous argumentation, to mathematicians, scientists, and philosophers alike. Its appeal is eloquently conveyed in John Aubrey’s short biography<sup>1</sup> of the philosopher Thomas Hobbes (1588–1697), who made his first serious contact with mathematics at the age of 40:

“Being in a Gentleman’s Library, Euclid’s *Elements* lay open, and ’twas the 47 *El. libri* 1 [Pythagoras’s Theorem]. He read the proposition. By G—, said he (he would now and then swear an emphaticall Oath by way of emphasis) *this is impossible!* So he reads the Demonstration of it, which referred him back to such a Proposition; which proposition he read. That referred him back to another, which he also read. *Et sic deinceps* [and so on] that at last he was demonstratively convinced of that trueth. This made him in love with Geometry.”<sup>1</sup>

The encounter turned Hobbes into an enthusiastic amateur geometer “wont to draw lines on his thigh and on the sheets, abed.” He became notorious in later years for bombarding

top mathematicians with his error-ridden ruler-and-compass geometric constructions, some of which are now known to be impossible in principle.

But what, exactly, constitutes a proof? Who is to say whether a proof is correct or not? Maintaining that a proof need convince only the person reading it gives the notion a subjective character. In practice, proofs tend to become generally accepted when they persuade not just one or two people but a broad or particularly influential group of mathematicians. Yet as Hobbes himself noted, this does not entirely avoid the subjective and fallible character of the judgment:

“But no one mans Reason, nor the Reason of any one number of men, makes the certaintie; no more than an account is therefore well cast up, because a great many men have unanimously approved it.”<sup>12</sup>

In the history of mathematics, there is no shortage of controversy over the validity of mathematical arguments. Berkeley’s extended critique of the methods of the calculus in *The Analyst* (1734) is one example. Another is the “vibrating string controversy” among Leonhard Euler, Jean d’Alembert, and Daniel Bernoulli, hinging on whether an “arbitrary” continuous function on a real interval could be represented by a trigonometric series. Carl Friedrich Gauss is usually credited with providing the first correct proof of the fundamental theorem of algebra, asserting that every nonconstant polynomial over the complex numbers has a root, in his doctoral dissertation of 1799; but the history of that theorem is especially knotty, since it was not initially clear what methods could legitimately be used to establish the existence of the roots in question. Similarly, when Gauss presented his proof of the law of quadratic reciprocity in his *Disquisitiones Arithmeticae* (1801), he began with the observation that Legendre’s alleged proof a few years prior contained a serious gap.

Mathematicians have always been reflectively conscious of their methods, and, as proofs grew more complex in the 19<sup>th</sup> century, mathematicians became more explicit in emphasizing the role of rigor. This is evident in, for example, Carl Jacobi’s praise of Johann Peter Gustav Lejeune Dirichlet:

“Dirichlet alone, not I, nor Cauchy, nor Gauss knows what a completely rigorous mathematical proof is. Rather we learn it first from him. When Gauss says that he has proved something, it is very clear; when Cauchy says it, one can wager as much pro as con; when Dirichlet says it, it is certain...” (quoted by Schubring<sup>21</sup>).

Mathematics has, at critical junctures, developed in more speculative ways. But these episodes are invariably followed by corresponding periods of retrenchment, analyzing foundations and increasingly adopting a strict deductive style, either to resolve apparent problems or just to make the material easier to teach convincingly.<sup>6</sup>

Reflecting on the history of mathematics, we can to some extent disentangle two related concerns. The first is whether the methods used in a given proof are valid, or appropriate to mathematics. This has to do with coming to consensus as to the appropriate rules of argumentation. For example, is it legitimate to refer to negative numbers, complex numbers, and infinitesimals, and, if so, what properties do they have? Is it legitimate to use the axiom of choice or to apply the law of the excluded middle to statements involving infinite structures? The second concern has to do with whether, given a background understanding of what is allowed, a particular proof meets those standards; that is, whether or not the proof is correct. The foundational debates of the early 20<sup>th</sup> century, and the set-theoretic language and formalism that emerged, were designed to address the question as to what methods are legitimate, by establishing an in-principle consensus as to the inferences that are allowed. Formal verification is not designed to help with that; just as human referees can strive only to establish correctness with respect to an implicit conception of what is acceptable, formal correctness can be assessed only modulo an underlying axiomatic framework. But, as will become clear in the next section, establishing correctness is a nontrivial concern, and is where formal methods come into play.

