Machine-Assisted Proofs

James Davenport (moderator)* and Bjorn Poonen† and James Maynard‡ and Harald Helfgott§ and Pham Huu Tiep¶ and Luís Cruz-Filipe‖

7 August 2018

This panel took place on Tuesday 7th August 2018. After the moderator had introduced the topic, the panelists presented their experiences and points of view, and then took questions from the floor.

1 Introduction (James Davenport)

1.1 A (very brief, partial) history

1963 "Solvability of Groups of Odd Order": 254 pages[1] [20]. Also Birch & Swinnerton–Dyer published [7], the algorithms underpinning their conjectures.

1976 "Every Planar Map is Four-Colorable": 256 pages + computation [2].

1989 Revised Four-Color Theorem proof published [3].

1998 Hales announced proof of Kepler Conjecture.

2005 Hales' proof published in an abridged form "uncertified"[2] [26].

*Department of Computer Science, University of Bath, Bath BA2 7AY, U.K., J.H.Davenport@bath.ac.uk

†MIT Department of Mathematics 32 Vassar St., Bldg. 2-243 Cambridge, MA 02139, USA, poonen@math.math.edu

‡Professor of Number Theory, University of Oxford, UK

§Mathematisches Institut Bunsenstraße 3–5 D-37073 Göttingen, Germany; IMJ-PRG, UMR 7586, 58 avenue de France, Bâtiment S. Germain, case 7012, 75013 Paris CEDEX 13, France, harald.helfgott@gmail.com.

¶Department of Mathematics, Rutgers University, Piscataway, NJ 08854, USA tiep@math.rutgers.edu

‖Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark, lcfilipe@gmail.com

[1] "one of the longest proof to have appeared in the mathematical literature to that point." [24].

[2] *Mathematical Reviews* states "Nobody has managed to check all the details of the proof so far, but the theoretical part seems to be correct. The whole proof is considered and assumed to be correct by most of the mathematical community." https://mathscinet.ams.org/mathscinet-getitem?mr=2179728.

2008 Gonthier stated formal proof of Four-Color Theorem [22].

2012 Gonthier/Théry stated[3] formal proof of Odd Order Theorem [23, 24].

2013 Helfgott published (arXiv) proof of ternary Goldbach Conjecture [33].

2014 Flyspeck project announced formal proof of Kepler Conjecture [27].

2015 Maynard published "Small gaps between primes" [47].

2017 Flyspeck paper published [28].

The Odd Order Theorem is important, but chiefy because it leads to the classification of finite simple groups. One might ask when this will be formally proved, and indeed I did ask Georges Gonthier this question. He answered that he worked, not so much from [20] itself as from [6, 56], two substantial books which between them described much work simplifying and clarifying the argument, and that such work had yet to be done for the full classification.

1.2 Questions for Consideration

What are the implications for

- authors
- journals and their publishers
- the refereeing process (we note that, although [26] took seven years not to be completely refereed, [28] still took three years to be refereed: "formal" is not the same as "simple".)
- readers
- the storage and curation of such proofs, and, if necessary, the software necessary to run such proofs.?

These questions are not independent: the refereeing process is run by journals, and one can ask whether the journal should keep the machine-readable proof, as with [26], or whether Helfgott is right with "available on request", or maybe Maynard's "at `www.arxiv.org`".

2 Bjorn Poonen

2.1 Kinds of machine assistance

There are various forms of machine-assisted proof, and they need different approaches.

[3] "Both the size of this proof and the range of mathematics involved make formalization a formidable task" [24].

- **Experimental mathematics**: Humans design experiments for the computers to carry out, in order to discover or test new conjectures.
- **Human/machine collaborative proofs**: Humans reduce a proof to a large number of cases, or to a detailed computation, which the computer carries out.
- **Formal proof verification**: Humans supply the steps of a proof in a language such that the computer can verify that each step follows logically from previous steps.
- **Formal proof discovery**: The computer searches for a chain of deduction leading from provided axioms to a theorem.

Cloud computing services make it possible to do computations much larger than most people would be able to do with their own physical computers.

Example 1. *My colleague Andrew Sutherland at MIT did a 300 core-year computation in 8 hours using (a record) 580,000 cores of the Google Compute Engine for about $20,000.* As he says,

> "Having a computer that is 10 times as fast doesn't really change the way you do your research, but having a computer that is 10,000 times as fast does."

2.2 Large databases

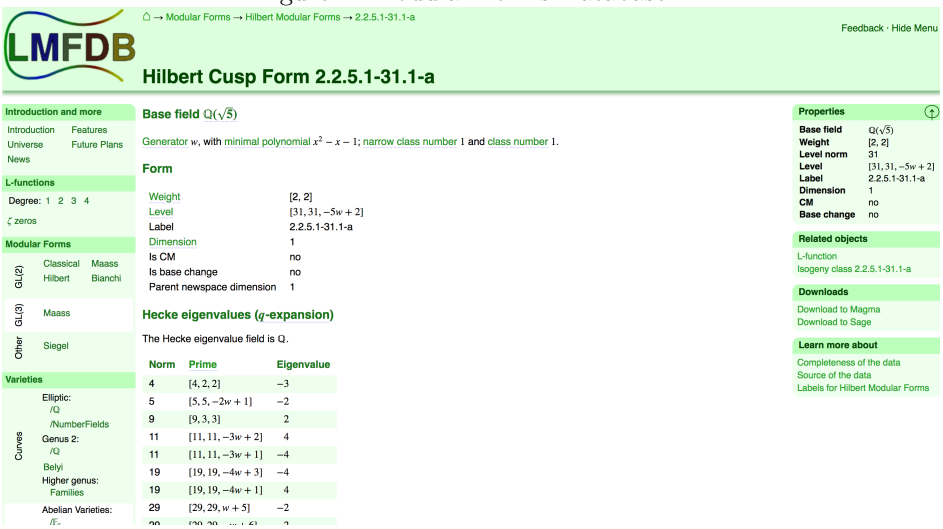
There are now many databases of mathematical facts, such as the Atlas of Finite Simple Groups [12] and the Online Encyclopedia of Integer Sequences [60, 61]. My personal tool is the LMFDB — The L-functions and modular forms database (`www.lmfdb.org`).

2.3 Refereeing computations

Almost all math journals now publish papers depending on machine computations.

- Should papers involving machine-assisted proofs be viewed as more or less suspect than purely human ones?
- What if *the referee is not qualified* to check the computations, e.g., by redoing them?
- What if the computations are *too expensive* to do more than once?
- What if the computation involves *proprietary software* (e.g., MAGMA) for which there is no direct way to check that the algorithms do what they claim to do?
- Should one insist on **open-source software**, or even software that has gone through a **peer-review process** (as in Sage, for instance)?
- In what form should computational proofs be published?

Figure 1: Modular Forms Database



2.4 Some opinions

- The burden should be on authors to provide understandable and verifiable code, just as the burden is on authors to make their human-generated proofs understandable and verifiable.
- Ideally, code should be written in a high-level computer algebra package whose language is close to mathematics, to minimize the computer proficiency demands on a potential reader.

Acknowledgements. Many thanks are due to my colleague Andrew V. Sutherland.

3 James Maynard

3.1 Opportunities and challenges of use of machines

The use of computers in mathematics is widespread and likely to increase.

This presents several **opportunities**:

- Personal assistant: Guiding intuition, checking hypotheses
- Theorem proving: Large computations, checking many cases
- Theorem checking: Formal verification

But equally it presents several **challenges**:

- Are computations rigorous?
- Are proofs with computation understandable to humans?
- How do we find errors?

3.2 My use of computation

I use computation daily to guide my proofs and intuition. Two particular kinds of computations occur in my proofs.

The first is when I want to understand the spectrum of infinite dimensional operator, but of course computers can't handle these as such. Then I consider well-chosen finite-dimensional subspace and do explicit computations there.

- Do non-rigorous computation to guess good answers, then compute answers using exact arithmetic as in [47].
- Happy trade-off between quality of numerical result and computation time.

The second is when, after doing theoretical manipulations, I need to show that some messy explicit integral is less than 1, as in [46].

- My computations are non-rigorous!
- Very difficult to referee — many potential sources for error.

4 Harald Helfgott

4.1 The varieties of machine assistance: Main issues.

When we say that our work on a proof is "machine-assisted", we may mean any of several related things. First, we may simply refer to exploratory work – computations and plots that give us an idea of which statements are true. We may also speak of computations that are truly part of a proof – dealing with case-work, say, or numerical computations, which of course ought to be rigorous. Lastly, we may speak of *formal proofs*: their production and verification could in principle be done by hand, but part of their point is that they can be verified mechanically; moreover, as far as the working mathematician is concerned, they were essentially a useful fiction before computer assistance in their creation became possible.

My present concerns center on the second kind of machine assistance just listed, namely, the use of computations – numerical or symbolic – as part of a proof. The first issue to discuss is what is meant by rigorous computation. (That would be less of an issue for exploratory work, where all one wants is results that are sufficiently precise and almost certainly correct.) Another issue is that of error. Are computer errors at all likely? How do we reduce the possibility of programming errors, or, more generally, errors in machine-human interaction?

For that matter, can computers help us check for errors in proofs altogether, whether the process of proof was computer-assisted or not?

This last question does lead naturally to the matter of formal proofs. That matter gives rise to another sort of concern: what is the meaning and purpose of a proof, for us, as human beings practicing mathematics? I will not do more than touch upon that domain. What can be discussed more factually is the extent to which formal proofs are already a reality, particularly in number theory, and *how* they are a reality; such a discussion has to precede and inform more philosophical considerations.

4.2 Rigorous computation

We will be first of all concerned with the production of numerical results that are guaranteed to be correct. (The same matter goes by the names of "reliable computing", or "validated numerics".) In exploratory work, we may ask, say, for the value of $\sin(0.1)$; however, a moment's thought shows that the true numerical value of $\sin(0.1)$ cannot even be stored in a computer – a decimal or binary expansion has to be cut off at some point. What can we do?

4.2.1 Basics on interval arithmetic

Rigorous computations with real numbers must take into account rounding errors and other sources of lack of precision. As we were saying, a generic element of $\mathbb{R}$ cannot even be represented in a computer. A transcendental number is, in general, given by an infinite sequence of digits, and computers have only finite space. Furthermore, computers are built so that they deal with integers, and, by extension, with rational numbers; they are fastest when working with rationals whose denominators are powers of 2.

Interval arithmetic provides a way to keep track of rounding errors automatically, while providing data types for work in $\mathbb{R}$ and $\mathbb{C}$. The basic data type is an *interval* $[a, b]$, where $a, b \in \mathbb{Q}$. For obvious reasons of efficiency, we may restrict a and b to be elements of the localization Q_2 of $\mathbb{Z}$ by the powers of 2, i.e., the ring Q_2 of rationals whose denominators are powers of 2. (We also may decide to work with $Q_2 \cup \{-\infty, \infty\}$ instead of Q_2.)

A procedure is said to implement a function $f : \mathbb{R}^k \to \mathbb{R}$ if, given $B = ([a_i, b_i])_{1 \leq i \leq k}$, where $a_i, b_i \in Q_2$, the procedure returns an interval (a, b) containing $f(B)$, with $a, b \in Q_2$. Of course, one would expect a good implementation not to return an interval (a, b) much larger than it needs to be.

Interval arithmetic in this sense was first proposed in the 1950s and 1960s ([49]; see also [68], [65], [62], among others). There are several commonly used open-source implementations. (The package ARB [41] implements a slight variant, *ball arithmetic*.) A good interval-arithmetic package should implement not just the four basic operations and some basic functions such as exp, log or sin, but also as many commonly used transcendental functions as possible.

The main drawback of interval arithmetic is its time consumption. Very roughly speaking, a procedure coded using a good interval-arithmetic package

will typically take about 8 times as long as the same procedure coded without interval arithmetic.

There is also the fact that the underlying floating-point routines must give validated results, that is, results with a precise bound on the error term; it is best if results are correct up to the last bit, with whatever rounding type is specified, as all further bits after the last guaranteed bit are of course useless. Correctly rounded floating-point results may feel like a basic right, but the fact is that most processors do not guarantee this arguable right for anything other than the four basic operations, and in fact often violate it. Thus, transcendental functions have to be implemented in software. See, e.g., [16]. (Of course, it may not be fair to count the need for a software implementation as part of the overhead of interval arithmetic; we should not use wrong routines implemented in hardware to begin with.)

The speed of computers in our days makes large-scale computations in interval arithmetic possible; see, for instance, D. Platt's verification of the Riemann Hypothesis for zeroes with imaginary part $\leq 1.1 \cdot 10^{11}$ [57].

Alternatives. It is possible to avoid interval arithmetic (or rather its usage during the computation, with the attendant overhead) by keeping track of rounding errors *a priori*, that is, analyzing carefully to what extent an error of a given size in the input to a given procedure can affect the output. This is one of the classic subjects of numerical analysis [67], [35]. One clear drawback is that proceeding in this fashion for anything outside a small set of well-studied tasks involves saving computer time at the expense of human time, and creates one more occasion for human error.

It would make sense for it to be possible to be able to analyze rounding errors a priori *with the help of a computer*. (The same goes for errors that result from, say, truncating a series.) Systems for doing as much exist, and in fact produce formal proofs (q.v.): FPTaylor (which produces certificates for the formal system HOL Light), Gappa (which gives certificates for the formal system Coq), Real2Float (Coq again)...

However, none of these systems can treat loops, at least not if the floating-point computation carried out in an iteration of the loop depends on the floating-point computations carried out in previous iterations. This limitation is severe: it makes these systems unusable for one the kinds of computations most common in number theory, namely, a computation or verification involving sums or some other quantities defined by an iterative or recursive process. (For an example, see the discussion of the sum $m(x)$ in §4.3.)

On the other hand, systems for analyzing rounding errors can be and have been used to verify that at least some of the routines used by interval-arithmetic packages are correct. Moreover, there are presently tools that, if developed further, should become able to give formal certificates for at least some kinds of computations with loops in a fairly near future.

4.2.2 Comparisons. Maxima and minima

In exploratory work, it may be fine to plot two functions $f, g : I \to \mathbb{R}$, where $I = [0, 1]$ (say) and decide that $f(x) < g(x)$ for all $x \in I$ because the graph shows that it is so. Of course, that would not do as a proof. Fortunately, just as we can let a computer plot f and g, we can let a computer prove that $f(x) < g(x)$ for all $x \in I$.