### Correctness Concerns

Mathematical proofs are complex objects, and becoming more so. Human

referees, even those with the best of intentions, are fallible, and mistakes are inevitably made in the peer-review process. A book written by Lecat in 1935 included 130 pages of errors made by major mathematicians up to 1900, and even mathematicians of the stature of J.E. Littlewood have published faulty proofs:

“Professor Offord and I recently committed ourselves to an odd mistake (*Annals of Mathematics Annals of Mathematics* (2) 49, 923, 1.5). In formulating a proof a plus sign got omitted, becoming in effect a multiplication sign. The resulting false formula got accepted as a basis for the ensuing fallacious argument. (In defence, the final result was known to be true.)”<sup>14</sup>

Every working mathematician must routinely deal with inferential gaps, misstatements, missing hypotheses, unstated background assumptions, imprecise definitions, misapplied results, and the like. A 2013 article in the *Notices of the American Mathematical Society* (Grcar<sup>7</sup>) on errors in the mathematical literature laments the fact that corrections are not published as often as they should be. Some errors are not easily repaired. The first purported proof of the four-color theorem in 1879 stood for a decade before a flaw was pointed out. Referees reviewing Andrew Wiles’s first proof of Fermat’s Last Theorem found a mistake, and it took Wiles and a former student, Richard Taylor, close to a year to find a way to circumvent it. Daniel Gorenstein announced, in 1983, that the classification of finite simple groups had been completed, unaware there was a gap in the treatment of the class of “quasisimple” groups. The gap was not filled until 2001, and doing so required a 1,221-page proof by Michael Aschbacher and Stephen Smith.

Even when an argument turns out to be correct, judging it to be so can take a long time. Grigori Perelman posted three papers on arXiv in 2002 and 2003 presenting a proof of Thurston’s geometrization conjecture. This result, in turn, implies the Poincaré conjecture, one of the Clay Mathematics Institute’s famed Millennium Prize challenges. The proof was scrutinized by dozens of researchers, but it was not until 2006 that three independent groups determined that

any gaps in Perelman's original proof were minor, and could be filled using the techniques he had developed.

The increased complexity is exacerbated by the fact that some proofs rely on extensive calculation. Kenneth Appel's and Wolfgang Haken's 1976 proof of the four-color theorem relied on an exhaustive computer enumeration of combinatorial configurations. Subsequent proofs, though more efficient, have this same character. Proofs that depend on explicit checking of cases are nothing new in themselves; for example, proofs of Bertrand's conjecture (for  $n \geq 1$  there is a prime  $n \leq p \leq 2n$ ) often begin with a comment like "Let us assume  $n \geq 4,000$ , since one can verify it explicitly for other cases." But this feature took on dramatic proportions with Thomas Hales's 1998 proof of the Kepler conjecture, stating that no packing of spheres in 3D space has higher density than the natural face-centered cubic packing commonly used to stack oranges, cannonballs, and such. Hales, working with Samuel Ferguson, arrived at a proof in 1998 consisting of 300 pages of mathematics and calculations performed by approximately 40,000 lines of computer code. As part of the peer-review process, a panel of 12 referees appointed by the *Annals of Mathematics* studied the proof for four full years, finally returning with the verdict that they were "99% certain" of the correctness, but in the words of the editor Robert MacPherson:

"The news from the referees is bad, from my perspective. 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. This is not what I had hoped for..."

"Fejes Tóth thinks that this situation will occur more and more often in mathematics. He says it is similar to the situation in experimental science—other scientists acting as referees can't certify the correctness of an experiment, they can only subject the paper to consistency checks. He thinks that the mathematical community will have to get used to this state of affairs."

The level of confidence was such that the proof was indeed published in the *Annals*, and no significant error has been found in it. Nevertheless, the



**Every working mathematician routinely has to deal with inferential gaps, misstatements, missing hypotheses, unstated background assumptions, imprecise definitions, misapplied results, and the like.**



verdict is disappointingly lacking in clarity and finality. In fact, as a result of this experience, the journal changed its editorial policy on computer-assisted proof so it will no longer even try to check the correctness of computer code. Dissatisfied with this state of affairs, Hales turned to formal verification, as we will see.

In November 2005, the *Notices of the American Mathematical Society* published an article by Brian Davies called "Whither Mathematics?" that raised questions about the mounting complexity of mathematical proof and the role of computers in mathematics. In August 2008, the *Notices* published an opinion piece by Melvyn Nathanson that also raised concerns about the status of mathematical proof:

"...many great and important theorems don't actually have proofs. They have sketches of proofs, outlines of arguments, hints and intuitions that were obvious to the author (at least, at the time of writing) and that, hopefully, are understood and believed by some part of the mathematical community."<sup>18</sup>

He concluded:

"How do we recognize mathematical truth? If a theorem has a short complete proof, we can check it. But if the proof is deep, difficult, and already fills 100 journal pages, if no one has the time and energy to fill in the details, if a 'complete' proof would be 100,000 pages long, then we rely on the judgments of the bosses in the field. In mathematics, a theorem is true, or it's not a theorem. But even in mathematics, truth can be political."<sup>18</sup>

Nathanson's essay did not explicitly mention contemporary work in formal verification. But a few months later, in December 2008, the *Notices* devoted an entire issue to formal proof, focusing on methods intended to alleviate Nathanson's concerns.

### **Automating Mathematics**

As noted, for suitable formal proof systems, there is a purely mechanical process for checking whether an alleged proof is in fact a correct proof of a certain proposition. So, given a proposition  $p$ , we could in principle run a search program that examines in some suitable sequence (in order of, say, length and then alphabetical order) every potential proof of  $p$  and termi-

nates with success if it finds one. That is, in the terminology of computability theory, the set of provable formulas is recursively enumerable.