In fact, it is particularly simple to do so by means of interval arithmetic. Let f and g be implemented in interval arithmetic. If $f(I)$ and $g(I)$ do not overlap, we obtain either that $f(x) < g(x)$ for all $x \in I$ (if $f(I)$ is to the left of $g(I)$) or that $f(x) > g(x)$ for all $x \in I$ (otherwise). If $f(I)$ and $g(I)$ do overlap, we divide I in half, and recur: we do what we have just described to each half. We stop when we obtain that the statement we are trying to prove is false on some subinterval, or when we have divided I into subintervals on each of which the statement is true. While this approach runs into difficulties for x close to a point x_0 such that $f(x_0) = g(x_0)$, we can usually resolve such a situation by comparing the derivatives or higher derivatives of f and g at or near x_0.

A very similar version of the bisection method in interval arithmetic (combined, if necessary, with automatic differentiation) can be used to locate maxima and minima, as well as roots.

4.2.3 Numerical integration (quadrature)

There are several well-known ways to estimate an integral and bound the error in the estimate, starting with, say, the trapezoid rule, or Simpson's rule, and including Euler-Maclaurin or Gaussian quadrature, for instance. It is generally possible to implement these methods in interval arithmetic.

Matters can become more complicated if the integrand is not everywhere differentiable, or even if it not differentiable on the whole *closure* of the interval we are working on. A common case is that of an integrand of the form $|f(x)|$, where f is everywhere differentiable, but $|f|$ is not differentiable at all zeroes of $f(x) = 0$. That case and several other ones ought to be done automatically; they still require ad-hoc work at the time of writing.

Even more ad-hoc work is required (for now) if we must compute a complex integral. For instance, it is possible to show that, for $P_r(s) = \prod_{p \le r}(1 - p^{-s})$ and R the straight path from $200i$ to $40000i$,

$$\frac{1}{\pi i} \int_R |P(s + 1/2)P(s + 1/4)| \frac{|\zeta(s + 1/2)||\zeta(s + 1/4)|}{|s|^2} |ds| = 0.009269 + \text{error},$$

where $|\text{error}| \le 3 \cdot 10^{-6}$. However, "it is possible" means for now "one can obtain this result by e-mailing ARB's author (F. Johansson), editing the code he kindly sends you, and then running it overnight". It is clear that the situation should (and will) improve.

4.3 Combining asymptotics and computations

Let us now discuss the way in which the need for computations typically arises in number theory. (It is not the only way; one can also need, say, verifications of finite chunks of the Riemann or Generalized Riemann Hypotheses of the kind we have already mentioned.) Often, methods from analysis yield estimates of the following form:

$$\textbf{expression}(n) = f(n) + \text{error}, \tag{1}$$

where $|\text{error}| \leq g(n)$ and $g(n)$ is much smaller than $|f(n)|$ for n sufficiently large. The question is what to do when n is not sufficiently large.

One answer is, of course, that one may compute **expression**(n) for n not sufficiently large, that is, n smaller than a constant N. If we reduce our bound $g(n)$, then we are reducing the constant N such that $g(n)$ is sufficiently smaller than $|f(n)|$ for $n \geq N$; reducing N to a reasonable level is thus a challenge for us as analysts. Computing **expression**(n) reasonably efficiently for $n < N$ is a challenge for us as programmers, or, what is almost always more interesting, as algorithm designers. Whether the algorithm for computing **expression**(n) for $n \leq N$ runs in time $O(N)$, $O(N^2)$ or $O(N^3)$ is obviously more important than the extent to which the code has been optimized.

An interesting example is given by the proof of the ternary Goldbach conjecture, taken as a whole. (The first version of the proof is available online, both as a series of preprints and as a book draft [32]; the version to be published is in preparation.) As is well-known, the ternary Goldbach conjecture states that every odd integer $n \geq 7$ can be written as the sum of three prime numbers.

Almost all of the effort involved in the proof went into proving an estimate of the form (1) for **expression**(n) defined to be the number of ways one can write n as the sum of three primes p_1, p_2, p_3 (counted with certain weights). It was then very easy to show that $g(n) < |f(n)|$ for $n \geq 10^{27}$ odd. It followed that every odd number $n \geq 10^{27}$ can be written as the sum of three primes, i.e., ternary Goldbach holds for $n \geq 10^{27}$. It remained to show that every odd number $7 \leq n \leq 10^{27}$ can also be written as the sum of three primes; then the ternary Goldbach conjecture would follow.

Checking each odd number $\leq 10^{27}$ is out of current computational reach. However, there is the following well-known trick. We can easily construct an increasing succession of primes, all of them $< N$ but for the last one, such that the difference between any two consecutive primes in the list is at least 4 and at most $M - 2$, say; the time taken is not much larger than $O(N/M)$. (Platt and I checked that for $N = 8.875 \cdot 10^{30}$ and $M = 4 \cdot 10^{18}$; by now, there is even a formal proof of the same for $N = 10^{28}$ and $M = 4 \cdot 10^{18}$ (see [58]).) Then we can use a verification by brute force that the strong Goldbach conjecture holds for all even numbers $4 \leq n \leq M$, i.e., every even number $4 \leq n \leq M$ can be written as the sum of two primes. It then follows that every odd number $7 \leq n \leq N$ can be written as the sum of three primes: just let p be the largest prime in the sequence such that $n - p \geq 4$, and apply strong Goldbach to $n - p$, which has to be at least M. We obtain that $n - p = p_1 + p_2$. Thus $n = p + p_1 + p_2$, and so we are done. The total time taken is not much larger than $O(M + N/M)$,

so we can simply set $M = \sqrt{N}$, and obtain an algorithm running in time not much more than $O(\sqrt{N})$, which is not too much for $N = 10^{27}$.

As it happens, it had already been checked that strong Goldbach holds for all even $n \le M = 4 \cdot 10^{18} + 2$ ([54], plus a trivial check for $4 \cdot 10^{18} + 2$). Hence, we can simply set $M = 4 \cdot 10^{18} + 2$, and the proof of the ternary Goldbach conjecture is done.

Let us see a different, much smaller example. We would like to bound the sum $m(x) = \sum_{n \le x} \mu(n)/n$ for general $x > 0$. (That sum and its variants play an important role in the proof of ternary Goldbach.) The strongest explicit bound is due to Ramaré [59], who proved that

$$|m(x)| \le \frac{0.0144}{\log x} \tag{2}$$

for $x \ge 96955$. (We would have a bound of $O_\epsilon(1/x^{1/2-\epsilon})$ if we assumed the Riemann hypothesis, but that is out of reach; there are also unconditional non-explicit bounds that are qualitatively much stronger than (2) as $x \to \infty$, but there are serious obstacles to making them explicit with constants reasonable enough for the bounds to be usable.)

Clearly, we can compute $m(x)$ for all small integer values of x. I wrote a C program using interval arithmetic to do so, and obtained, after a couple of weeks on a rather ordinary PC, that

$$|m(x)| \le \sqrt{2/x} \tag{3}$$

for all real $0 < x \le N = 10^{14}$, with 0.569449 instead of $\sqrt{2}$ if $x \ge 3$ is assumed. Note that, if a standard conjecture holds, the bound (3) is actually too strong to hold for all x.

Is it overshooting to test (3) for all $x \le 10^{14}$? Perhaps, but, since the obvious algorithm for checking (3) for all $x \le N$ takes time essentially linear on N, we might as well allow ourselves a large N. Also to the point – it is possible to combine (2) and (3) to prove a result valid for all x, small and large: given any $y > 1$ and any $0 < x \le y$,

$$|m(x)| \le \sqrt{\frac{2}{x}} + 0.0144 \cdot \frac{y^c}{\log y} \frac{1}{x^c},$$

where $c = 1/\log N = 1/\log 10^{14}$. The bound here is a sum of powers of x, something highly convenient for several purposes. The larger N is, the better the bound.