But this naive program has the feature that if  $p$  is not provable, it will run fruitlessly forever, so it is not a yes/no decision procedure of the kind Leibniz imagined. A fundamental limitative result due to Church and Turing shows this cannot be avoided, because the set of provable formulas is not recursive (computable). This inherent limitation motivates the more modest goal of having the computer merely check the correctness of a proof provided (at least in outline form) by a person. Another reaction to limitative results like this, and related ones due to Gödel, Tarski, Post, and others, is to seek special cases (such as restricting the logical form of the problem) where a full decision procedure is possible.

All three possibilities have been well represented in the development of formal proof and automated reasoning:

- ▶ Complete but potentially nonterminating proof search;
- ▶ Decision procedures for special classes of problems; and
- ▶ Checking of proof hints or sketches given by a person.

In the category of general proof search, there were pioneering experiments in the late 1950s by Gilmore, Davis, Putnam, Prawitz, Wang, and others, followed by the more systematic development of practically effective proof procedures, including “tableaux” (Beth, Hintikka), “resolution” (Robinson, Maslov), and “model elimination” (Loveland), as well as more specialized techniques for “equational” reasoning, including “Knuth-Bendix completion.” Perhaps the most famous application of this kind of search is the proof of the Robbins conjecture, discovered by McCune<sup>17</sup> using the automated theorem prover *EQP* in 1996 that settled a problem that had been open since the 1930s.

In the category of decision procedures, perhaps the first real “automated theorem prover” was Davis’s procedure for the special case of Presburger arithmetic, a generalization of integer programming. Implementations of many other decision procedures have followed, motivating further theoretical developments. Some procedures



**Although automation is an exciting and ambitious goal, there is little realistic hope of having automated provers routinely prove assertions with real mathematical depth.**



have been particularly effective in practice (such as Gröbner basis algorithms and Wu’s method, both applicable to theorem proving in geometry).

Although automation is an exciting and ambitious goal, there is little realistic hope of automated provers routinely proving assertions with real mathematical depth. Attention has thus focused on methods of verification making use of substantial interaction between mathematician and computer.

### **Interactive Theorem Proving**

The idea behind interactive theorem proving is to allow users to work with a computational “proof assistant” to convey just enough information and guidance for the system to be able to confirm the existence of a formal axiomatic proof. Many systems in use today actually construct a formal proof object, a complex piece of data that can be verified by independent checkers.

One of the earliest proof systems was de Bruijn’s *Automath*, which appeared in the late 1960s. The computational assistance it rendered was minimal, since the project’s emphasis was on developing a compact, efficient notation for describing mathematical proof. An early milestone was Jutting’s 1977 Ph.D. thesis in which he presented a complete formalization of Landau’s book on a foundational construction of the real numbers as “Dedekind cuts,” deducing that the reals so constructed are a complete ordered field.

Proof checkers soon came to incorporate additional computer assistance. Andrzej Trybulec’s *Mizar* system, introduced in 1973 and still in use today, uses automated methods to check formal proofs written in a language designed to approximate informal mathematical vernacular. The Boyer-Moore *NQTHM* theorem prover (an ancestor of *ACL2*), also actively used today, was likewise introduced in the early 1970s as a fully automatic theorem prover; in 1974 the project’s efforts shifted to developing methods of allowing users to prove facts incrementally, then provide the facts as “hints” to the automated prover in subsequent proofs.

Influential systems introduced in the 1980s include Robert Constable’s *Nuprl*, Mike Gordon’s *HOL*, and the

Coq system, based on a logic developed by Thierry Coquand and Gerard Huet, and, in the 1990s, Lawrence Paulson's *Isabelle* and the *Prototype Verification System*, or PVS, developed by John Rushby, Natarjan Shankar, and Sam Owre.<sup>a</sup> By 1994, William Thurston could write the following in an article in the *Bulletin of the American Mathematical Society*:

"There are people working hard on the project of actually formalizing parts of mathematics by computer, with actually formally correct formal deductions. I think this is a very big but very worthwhile project, and I am confident we will learn a lot from it."<sup>23</sup>

Many of these systems are based on an architecture developed by Robin Milner with his 1972 proof LCF proof checker, which implemented Dana Scott's *Logic of Computable Functions*. An LCF-style prover is based on a small, trusted core of code used to construct theorems by applying basic rules of the axiomatic system. Such a system can then include more elaborate pieces of code built on top of the trusted core, to provide more complex proof procedures that internally decompose to (perhaps many) invocations of basic rules. Correctness is guaranteed by the fact that, ultimately, only the basic rules can change the proof state; everything the system does is mediated by the trusted core. (This restriction is often enforced by using a functional programming language like ML and OCaml and implementing the basic inference rules as the only constructors of an abstract data type.)

Many provers support a mode of working where the theorem to be proved is presented as a "goal" that is transformed by applying "tactics" in a backward fashion; for example, Figure 1 is a proof of the fact that every natural number other than 1 has a prime divisor in the *Isabelle* proof assistant.

The "lemma" command establishes the goal to be proved, and the first instruction invokes a form of complete induction. The next two statements split the proof into two cases, depending on whether or not  $n = 0$ . When  $n = 0$ ,

setting  $p = 2$  witnesses the conclusion; the system confirms this automatically (using "auto"). Otherwise, the proof splits into two cases, depending on whether or not  $n$  is prime. If  $n$  is prime, the result is immediate. The case where  $n$  is not prime is handled by appealing to a previously proved fact.