Of course, in this example, it was easy to give a fast algorithm. For some other sums I had to deal with, the obvious algorithm would have run in time $O(N^2)$ or $O(N^3)$, and finding an algorithm that would run in, say, time $O(N \log N)$ and space roughly $O(\sqrt{N})$ was a challenge.

4.4 Error avoidance

In practice, computer errors in the strict sense are barely an issue, except perhaps for very large computations. (By "computer error" we mean a malfunction

in a correctly designed circuit. There is also the issue of incorrectly designed circuits, such as those in the original Pentium chip, whose flaw, as it happens, was found through a computation in number theory ([53], [9]). Such situations are nowadays largely avoided through *formal verification* – another interesting area.) Computers have long had various inbuilt tools (e.g., checksums) for error detection and correction. Moreover, there is the obvious fact that the same computation can be run time and again, possibly on different hardware. Such a thing would be cumbersome or expensive only if the computation were very large indeed. (Admittedly, that is also the case when the probability of computer error might not be overwhelmingly negligible.)

Almost all of the time, the actual issue is human error: errors in programming, and errors in input/output. Of course, human beings also make mistakes when they don't use computers. There are conceptual mistakes, and then there are silly errors, especially in computations or tedious case-work. Presumably, whatever discipline we adopt to avoid errors in programming and input/output can also be adapted to weed out silly mistakes in general.

Before we discuss formal proofs, we should touch upon the more humble matter of everyday discipline.[4] Here I can simply describe my current practice, particularly in the context of my proof of the ternary Goldbach conjecture.

When facing computational tasks, I tend to program a great deal at first. However, I try to minimize the number of programs used in the submitted version of a book or paper, and keep them short. Nowadays I classify tasks in two categories. For time- and space-intensive computations, I am whittling down my code to a few programs in C, submitted or to be submitted to referees. The programs are or will be available upon request.

For small computations, I now use Sage/Python code, almost all of it included in the TeX source of the book. (The code for small computations that take more than a couple of seconds has been submitted together with the programs in C just mentioned.) It is an easy matter to keep Python code brief and readable. Thanks to SageTeX, input and output are automated, in that the output of Sage/Python code is displayed automatically in the file that TeX outputs. For instance, given two variables called `result1` and `result2`, with values 5 and 7, we can type the TeX code

```
Hence,  $f(x)\leq \sage{result1 + result2}$.
```

and obtain

> Hence, $f(x) \leq 12$.

In this way, the possibility of human error in copying input/output and updating material is reduced. All code can be run afresh between two compilations of the TeX file; indeed, I run it afresh (and have to run it afresh) after any change to any code in the file. Someone downloading the source code may decide to run the code or not; it is very easy to do so for anyone with Sage installed. It is

[4] Specialists in formal proofs remind me that "everyday discipline" in the sense I am about to discuss is also an issue for them.

also very easy to display all code; all that is involved is switching a single flag at the beginning of the source code.

There is a question that cannot be avoided, namely, whether it is right to use a large computer-algebra system in a proof. It is in general unsatisfactory to say "By M.,..."[5] before a claim, as some papers do. Consequently, in the first version of my proof of ternary Goldbach, I used Sage only in an exploratory role, and did all coding in C. At the same time, Sage is open-source, peer-reviewed and highly modular (though some of its larger, older components may not themselves be highly modular or peer-reviewed in quite the same way). Because Sage is modular, we depend only on the correctness of the relatively small parts of it that we use in the proof; in my case, that amounts to Python, basic symbolic algebra and ARB. In part for this reason, and in part for the reasons outlined above (code readability, code organization, reduction of human error in input/output), I decided that the advantages of using Sage/Python for small computations that can be embedded in the TeX code overwhelmed any reasonable misgivings one might have about using Sage in a proof in this particular way.

4.5 Automated proofs. Formal proofs

4.5.1 Automated proofs in practice

Informally speaking, Gödel's incompleteness theorem states that, in a general axiomatic system – in particular, in any finite axiomatic system that includes basic arithmetic, including exponentiation – there are truths that cannot be proved starting from the axioms. In particular, we cannot ask a machine to prove all true statements.

Moreover, even in axiomatic systems that *are* complete, the problem of finding a proof may be (and in fact is) computationally hard. Indeed, the first problem to be proven to be NP-complete was that of Boolean satisfiability (SAT); in other words, if we could determine rapidly whether the value of the variables in a Boolean formula can be set so that the formula is true, we could solve rapidly *any* problem in a very broad class (essentially, any problem whose solution can be verified rapidly).

Does that mean that fully automated theorem proving is hopeless? Not necessarily. First, a machine could prove *some* true statements, even if it cannot prove every true statement; after all, human beings are in that same situation. Second, a computer can proceed by heuristics to solve quickly some cases of a problem that is computationally hard in general. Third, a problem may be small enough (in terms of its number of variables, say) that an inefficient algorithm running on a computer can still solve the problem in a reasonable time, even when its solution would be non-obvious or cumbersome to a human.

I have had little experience with automated theorem proving – and in fact was surprised to see that it could be used in practice at all. At some point, I

[5] Here M. stands for Mammoth, or anything else that is closed-source and very large.

wanted to prove the following lemma: for all $0 < x \leq y_1, y_2 < 1$ with $y_1^2 \leq x$, $y_2^2 \leq x$,

$$1 + \frac{y_1 y_2}{(1 - y_1 + x)(1 - y_2 + x)} \leq \frac{(1 - x^3)^2(1 - x^4)}{(1 - y_1 y_2)(1 - y_1 y_2^2)(1 - y_1^2 y_2)}. \qquad (4)$$

Now, this is a statement in the theory of real closed fields, which is in fact complete; a proof has to exist. It turns out that there is a freely available program QEPCAD [36] implementing an algorithm that proves statements in the theory of real closed fields (by *CAD*, that is, cylindrical algebraic decomposition [11]). Can it deal with (4)? Not on my desktop, and that is not surprising: the computational complexity of CAD is doubly exponential in n, with base proportional to the maximal degree of the polynomials involved. However, thinking a little, it is not hard to eliminate a variable in (4), in that one can show that the maximum of the left side minus the right side has to be attained when either $y_1 = y_2$, $y_i = \sqrt{x}$ or $y_i = x$ holds for at least one of $i = 1, 2$. QEPCAD proves the resulting inequality in two variables with ease.

Since (4) has quantifiers of only one kind ($\forall x \forall y_1 \forall y_2$), there are alternative algorithms that solve the problem in time exponential on the number of variables (see [5, Ch. 11] and references therein). However, there does not seem to be a practical, reasonably efficient implementation of the exponential-time algorithm commonly available just yet.

In the end, I found an alternative way to prove what I truly wanted, and so I no longer needed (4) at all. It was still interesting to learn that automatic provers could sometimes establish useful auxiliary lemmas. (Of course, in some sense, the simple combinations of the bisection method and interval arithmetic we have already discussed also fall into this category.) It was also interesting to learn of the extent to which practice still fell behind theory – and of how "computationally hard" does not always mean "hopeless" in practice.

Again, my personal comments should be understood as coming from the possibly naïve perspective of a number theorist. I have recently learned that Boolean-satisfiability (SAT) solvers are being used intensively in combinatorics. There, one of the main issues is error in the human/computer interface: the interface of an automated solver is not necessarily intuitive, and some claimed proofs in the literature are the result of inputting the problem into a SAT solver incorrectly.

At any rate, it seems clear that the role of automated provers in the foreseeable future will be one of *assistance* to the human prover. Thus we are led to our next, broader subject.

4.5.2 Formal proofs and proof assistants

Contrary to what we may sometimes tell ourselves, what we call a proof in our everyday practice is not quite the same as what we would call a proof in a logic course. The latter – called a *formal proof* for clarity – is a sequence of symbols whose correctness is a purely syntactic property that can be checked by a monkey

grinding an organ. In contrast, a proof, for the working mathematician, is a convincing argument that can in principle be turned into a formal proof.

Until relatively recently, "in principle" came with an enormous caveat: none but the simplest sort of proofs was turned into formal proofs. Our imaginary organ and monkey were first replaced by a (real) computer system in 1967 (de Bruijn's AUTOMATH [19]), but the task of producing a formal proof remained solely in the hands of humans, and was very cumbersome.

The situation has gradually changed thanks to *proof assistants*, programs whose role it is to help a human being write a formal proof. There is now a number of such assistants: Mizar, Coq, Isabelle, HOL Light,...

One notable recent success was the second proof of Hales' sphere packing theorem. As is well-known, the first proof ([26]; see also [43]) was computer-assisted in the more traditional ways discussed in previous sections. That first proof met with some misgivings: it involved a great deal of case-work (thus increasing the possibility of human error, in part because the refereeing process became very onerous), and the computer code ran up to about 40000 lines (making the code practically impossible to referee). The second-generation proof [28] is a formal proof, written and verified by means of the Isabelle and HOL Light proof assistants.

In which fields of mathematics is such an effort now feasible? Large areas, including much of basic real analysis, remained uncovered until recently. As far as analytic number theory is concerned: the first formal proof of the Prime Number Theorem [4] was constructed in 2005, by means of Isabelle; it is based on Selberg's elementary proof, which is often seen as more difficult, or less natural, than more traditional proofs by complex analysis. The first analytic proof, based on Newman's simplified version of the traditional approach [52], was given in [31]; it relied on the fact that some of real and complex analysis had become available in HOL Light [29], [30], and also necessitated giving formal proofs of some basic facts about the Riemann zeta function.

In analytic number theory, we are, then, perhaps just past the beginning. The next natural step would be to formalize much of a first-year graduate textbook – say, the results in [17] and [48], possibly with different proofs, together with some sieve theory, and also [63]; then we would need large parts of [40]. (Replace chapters of the older books here by more modern sources when needed.) Then we would be able to start working on newer or new material. Of course, one can also proceed backwards, setting oneself a challenge (several of the major results in [40] would do nicely) and working backwards from it, proving whatever basic results one needs, much as Gonthier et al. did with the Feit-Thompson Theorem [24].

This last strategy is, in a sense, similar to what I did in the end for ternary Goldbach; I had to ask about, learn about, and prove explicit analogues of many basic results in analytic number theory. Sometimes, to my naïve surprise, I had to do without a standard technique or result, since no practical explicit analogue existed or could realistically be proved. We will presumably face the same kind of challenge when we try to give formal proofs of the main results of twentieth-century number theory, including, why not, Vinogradov's three-prime

theorem.

What about formal proofs of explicit results, or of statements whose proofs make crucial use of explicit intermediate results? Why not a formal proof of ternary Goldbach? Why not indeed, in the long run, but we have to be realistic about the fact that both making a proof explicit and making it formal take plenty of work, and of course can also lengthen the proof greatly. It may be that the de Bruijn factor – that is, the informal quantity defined as the ratio of the length of the formal version of the proof to the original, conventional ("informal"?) version, however stored – is lower for some explicit results, as their proofs tend to include a level of detail not always needed for non-explicit results. Time will tell. Of course one can also say that formal proofs are more sorely needed for explicit results; while, in general, they do not elicit the same suspicion as results with plenty of case-work, they may be more likely to be in the "fixable, but incorrect as stated" category than their non-explicit counterparts. (Asymptotic notation is a carpet under which both known and unnoticed dirt can be conveniently swept.) Thus it would be a very worthwhile task – for the fairly near future – to give formal proofs for basic explicit estimates that are used again and again.

4.6 Final considerations

A point often made in connection with computer-assisted proofs is that the purpose of proof is not just to establish truth, but to demonstrate and advance understanding. The worth of a proof understandable to no one would thus be limited.

This viewpoint is valid, but may not really be specially relevant to computer-assisted proofs as they are currently developing. It could certainly be a point against automated provers, particularly if their output is not intelligible. However, in most fields, automated provers seem likely to continue playing at most an auxiliary role, proving small lemmas that would be cumbersome and not particularly enlightening for a human to prove. (Does (4) have any "meaning"?) Computer-assisted formal proofs are a different matter: there, we start with a proof in the everyday sense, that is, a proof that is produced by human beings and understandable to a (hopefully proper) superset of the same; we then formalize it, with the help of a computer, having the more solid establishment of truth as the primary or sole aim.

As for case-work: while it is often locally easy to understand (so to speak), it is true that it can feel meaningless, unaesthetic, and an invitation to error by means of tedium. However, that is an issue with case-work in general, and not with computer-based case-work in particular. Computers can, at least, play a role in reducing or eliminating error from case-work.

Moreover, the kind of finite verification typical of number theory cannot really be said to be case-work in this sense. Instead of having many slightly different cases, we have typically a single equality or inequality that must be verified for very many integers, or for all values of a variable under a certain threshold. Moreover, what we verify is often far from meaningless: if a compu-

tation verifies that the first 10^9 non-trivial zeroes of the Riemann zeta function lie on the critical line, or that $\sum_{n \leq x} \mu(n)$ is bounded by not much more than $\sqrt{x}$ for all $x \leq N$, we are verifying finite parts or finite consequences of standard conjectures that we have very good reasons to believe in – and good reasons to believe out of reach.

The assistance of computers can lead us both to results that were previously unattainable and to a higher standard in certainty and rigor. It may be some time before the standard of rigor set by formal proofs becomes widespread, but we have come to the point where computers, correctly used, can quell misgivings rather than give rise to them.

Acknowledgements. Many thanks are due to Guillaume Melquiond, Assia Mahboubi and Victor Magron for their feedback.

5 Machine-Assisted Proofs in Group Theory and Representation Theory: Pham Huu Tiep[6]

Typically, many proofs in mathematics rely on *mathematical induction*. In group theory and representation theory, this inductive approach often follows a modified strategy, which can be described as follows. Suppose the goal is to prove a certain statement ($\mathcal{P}$) concerning a (finite or algebraic) group G.

(i) Then the first step is to prove a *reduction theorem* to reduce to the case where G is (very close to be) *simple*, using perhaps the *Classification of Finite Simple Groups*. (One should note that, these reduction theorems usually require one to establish a much stronger condition ($\mathcal{P}^*$) for simple groups than the original condition ($\mathcal{P}$). See [37] and [51] for some recent reduction theorems.)

(ii) Next, one works out a uniform proof, which handles the simple groups G of *large enough* order.

(iii) The *induction base* is then to treat all "small" simple groups G.

In either strategy, the induction base usually needs a completely different treatment, rather than the uniform case of large groups, which often involves the, sometimes massive, use of computer calculations.