A user can step through such a "procedural" proof script within the proof assistant itself to see how the goal state changes in successive steps. But the script is difficult to read in isolation, since the reader must simulate, or guess, the results of applying each tactic. Ordinary mathematical proofs tend to emphasize, in contrast, the intermediate statements and goals, often leaving the justification implicit. The *Mizar* proof language was designed to model such a "declarative" proof system, and

such features have been incorporated into LCF-style provers. For example, Figure 2 is a proof of the same statement, again in *Isabelle*, but written in a more declarative style.

A number of important theorems have been verified in the systems described here, including the prime number theorem, the four-color theorem, the Jordan curve theorem, the Brouwer fixed-point theorem, Gödel's first incompleteness theorem, Dirichlet's theorem on primes in an arithmetic progression, the Cartan fixed-point theorems, and many more.<sup>b</sup> In the next section we describe even more impressive milestones, though even more im-

b For a list of "100 great theorems," including those formalized in various systems, see <http://www.cs.ru.nl/~freek/100/>

Figure 1. An LCF tactic-style proof in Isabelle.

```
lemma prime_factor_nat: "n ~= (1::nat) ==> EX p. prime p & p dvd n"
  apply (induct n rule: nat_less_induct)
  apply (case_tac "n = 0")
  apply (rule_tac x = 2 in exI)
  apply auto
  apply (case_tac "prime n")
  apply auto
  apply (subgoal_tac "n > 1")
  apply (frule (1) not_prime_eq_prod_nat)
  apply (auto intro: dvd_mult dvd_mult2)
done
```

Figure 2. A declarative proof in Isabelle.

```
lemma prime_factor_nat: "n ~= (1::nat) ==> EX p. prime p & p dvd n"
proof (induct n rule: nat_less_induct)
  fix n :: nat
  assume "n ~= 1" and
  ih: "ALL m < n. m ~= 1 --> (EX p. prime p & p dvd m)"
  then show "EX p. prime p & p dvd n"
  proof -
    { assume "n = 0"
      hence "prime (2 :: nat) & 2 dvd n"
        by auto
      hence ?thesis by blast }
    moreover
    { assume "prime n"
      hence ?thesis by auto }
    moreover
    { assume "n ~= 0" and "~prime n"
      with `n ~= 1` have "n > 1" by auto
      with `~prime n` and not_prime_eq_prod_nat
      obtain m k where "n = m * k" and "1 < m" and "m < n" by blast
      with ih obtain p where "prime p" and "p dvd m" by blast
      with `n = m * k` have ?thesis by auto }
    ultimately show ?thesis by auto
  qed
qed
```

a For a more comprehensive list of provers, see <http://www.cs.ru.nl/~freek/digimath/index.html>; for an overview of 17 proof assistants in use today, see Wiedijk.<sup>26</sup>

portant are the bodies of mathematical theory that have been formalized, often on the way to proving such “big name” theorems. Substantial libraries have been developed for elementary number theory, real and complex analysis, measure theory and measure-theoretic probability, linear algebra, finite group theory, and Galois theory. Formalizations are now routinely described in journals, including the *Journal of Automated Reasoning*, *Journal of Formalised Reasoning*, and *Journal of Formalized Mathematics*. The annual *Interactive Theorem Proving* conference includes reports on formalization efforts and advances in interactive theorem proving technology.

Also worth noting is that interactive proof systems are also commonly used to verify hardware and software sys-

tinct from those in the verification of ordinary mathematics, and the details would take us too far afield. So, here, in this article, we deliberately set aside hardware and software verification, referring the reader to Donald MacKenzie’s book *Mechanizing Proof: Computing, Risk and Trust*<sup>16</sup> for a thoughtful exploration of the topic.

**Contemporary Efforts**

Interactive theorem proving reached a major landmark on September 20, 2012, when Georges Gonthier announced he and a group of researchers under his direction had completed a verification of the Feit-Thompson theorem. The project relied on the *Coq* interactive proof assistant and a proof language, *SSReflect*, Gonthier designed. The Feit-Thompson theo-