Let us illustrate this on the example of the proof of the *Ore conjecture* [55]:

Conjecture 2 (Ore, 1951). *Every element g in any finite non-abelian simple group G is a commutator, i.e. can be written as $g = xyx^{-1}y^{-1}$ for some $x, y \in G$.*

[6]The author gratefully acknowledges the support of the NSF (grants DMS-1839351 and DMS-1840702). He also thanks Gabriel Navarro and Eamonn O'Brien for helpful comments on the topic of this discussion.

Many important but partial results on the conjecture were established by Ore himself, Miller, R. C. Thompson, Neubüser-Pahlings-Cleuvers, and most notably, Ellers-Gordeev. The conjecture was finally proved in [44]:

Theorem 3 (**Liebeck-O'Brien-Shalev-T, 2010**). *Conjecture 2 holds for all finite non-abelian simple groups.*

Even building on all previous results, the proof of this "LOST-theorem" is still 70-pages long. So how does this proof go? A detailed account of it was given in the 2013 Bourbaki seminar [45]. A key ingredient of the proof is the following formula, where $\mathrm{Irr}(G)$ denotes the set of all complex irreducible characters of the finite group G:

Lemma 4 (**Frobenius character sum formula**). *Given a finite group G and an element $g \in G$, the number of pairs $(x, y) \in G \times G$ such that $g = xyx^{-1}y^{-1}$ is*

$$|G| \cdot \sum_{\chi \in \mathrm{Irr}(G)} \frac{\chi(g)}{\chi(1)}.$$

So in order to show that a given element $g \in G$ is a commutator, one just needs to show that $\sum_{\chi \in \mathrm{Irr}(G)} \chi(g)/\chi(1) \neq 0$. Now, let G be one of the groups in the induction base for the proof [44] of the Ore conjecture.

(a) In many cases, the character table of G is well known, either published and/or publicly available, in which case we can just use Lemma 4.

(b) In the remaining case, where the character table of G is not available, but if $|G|$ is not too large, then we construct the character table of G and then proceed as before. To construct the character table of such a group G, one starts with a "nice" presentation or representation of G. Then one can try to use various operations with group characters to produce enough characters of G to generate the full group $\mathbb{Z}\mathrm{Irr}(G)$ of complex characters of G. With respect to the usual inner product

$$[\alpha, \beta] = \frac{1}{|G|} \sum_{x \in G} \alpha(x)\overline{\beta(x)},$$

$\mathbb{Z}\mathrm{Irr}(G)$ is a Euclidean lattice, whose minimal vectors are (up to sign) precisely the *irreducible* characters of G that we are after. So one can try to follow, say the *LLL-algorithm*, to find these irreducible characters. In practice (and in [44]), one can use Unger's algorithm [64], implemented in MAGMA [8].

(c) But some of the groups G in the induction base of [44], like $\mathrm{Sp}_{10}(\mathbb{F}_3)$, $\Omega_{11}(\mathbb{F}_3)$, or $\mathrm{U}_6(\mathbb{F}_7)$, are still too big for the computation in (b). For these too-big groups G, we implement another strategy. Namely, for any given $g \in G$, we run a randomized search for $y \in G$ such that y and gy are conjugate. (In fact, one needs to do it only for one representative of each

conjugacy class of G. So, for the largest sporadic simple group, the *Monster*, of order about $8 \cdot 10^{53}$, one needs to work with only 194 such representatives.) Once such a y can be found, then we have $gy = xyx^{-1}$ for some $x \in G$, i.e. $g = [x, y]$, which is what we wanted to show!

The first obvious question that arises is: How *long* was this computation? All in all, it took us about 150 weeks of CPU time of a 2.3GHz computer with 250GB of RAM to complete (by April 2008) all the computations needed for the proof in [44] of the Ore conjecture. (Certainly, this amount of CPU time could be significantly reduced with the better computational algorithms available now, ten years later.)

The second, and more important, question is: How *reliable* is this computation? In the cases of (a) and (b) where we used or computed the character table of G, the relevant character tables have been subjected to various checks which attest to their accuracy. The tables are also publicly available in character table libraries, so they can be checked by others and used for independent verification of the conjecture. Next, in the larger cases of (c), the randomized computation was used to find y that gy and y *should* be G-conjugate for a given g, and then one checks directly (using the given presentation or representation of G) that gy and y are *indeed* G-conjugate. One should notice that this is quite different in nature to other machine-assisted proofs which reduce an elaborate proof to many cases – each is then decided by machine, often reporting "yes" or "no" to the existence of some object.

In group theory and representation theory, as in many other areas of mathematics, perhaps even more important than the machine-assisted proofs are

- *machine-assisted discovered theorems*, and
- *machine-assisted discovered counterexamples.*

Let us mention a couple of examples of these two kinds.

- The *Galois-McKay conjecture* was formulated by Navarro in [50] after many, many days of computing in GAP [25].
- Also after some long experimental computations with the symmetric groups S_n, $n \leq 50$, Isaacs, Navarro, and I found a natural *McKay correspondence* (for the prime 2), which should hold for all symmetric groups, and which was subsequently proved in our joint paper with Olsson [39].
- An old conjecture in Character Theory states that if a finite group G is rational (that is, all $\chi \in \mathrm{Irr}(G)$ are rational-valued), then so are the Sylow 2-subgroups of G. However, after some long (but directed!) search using the "SmallGroups" databases contained in the computer packages [25] and [8], Isaacs and Navarro have been able to find two counterexamples of order $2^9 \cdot 3$ to this conjecture, see [38].

6 Luís Cruz-Filipe

The awareness of the mathematical community towards the use of computers in proofs of mathematical results started in 1976 with the announcement of the proof of the Four-Color Theorem [2].

However, there are earlier examples of proofs that were partially or completely done by a machine. An interesting example is Floyd and Knuth's proof of optimality of the 16-comparator sorting network on 7 inputs (Theorem 5 in [21]), which starts:

> This theorem was proved by exhaustive enumeration on a CDC G-21 computer at Carnegie Institute of Technology in 1966.

Optimality of sorting networks is an interesting problem in combinatorics. The question we want to answer is: what is the length $S(n)$ of the shortest sequence of compare-and-swap operations (i.e., atomic operations consisting of sorting a pair of values) that will sort all inputs of a given length n? Floyd and Knuth's work cited above addresses this problem for $2 \leq n \leq 7$. Except for the values of $S(4)$ and $S(6)$, which are derived from those of $S(3)$ and $S(5)$ by application of a more general theorem, all values are derived by exhaustively enumerating the possible sequences of length $S(n)-1$ and showing that, for each of them, there is an input of length n that they do not sort. As the authors note in their conclusion, this method is however "quite unsatisfactory for higher values of n": the number of cases that need to be analyzed for establishing $S(7)$ is unmanageable for a human being, and lay at the limits of what could be computed in 1966.

Two styles of proving. Floyd and Knuth's proof, as well as the proof of the Four-Color Theorem, are examples of one family of machine proofs: they establish a property by means of an *ad-hoc* computer program that is written for that specific purpose. Nowadays, this approach is not very common: verifying such proofs also demands that the program be verified, a task that does not fall under the usual scope of the peer-reviewing process. Instead, it is generally considered preferable to use a *theorem prover*: a general-purpose program that can construct and/or verify proofs in a particular logic.

Trusting a proof produced with the help of a theorem prover also has some implications. First, one still needs to trust a computer program – the theorem prover itself. The difference with respect to using *ad-hoc* programs is that we are now considering a program that has been subject to much wider scrutiny, and it can be reasonably argued that the ensuing proof is as trustworthy as a published mathematical proof that has been subject to peer-reviewing. The other point that needs to be checked is that the encoding of the actual mathematical problem in the theorem prover's logic is correct: this is generally accepted as part of the peer-reviewing process, as these encodings are typically included and discussed in submitted articles. (This aspect is also an issue when using *ad-hoc* computer programs in proofs, but in that scenario the encodings used tend to be much more direct.)

Theorem provers come in a wide variety of styles and flavors, ranging from very general purpose to more specifically tailored for a particular family of problems. They use different logics, and are often uncomparable in their usefulness. A very encompassing overview of the world of theorem provers can be found in [66].

Machine-assisted proofs today. Forty years after the announced proof of the Four-Color Theorem, machine-assisted proofs are everywhere. Arguably, their widest area of application is hardware and software verification, of which Floyd and Knuth's problem is an example. In an area where we are increasingly dependent on computers to perform critical tasks – from controlling air traffic to administering medicine to patients – it is more and more important that there be no errors in the execution of those tasks, whether due to programming errors or to hardware flaws.

As regards hardware, many useful properties can be verified in a fully automated way [42]. As such, formal verification is becoming more commonplace in industry. Software verification tends to be more complex, but the number of success stories is increasing. In recent years there have been large projects addressing formalization of large fragments of widely-used programming languages, making it possible for non-experts to experiment with proving properties of the programs they write.

Much more challenging is the formalization of mathematical proofs in a computer. These proofs tend to be much less mechanic and systematic, requiring constant interaction between the computer and an expert – where the expert "guides" the computer through the main steps of the proof. Mathematics also tends to build upon itself, and researchers often find that the biggest challenge in formally verifying an "interesting" result lies on the unavailability of libraries formalizing the relevant underlying theories. Nevertheless, recent formalizations of e.g. the proof of the Four-Color Theorem [22] and the proof of Kepler's conjecture [27] show that this goal is not hopeless. (A recent contribution to this list is a formal proof of Floyd and Knuth's results on sorting networks [14], along with the recently established value of $S(9)$ [10].)

A different approach to machine-verified proofs of mathematical results, which some may argue is less elegant, capitalizes on the sucess of SAT solvers. A SAT solver is a program that tackles the Boolean satisfiability problem: given a propositional formula, decide whether it has a satisfying assignment. Although this problem is NP-complete, years of intensive investiment in researching efficient techniques for solving it has resulted in extremely powerful tools that can solve formulas with millions of variables and billions of clauses. To test the limits of these programs, several researchers have experimented with encoding open mathematical problems (typically from combinatorics) as propositional formulas, which were then successfully proven to be unsatisfiable [34].

Until recently, SAT solvers were viewed by some with skepticism: their complexity made them impossible to analyze in practice, and in the case a formula was claimed to be unsatisfiable no independently verifiable guarantee of this fact

was provided. The situation changed recently, and the majority of today's SAT solvers also output traces that allow independent, formally verified systems to check proofs of unsatisfiability of propositional formulas [15, 13].

7 Questions

Q Isn't there a problem with proprietary systems (e.g. Magma, Maple or Mathematics)? You might not be able to find referees who have them.

A There is a problem here, but it is more about rarity than about the proprietary nature, and in fact you are more likely to find a referee who knows (and has) one of these than you are some rare open-source package. The real problem is the proprietary nature of the algorithms. "Yes, this large piece of code, which may be proprietary or open-source, gives me this result, but do I trust it?" For computer algebra, the question is discussed in [18].

On the other hand, it is also the case that Sage has made installing open-source mathematics programs easier, in so far as it seems to incorporate most of them.

Q What about a system that produces verified code, which is then compiled and run?

A That's a good question, and there is some of this in Flyspeck [28]. See also Paulson's MetiTarski project [1]. Of course, one would need a verified compiler, but such things exist these days.

Q Is there much consistency between journals on how these proofs are treated?

A There isn't even much consistency within a given journal. [26] and [47] were both in *Annals of Mathematics*, yet seem to have been treated differently. There is no caution on [47], and it was published much more rapidly than [26]. Conversely the computer programs underpinning [26] are on the *Annals of Mathematics* website, whereas [47] simply says "An ancillary Mathematica(R) file detailing these computations is available alongside this paper at `www.arxiv.org`". The computations in [47] were relatively simple, and carried out exactly: what would *Annals of Mathematics* have made of [46]?

References

[1] B. Akbarpour and L.C. Paulson. MetiTarski: An Automatic Theorem Prover for Real-Valued Special Functions. *J. Automated Reasoning*, 44:175–205, 2010.

[2] K.I. Appel and W. Haken. Every Planar Map is Four-Colorable. *Bull. A.M.S.*, 82:711–712, 1976.

[3] K.I. Appel and W. Haken. Every Planar Map is Four-Colorable (With the collaboration of J. Koch). *Contemporary Mathematics*, 98, 1989.

[4] Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A formally verified proof of the prime number theorem. *ACM Transactions on Computational Logic (TOCL)*, 9(1):2, 2007.

[5] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*. Algorithms and computation in mathematics. Springer, Berlin, Heidelberg, New York, 2006.

[6] Helmut Bender, George Glauberman, and Walter Carlip. *Local analysis for the odd order theorem*, volume 188 of *LMS Lecture Notes*. Cambridge University Press, 1994.

[7] B.J. Birch and H.P.F. Swinnerton-Dyer. Notes on Elliptic Curves I. *J. für reine und angew. Math.*, 212:7–23, 1963.

[8] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system I: The user language. *J. Symbolic Comput.*, 24:235–265, 1997.

[9] Barry Cipra. How number theory got the best of the Pentium chip. *Science*, 267(5195):175, 1995.

[10] M. Codish, L. Cruz-Filipe, M. Frank, and P. Schneider-Kamp. Sorting nine inputs requires twenty-five comparisons. *J. Comput. Syst. Sci.*, 82(3):551–563, 2016.

[11] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata theory and formal languages (Second GI Conf., Kaiserslautern, 1975)*, pages 134–183. Lecture Notes in Comput. Sci., Vol. 33. Springer, Berlin, 1975.

[12] J.H. Conway, R.T. Curtis, S.P. Norton, R.A. Parker, and R.A. Wilson. *Atlas of finite simple groups*. Clarendon, Oxford, 1985.

[13] L. Cruz-Filipe, M.J.H. Heule, W.A. Hunt Jr., M. Kaufmann, and P. Schneider-Kamp. Efficient certified RAT verification. In L. de Moura, editor, *Proceedings of CADE 26*, volume 10395 of *LNCS*, pages 220–236. Springer, 2017.

[14] L. Cruz-Filipe, K.S. Larsen, and P. Schneider-Kamp. Formally proving size optimality of sorting networks. *J. Automated Reasoning*, 59(4):425–454, 2017.

[15] L. Cruz-Filipe, J. Marques-Silva, and P. Schneider-Kamp. Efficient certified resolution proof checking. In A. Legay and T. Margaria, editors, *Proceedings of TACAS (I), held as part of ETAPS*, volume 10205 of *LNCS*, pages 118–135, 2017.

[16] Catherine Daramy-Loirat, David Defour, Florent de Dinechin, Matthieu Gallet, Nicolas Gast, Christoph Lauter, and Jean-Michel Muller. CR-LIBM A library of correctly rounded elementary functions in double-precision. Research report, LIP, December 2006. https://hal-ens-lyon.archives-ouvertes.fr/ensl-01529804.

[17] Harold Davenport. *Multiplicative number theory*. Markham Publishing Co., Chicago, Ill., 1967. Lectures given at the University of Michigan, Winter Term.