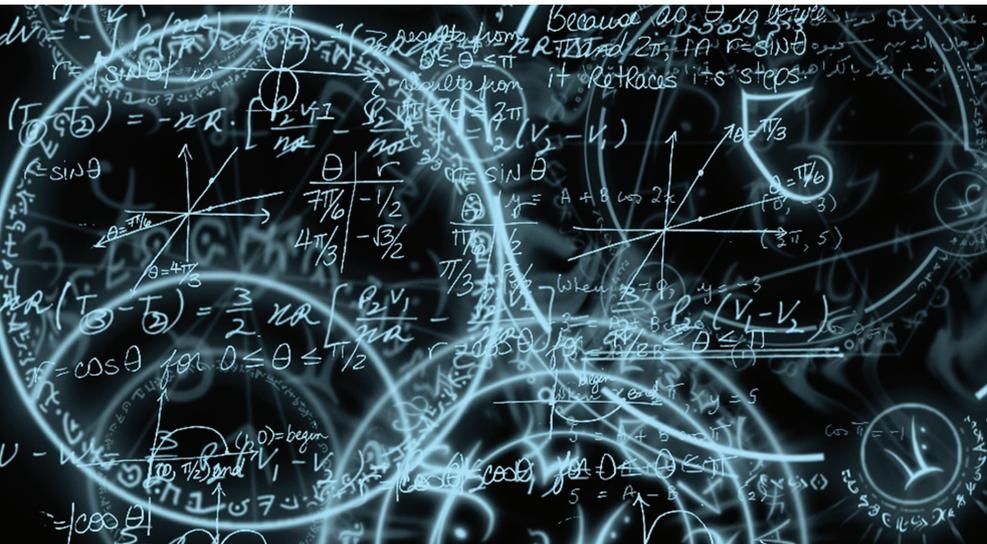
rem, sometimes called the odd-order theorem, says every finite group of odd order is solvable; equivalently, that the finite simple groups of odd order are exactly the cyclic groups of prime order. This theorem was an important first step in the classification of finite simple groups mentioned earlier. The original proof by Walter Feit and John Thompson, published in 1963, filled 255 journal pages. While a proof that long would not raise eyebrows today, it was unheard of at the time.

ditional interpretation. The resulting proof has approximately 150,000 lines of “code,” or formal proof scripts, including 4,000 definitions and 13,000 lemmas and theorems. As a basis for the formalization, Gonthier and his collaborators had to develop substantial libraries of facts about finite group theory, linear algebra, Galois theory, and representation theory. From there, they worked from a presentation of the Feit-Thompson theorem in two texts, one by Helmut Bender and George Glauberman describing the “local analysis,” the other by Thomas Peterfalvi describing a “character-theoretic” component. As one might expect, they had to cope with numerous errors and gaps, some not easy to fix, though none fatal.

Hales’s *Flyspeck* project is another ambitious formalization effort. In response to the outcome of the referee process at the *Annals*, Hales decided to formally verify a proof of the Kepler conjecture. (The name “*Flyspeck*” is a contraction of “Formal Proof of the Kepler Conjecture.”) He made it clear he viewed the project as a prototype:

“In truth, my motivations for the project are far more complex than a simple hope of removing residual doubt from the minds of few referees. Indeed, I see formal methods as fundamental to the long-term growth of mathematics.”<sup>9</sup>

The proof involves three essential uses of computation: enumerating a class of combinatorial structures called “tame hypermaps”; using linear-programming methods to establish bounds on a large number of systems of linear constraints; and using interval methods to verify approximately 1,000 nonlinear inequalities that arise in the proof. All this is in addition to the textual “paper” proof, which in and of itself is quite long and involves Euclidean measure theory, geometric properties of polyhedral, and various combinatorial structures. Partly as a result of the formalization effort, both the “paper” and “machine” parts of proof have been streamlined and reorganized.<sup>8</sup> The combination of non-trivial paper proofs and substantial time-consuming computational components make it a particularly difficult formalization challenge, yet after a substantial effort by a large, geograph-



tems. In principle, this is no different from verifying mathematical claims; for the purposes of formal verification, hardware and software systems must be described in mathematical terms, and the statement that such a system meets a certain specification is a theorem to be proved. Most of the systems described here are thus designed to serve both goals. The connections run deeper; hardware and software specifications often make sense only against background mathematical theory of, say, the integers or real numbers; and, conversely, methods of verifying software apply to the verification of code that is supposed to carry out specifically mathematical computations. However, hardware and software verification raises concerns largely dis-

rem, sometimes called the odd-order theorem, says every finite group of odd order is solvable; equivalently, that the finite simple groups of odd order are exactly the cyclic groups of prime order. This theorem was an important first step in the classification of finite simple groups mentioned earlier. The original proof by Walter Feit and John Thompson, published in 1963, filled 255 journal pages. While a proof that long would not raise eyebrows today, it was unheard of at the time.

Gonthier launched the project in 2006 with support from the Microsoft Research - Inria Joint Centre in Orsay, France. Because *Coq* is based on a constructive logic, Gonthier had to reorganize the proof in such a way every theorem has a direct computa-

ically distributed team, the project is nearing completion.

Another significant formal verification effort is the *Univalent Foundations* project introduced by Fields Medalist Vladimir Voevodsky.<sup>24</sup> Around 2005, Voevodsky, and independently Steve Awodey and Michael Warren, realized that constructive dependent type theory, the axiomatic basis for *Coq*, has an unexpected homotopy-theoretic interpretation. Algebraic topologists routinely study abstract spaces and paths between elements of those spaces; continuous deformations, or “higher-order” paths, between the paths; and deformations between the paths; and so on. What Voevodsky, Awodey, and Warren realized is that one can view dependent type theory as a calculus of topological spaces and continuous maps between them, wherein the assertion  $x = y$  is interpreted as the existence of a path between  $x$  and  $y$ .

More specifically, Voevodsky showed there is a model of constructive type theory in the category of “simplicial sets,” a mathematical structure well known to algebraic topologists. Moreover, this interpretation validates a surprising fact Voevodsky called the “univalence axiom,” asserting, roughly, that any two types that are isomorphic are identical. The axiom jibes with informal mathematical practice, wherein two structures that are isomorphic are viewed as being essentially the same. However, it is notably false in the universe of sets; for example, there is a bijection between the sets  $\{1, 2\}$  and  $\{3, 4\}$ , but the first set contains the element 1 while the second does not. Voevodsky has suggested that dependent type theory with the univalence axiom can provide a new foundation for mathematics that validates structural intuitions. There is also hope among computer scientists that univalence can provide a new foundation for computation where code can be designed to work uniformly on “isomorphic” data structures (such as lists and arrays) implemented in different ways.

In an excerpt from a grant proposal posted on his Web page, Voevodsky described his motivations for the project as follows:

“While working on the completion of the proof of the Block-Kato conjecture I have thought a lot about what to

do next. Eventually I became convinced that the most interesting and important directions in current mathematics are the ones related to the transition into a new era which will be characterized by the widespread use of automated tools for proof construction and verification.”

During the 2012–2013 academic year, the Institute for Advanced Study in Princeton, NJ, held a program on Univalent Foundations, drawing an interdisciplinary gathering of mathematicians, logicians, and computer scientists from all over the world.

### Rigor and Understanding

We have distinguished between two types of concern that can attend a mathematical proof: whether the methods it uses are appropriate to mathematics and whether the proof itself represents a correct use of those methods. However, there is yet a third concern that is often raised—whether a proof delivers an appropriate understanding of the mathematics in question. It is in this respect that formal methods are often taken to task; a formal, symbolic proof is for the most part humanly incomprehensible and so does nothing to augment our understanding. Thurston’s article<sup>23</sup> mentioned earlier added the following caveat:

“...we should recognize that the humanly understandable and humanly checkable proofs that we actually do are what is most important to us, and that they are quite different from formal proofs.”

The article was in fact a reply to an article by Jaffe and Quinn<sup>13</sup> in the *Bulletin of the American Mathematical Society* that proposed a distinction between “theoretical” mathematics, which has a speculative, exploratory character, and fully rigorous mathematics. The Jaffe-Quinn article drew passionate, heated responses from some of the most notable figures in mathematics, some rising to defend the role of rigor in mathematics, some choosing to emphasize the importance of a broad conceptual understanding. Given that both are clearly important to mathematics, the Jaffe-Quinn debate may come across as much ado about nothing, but the episode makes clear that many mathematicians are wary that excessive concern for rigor can displace mathematical understanding.

However, few researchers working in formal verification would claim that checking every last detail of a mathematical proof is the most interesting or important part of mathematics. Formal verification is not supposed to replace human understanding or the development of powerful mathematical theories and concepts. Nor are formal proof scripts meant to replace ordinary mathematical exposition. Rather, they are intended to supplement the mathematics we do with precise formulations of our definitions and theorems and assurances that our theorems are correct. One need only recognize, as we say here, that verifying correctness is an important part of mathematics. The mathematical community today invests a good deal of time and resources in the refereeing process in order to gain such assurances, and surely any computational tools that can help in that regard should be valued.

### The Quest for Certainty

In discussions of formally verified mathematics, the following question often arises: Proof assistants are complex pieces of software, and software invariably has bugs, so why should we trust such a program when it certifies a proof to be correct?

Proof assistants are typically designed with an eye toward minimizing such concerns, relying on a small, trusted core to construct and verify a proof. This design approach focuses concern on the trusted core, which consists of, for example, approximately 400 lines in Harrison’s *HOL light* system. Users can obtain a higher level of confidence by asking the system to output a description of the axiomatic proof that can be checked by independent verifiers; even if each particular verifier is buggy, the odds that a faulty inference in a formal proof can make it past multiple verifiers shrinks dramatically. It is even possible to use formal methods to verify the trusted core itself. There have been experiments in “self-verifications” of simplified versions of *Coq* by Bruno Barras and *HOL light* by Harrison,<sup>10</sup> as well as Jared Davis’s work on *Milawa*, a kind of bootstrapping sequence of increasingly powerful approximations to the *ACL2* prover.

To researchers in formal verification, however, these concerns seem

misplaced. When it comes to informal proof, mistakes arise from gaps in the reasoning, appeal to faulty intuitions, imprecise definitions, misapplied background facts, and fiddly special cases or side conditions the author failed to check. When verifying a theorem interactively, users cannot get away with any of this; the proof checker keeps the formalizer honest, requiring every step to be spelled out in complete detail. The very process of rendering a proof suitable for machine verification requires strong discipline; even if there are lingering doubts about the trustworthiness of the proof checker, formal verification delivers a very high degree of confidence—much higher than any human referee can offer without machine assistance.