[18] J.H. Davenport. Computer Algebra and Formal Proof. `http://staff.bath.ac.uk/masjhd/Slides/JHD2017IsaacNewton.pdf`, 2017.

[19] Nicolaas Govert De Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In *Symposium on automatic demonstration*, pages 29–61. Springer, 1970.

[20] W. Feit and J.G. Thompson. Solvability of Groups of Odd Order. *Pacific J. Math.*, 13:775–1029, 1963.

[21] R.W. Floyd and D.E. Knuth. The Bose-Nelson sorting problem. In J.N. Srivastava, editor, *A Survey of Combinatorial Theory*, pages 163–172. North-Holland, 1973.

[22] G. Gonthier. Formal Proof — The Four-Color Theorem. *Notices A.M.S.*, 55:1382–1393, 2008.

[23] G. Gonthier and L. Théry. Formal Proof — The Feit–Thompson Theorem. `http://www.msr-inria.inria.fr/events-news/feit-thompson-proved-in-coq`, 2012.

[24] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 163–179, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[25] The GAP group. GAP - *groups, algorithms, and programming*, Version 4.4. `http://www.gap-system.org`, 2004.

[26] T.C. Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162:1065–1185, 2005.

[27] T.C. Hales. Flyspeck project completion. *E-mail Sunday August 10th*, 2014.

[28] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, Thi Hoai An Ta, Nam Trung Tran, Thi Diep Trieu, Josef Urban, Ky Vu, and Roland Zumkeller. A Formal Proof of the Kepler Conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017.

[29] John Harrison. A HOL theory of Euclidean space. In *International Conference on Theorem Proving in Higher Order Logics*, pages 114–129. Springer, 2005.

[30] John Harrison. Formalizing basic complex analysis. *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, 10(23):151–165, 2007.

[31] John Harrison. Formalizing an analytic proof of the Prime Number Theorem (dedicated to Mike Gordon on the occasion of his 60th birthday). *Journal of Automated Reasoning*, 43:243–261, 2009.

[32] Harald Andrés Helfgott. The ternary Goldbach problem. Preprint. Available as `https://arxiv.org/abs/1501.05438`.

[33] Harald Andrés Helfgott. Major arcs for Goldbach's theorem. `http://arxiv.org/abs/1305.2897`, 2013.

[34] M.J.H. Heule and O. Kullmann. The science of brute force. *Commun. ACM*, 60(8):70–79, 2017.

[35] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2002.

[36] Hoon Hong and Christopher W. Brown. QEPCAD B – Quantifier elimination by partial cylindrical algebraic decomposition, May 2011. version 1.62.

[37] I. M. Isaacs, G. Malle, and G. Navarro. A reduction theorem for the McKay conjecture. *Invent. Math.*, 170:33–101, 2007.

[38] I. M. Isaacs and G. Navarro. Sylow 2-subgroups of rational solvable groups. *Math. Z.*, 272:937–945, 2012.

[39] I. M. Isaacs, G. Navarro, J. B. Olsson, and Pham Huu Tiep. Character restrictions and multiplicities in symmetric groups. *J. Algebra*, 478:271–282, 2017.

[40] Henryk Iwaniec and Emmanuel Kowalski. *Analytic number theory*, volume 53 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, RI, 2004.

[41] Fredrik Johansson. Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Transactions on Computers*, 66:1281–1292, 2017.

[42] T. Kropf. *Introduction to Formal Hardware Verification*. Springer, 1999.

[43] Jeffrey C. Lagarias. *The Kepler Conjecture: The Hales-Ferguson Proof*. Springer Science & Business Media, 2011.

[44] M. W. Liebeck, E. O'Brien, A. Shalev, and Pham Huu Tiep. The Ore conjecture. *J. Eur. Math. Soc.*, 12:939–1008, 2010.

[45] G. Malle. The proof of Ore's conjecture [after Ellers-Gordeev and Liebeck-O'Brien-Shalev-Tiep]. Séminaire Bourbaki, March 23, 2013.

[46] J.A. Maynard. 3-tuples have at most 7 prime factors infinitely often. *Mathematical Proceedings of the Cambridge Philosophical Society*, 155:443–457, 2013.

[47] J.A. Maynard. Small gaps between primes. *Annals of Mathematics*, 181:383–413, 2015.

[48] Hugh L. Montgomery and Robert C. Vaughan. *Multiplicative number theory. I. Classical theory*, volume 97 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 2007.

[49] Ramon Edgar Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1966.

[50] G. Navarro. The McKay conjecture and Galois automorphisms. *Annals of Mathematics*, 160:1129–1140, 2004.

[51] G. Navarro and Pham Huu Tiep. A reduction theorem for the Alperin weight conjecture. *Invent. Math.*, 184:529–565, 2011.

[52] Donald J Newman. Simple analytic proof of the prime number theorem. *The American Mathematical Monthly*, 87(9):693–696, 1980.

[53] Thomas R. Nicely. Pentium FDIV flaw FAQ. `http://www.trnicely.net/pentbug/pentbug.html`. Accessed: 2018-08-30.

[54] Tomàs Oliveira e Silva, Siegfried Herzog, and Silvio Pardi. Empirical verification of the even Goldbach conjecture and computation of prime gaps up to $4 \cdot 10^{18}$. *Math. Comput.*, 83(288):2033–2060, 2014.

[55] O. Ore. Some remarks on commutators. *Proc. Amer. Math. Soc.*, 2:307–314, 1951.

[56] Thomas Peterfalvi. *Character theory for the odd order theorem*, volume 272 of *LMS Lecture Notes*. Cambridge University Press, 2000.

[57] David Platt. *Computing degree* 1 *L-functions rigorously*. PhD thesis, Bristol University, 2011.

[58] Project Team MARELLE. Mathematical Reasoning and Software: activity report 2014. Technical report, Sophia Antipolis - Méditerranée, 2014. Theme: Proofs and Verification.

[59] Olivier Ramaré. Explicit estimates on several summatory functions involving the moebius function. *Mathematics of Computation*, 84(293):1359–1387, 2015.

[60] N.J.A. Sloane. The Online Encyclopedia of Integer Sequences. *Notices A.M.S.*, 50:912–915, 2003.

[61] N.J.A. Sloane. The Online Encyclopedia of Integer Sequences. `http://www.research.att.com/~njas/sequences`, 2007.

[62] Teruo Sunaga. Theory of an interval algebra and its application to numerical analysis [Reprint of Res. Assoc. Appl. Geom. Mem. 2 (1958), 29–46]. *Japan J. Indust. Appl. Math.*, 26(2-3):125–143, 2009.

[63] E.C. Titchmarsh. *The theory of the Riemann zeta-function.* 2nd ed., rev. by D. R. Heath-Brown. Oxford Science Publications. Oxford: Clarendon Press, 1986.

[64] W.R. Unger. Computing the character table of a finite group. *J. Symbolic Comput.*, 41:847–862, 2006.

[65] Mieczyslaw Warmus. Calculus of approximations. *Bulletin de lAcademie Polonaise de Sciences*, 4(5):253–257, 1956.

[66] Freek Wiedijk, editor. *The Seventeen Provers of the World*, volume 3600 of *LNCS*. Springer, 2006.

[67] James Hardy Wilkinson. *Rounding errors in algebraic processes.* H.M.S.O London, 1963.

[68] Rosalind Cecily Young. The algebra of many-valued quantities. *Mathematische Annalen*, 104:260–290, 1931.