Mathematical results obtained through extensive computation pose additional challenges. There are at least three strategies that can be used to verify such results. First, a user can rewrite the code that carries out the calculations so it simultaneously uses the trusted core to chain together the axioms and rules that justify the results of the computation. This approach provides, perhaps, the highest form of verification, since it produces formal axiomatic proofs of each result obtained by calculation. This is the method being used to verify the linear and nonlinear inequalities in the *Flyspeck* project.<sup>22</sup> The second strategy is to describe the algorithm in mathematical terms, prove the algorithm correct, then rely on the trusted core to carry out the steps of the computation. This was the method used by Gonthier in the verification<sup>5</sup> of the four-color theorem in *Coq*; because it is based on a constructive logic, *Coq*'s “trusted computing base” is able to normalize terms and thereby carry out a computation. The third strategy is to describe the algorithm within the language of the proof checker, then extract the code and run it independently. This method was used by Tobias Nipkow, Gertrud Bauer, and Paula Schultz<sup>19</sup> to carry out the enumeration of finite tame hypermaps in the *Flyspeck* project; ML code was extracted from formal definitions automatically and compiled. This approach to verifying the results of computation invokes additional layers of trust, that, for example, the extracted code is faith-



**Many mathematicians are wary that excessive concern for rigor can displace mathematical understanding.**



ful to the function described in axiomatic terms and the compiler respects the appropriate semantics. However, compared to using unverified code, this method provides a high degree of confidence as well.

Similar considerations bear on the use of automated methods and search procedures to support the formalization process; for example, one can redesign conventional automated reasoning procedures so they generate formal proofs as they proceed or invoke an “off-the-shelf” reasoning tool and try to reconstruct a formal proof from the output. With respect to both reasoning and computation, an observation that often proves useful is that many proof procedures can naturally be decomposed into two steps: a “search” for some kind of certificate and a “checking” phase where this certificate is verified.<sup>2</sup> When implementing these steps in a foundational theorem prover, the “finding” (often the difficult part) can be done in any way at all, even through an external tool (such as a computer algebra system<sup>11</sup>), provided the checking part is done in terms of the logical kernel; for example, linear programming methods can provide easily checked certificates that witness the fact that a linear bound is optimal. Similarly, semi-definite programming packages can be used to obtain certificates that can be used to verify nonlinear inequalities.<sup>20</sup> Along these lines, the *Flyspeck* project uses optimized, unverified code to find informative certificates witnessing linear and nonlinear bounds, then uses the certificates to construct fully formal justifications.<sup>22</sup> Such practices raise interesting theoretical questions about which symbolic procedures can in principle provide efficiently checkable certificates, as well as the pragmatic question of how detailed the certificates should be to allow convenient verification without adversely affecting the process of finding them.

### Prospects

We have not touched on many important uses of computers in mathematics (such as in the discovery of new theorems, exploration of mathematical phenomena, and search for relevant information in databases of mathematical facts). Correctness is only one important part of mathematics, as we have

emphasized, and the process of verification should interact continuously with other uses of formal methods. But even with this restriction, the issues we have considered touch on important aspects of artificial intelligence, knowledge representation, symbolic computation, hardware and software verification, and programming-language design. Mathematical verification raises its own challenges, but mathematics is a quintessentially important type of knowledge, and understanding how to manage it is central to understanding computational systems and what they can do.

The developments we have discussed make it clear that it is pragmatically possible to obtain fully verified axiomatic proofs of substantial mathematical theorems. Despite recent advances, however, the technology is not quite ready for prime time. There is a steep learning curve to the use of formal methods, and verifying even straightforward and intuitively clear inferences can be time consuming and difficult. It is also difficult to quantify the effort involved. For example, 15 researchers contributed to the formalization of the Feit-Thompson theorem over a six-year period, and Hales suspects the *Flyspeck* project has already exceeded his initial estimate of 20 person-years to completion. However, it is ultimately difficult to distinguish time spent verifying a theorem from time spent developing the interactive proof system and its libraries, time spent learning to use the system, and time spent working on the mathematics proper.

One way to quantify the difficulty of a formalization is to compare its length to the length of the original proof, a ratio known as the “de Bruijn factor.” Freek Wiedijk carried out a short study<sup>25</sup> and, in the three examples considered, the ratio hovers around a factor of four, whether comparing the number of symbols in plain-text presentations or applying a compression algorithm first to obtain a better measure of the true information content.

A better measure of difficulty is the amount of time it takes to formalize a page of mathematics, though that can vary dramatically depending on the skill and expertise of the person carrying out the formalization, density of the material, quality and depth of the supporting library, and formal-

izer’s familiarity with that library. In the most ideal circumstances, an expert can handle approximately a half page to a page of a substantial mathematical text in a long, uninterrupted day of formalizing. But most circumstances are far less than ideal; an inauspicious choice of definitions can lead to hours of fruitless struggle with a proof assistant, and a formalizer often finds elementary gaps in the supporting libraries that require extra time and effort to fill.

We should not expect interactive theorem proving to be attractive to mathematicians until the time it takes to verify a mathematical result formally is roughly commensurate with the time it takes to write it up for publication or the time it takes a referee to check it carefully by hand. Within the next few years, the technology is likely to be most useful for verifying proofs involving long and delicate calculations, whether initially carried out by hand or with computer assistance. But the technology is improving, and the work of researchers like Hales and Voevodsky makes it clear that at least some mathematicians are interested in using the new methods to further mathematical knowledge.

In the long run, formal verification efforts need better libraries of background mathematics, better means of sharing knowledge between the various proof systems, better automated support, better means of incorporating and verifying computation, better means of storing and searching for background facts, and better interfaces, allowing users to describe mathematical objects and their intended uses, and otherwise convey their mathematical expertise. But verification need not be an all-or-nothing proposition, and it may not be all that long before mathematicians routinely find it useful to apply an interactive proof system to verify a key lemma or certify a computation that is central to a proof. We expect within a couple of decades seeing important pieces of mathematics verified will be commonplace and by the middle of the century may even become the new standard for rigorous mathematics. □

#### References

1. Aubrey, J. *Brief Lives*. A. Clark, Ed. Clarendon Press, Oxford, U.K., 1898.
2. Blum, M. Program result checking: A new approach

- to making programs more reliable. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming* (Lund, Sweden, July 5–9), A. Lingas, R. Karlsson, and S. Carlsson, Eds. Springer, Berlin, 1993, 1–14.
3. Bourbaki, N. *Elements of Mathematics: Theory of Sets*. Addison-Wesley, Reading, MA, 1968; translated from the French *Théorie des ensembles* in the series *Éléments de mathématique*, revised version, Hermann, Paris, 1970.
4. Gödel, K. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik* 38, 1 (1931), 173–198; reprinted with English translation in Kurt Gödel: *Collected Works, Volume 1*, S. Feferman et al., Eds. Oxford University Press, Oxford, U.K., 1986, 144–195.
5. Gonthier, G. Formal proof—The four-color theorem. *Notices of the American Mathematical Society* 55, 11 (Dec. 2008), 1382–1393.
6. Grabner, J.V. Is mathematical truth time-dependent? In *New Directions in the Philosophy of Mathematics: An Anthology*, T. Tymoczek, Ed., Birkhäuser, Boston, 1986, 201–213.
7. Gracar, J.F. Errors and corrections in the mathematical literature. *Notices of the American Mathematical Society* 60, 4 (Apr. 2013), 418–432.
8. Hales, T. *Dense Sphere Packings: A Blueprint for Formal Proofs*. Cambridge University Press, Cambridge, U.K., 2012.
9. Hales, T. *The Kepler Conjecture* (unpublished manuscript), 1998; <http://arxiv.org/pdf/math/9811078.pdf>
10. Harrison, J. Towards self-verification of HOL Light. In *Proceedings of the Third International Joint Conference in Automated Reasoning* (Seattle, Aug. 17–20), U. Furbach and N. Shankar, Eds. Springer, Berlin, 2006, 177–191.
11. Harrison, J. and Théry, L. A sceptic’s approach to combining HOL and Maple. *Journal of Automated Reasoning* 21, 3 (1998), 279–294.
12. Hobbes, T. *Leviathan*. Andrew Crooke, London, 1651.
13. Jaffe, A. and Quinn, F. “Theoretical mathematics”: Toward a cultural synthesis of mathematics and theoretical physics. *Bulletin of the American Mathematical Society (New Series)* 29, 1 (1993), 1–13.
14. Littlewood, J.E. *Littlewood’s Miscellany*. Cambridge University Press, Cambridge, U.K., 1986.
15. Mac Lane, S. *Mathematics: Form and Function*. Springer, New York, 1986.
16. MacKenzie, D. *Mechanizing Proof: Computing, Risk and Trust*. MIT Press, Cambridge, MA, 2001.
17. McCune, W. Solution of the Robbins problem. *Journal of Automated Reasoning* 19, 3 (1997), 263–276.
18. Nathanson, M. Desperately seeking mathematical truth. *Notices of the American Mathematical Society* 55, 7 (Aug. 2008), 773.
19. Nipkow, T., Bauer, G., and Schultz, P. Flyspeck I: Tame graphs. In *Proceedings of the Third International Joint Conference in Automated Reasoning* (Seattle, Aug. 17–20). Springer, Berlin, 2006, 21–35.
20. Parrilo, P.A. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming* 96, 2 (2003), 293–320.
21. Schüring, G. Zur Modernisierung des Studiums der Mathematik in Berlin, 1820–1840. In *Amphora: Festschrift für Hans Wussing Zu Seinem 65. Geburtstag*, S.S. Demidov et al., Eds. Birkhäuser, Basel, 1992, 649–675.
22. Solovyev, A. *Formal Computation and Methods*. Ph.D. thesis, University of Pittsburgh, Pittsburgh, PA, 2012.
23. Thurston, W.P. On proof and progress in mathematics. *Bulletin of the American Mathematical Society (New Series)* 30, 2 (1994), 161–177.
24. Univalent Foundations Program, Institute for Advanced Study. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Princeton, NJ, 2013; <https://github.com/HoTT/book>
25. Wiedijk, F. *The de Bruijn Factor* (unpublished manuscript), 2000; <http://www.cs.ru.nl/~freek/factor/>
26. Wiedijk, F. *The Seventeen Provers of the World*. Springer, Berlin, 2006.

**Jeremy Avigad** (avigad@cmu.edu) is a professor in the Department of Philosophy and the Department of Mathematical Sciences at Carnegie Mellon University, Pittsburgh, PA.

**John Harrison** (johnh@chips.intel.com) is a principal engineer in Intel Corporation, Hillsboro, OR.