Formal methods in mathematics

Jeremy Avigad

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University

May 2015

Sequence of lectures

1. Formal methods in mathematics
2. Automated theorem proving
3. Interactive theorem proving
4. Formal methods in analysis

Formal methods in mathematics

"Formal methods" = logic-based methods in CS, in:

- automated reasoning
- hardware and software verification
- artificial intelligence
- databases

Goal: explore current and potential uses of formal methods in mathematics.

Computers in mathematics

Numeric and symbolic computation are ubiquitous in applied mathematics:

- engineering and control theory
- numeric simulation, weather prediction
- optimization and operations research
- statistics
- stochastic methods, monte carlo simulation, financial mathematics
- cryptography
- algorithmic game theory
- machine learning

I will not discuss these.

Computers in mathematics

Borwein and Bailey: "experimental mathematics" uses computation for:

1. Gaining insight and intuition.
2. Discovering new patterns and relationships.
3. Using graphical displays to suggest underlying mathematical principles.
4. Testing and especially falsifying conjectures.
5. Exploring a possible result to see if it is worth a formal proof.
6. Suggesting approaches for formal proof.
7. Replacing lengthy hand derivations with computer-based derivations.
8. Confirming analytically derived results.

I will focus more narrowly on theorem proving.

Computers in mathematics

There are many general and specialized packages for algebraic computation:

- Mathematica
- Maple
- Sage
- MATLAB
- Magma
- Gap (discrete algebra, group theory)
- PARI/GP (number theory)
- Kenzo (algebraic topology)
- SnapPea (3-manifolds)
- ...

I will not focus on these.

Computers in mathematics

Formal methods are based on logic and formal languages:

- syntax: terms, formulas, connectives, quantifiers, proofs
- semantics: truth, validity, satisfiability, reference

These can be used for:

- finding things (SAT solvers, constraint solvers, database query languages)
- proving things (automated theorem proving, model checking)
- verifying correctness (interactive theorem proving)

Preview: in computer science, there are impressive uses of formal methods in all these ways.

In mathematics, the first two are lagging.

Contents

"Conventional" computer-assisted proof:

- carrying out long, difficult, computations
- proof by exhaustion

Formal methods for discovery:

- finding mathematical objects
- finding proofs

Formal methods for verification:

- verifying ordinary mathematical proofs
- verifying computations.

The Four color theorem

Theorem

Four colors suffice to color a planar map.

Proved in 1976 by Appel and Haken.

The proof used extensive hand analysis to reduce the problem to showing that each element of a set of 1,936 maps could not be part of a counterexample.

They then used extensive computation to establish the claim in each case.

In 1997, Robertson, Sanders, Seymour, and Thomas simplified the proof (though it still relies on a lot of computation).

Hales' theorem (aka the Kepler conjecture)

Theorem

The optimal density of sphere packing in space is attained by the face-centered cubic lattice.

The proof involves about 300 pages of mathematical text.

Three essential uses of computation:

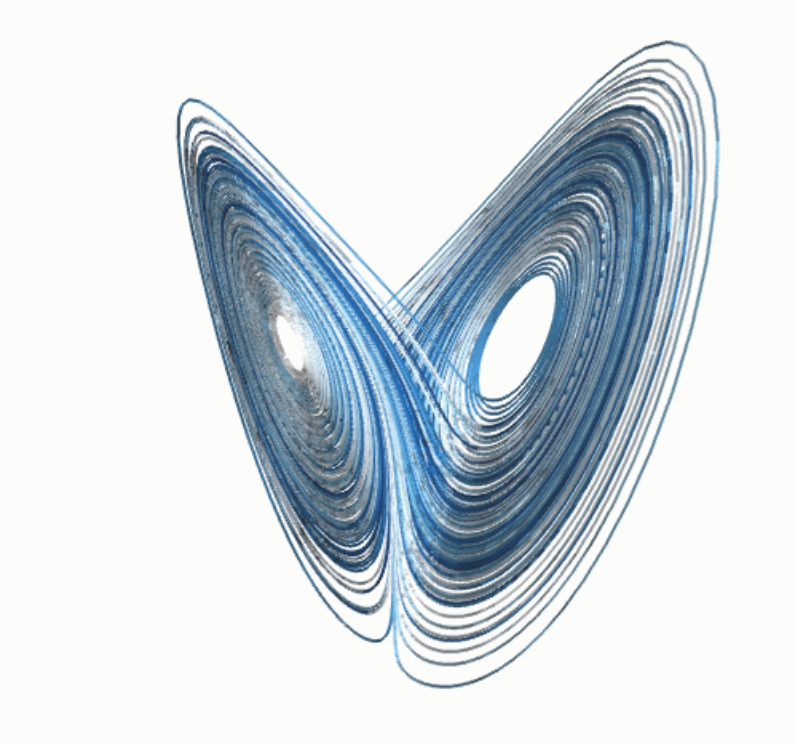
- enumerating tame hypermaps
- proving nonlinear inequalities
- showing infeasibility of linear programs

The Lorenz attractor

In 1963, Edward Lorenz studied the following system of differential equations:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z \end{aligned}$$

with $\sigma = 10, \beta = 8/3, \rho = 28$.



Smale's 14th problem: show that the Lorenz attractor is indeed a *strange attractor*.

The Lorenz attractor

This was shown by Warwick Tucker in 2002.

Ideas:

- Cover the attractor with little boxes, show each is mapped into other boxes.
- Near the origin (an equilibrium), use "normal form theory" and a symbolic approximation to the behavior.
- Away from the origin, use validated numerical methods, i.e. interval arithmetic.
- Show that the flow is "expanding" in a suitable sense.

Higher-dimensional sphere packing

One can also consider sphere packings in higher dimensions, and ask for

- upper bounds
- lower bounds

on

- lattice packings
- arbitrary packings.

In 2003, Henry Cohn and Noam Elkies published an article in the *Annals of Mathematics* which gives the best known upper bounds on arbitrary sphere packings in dimensions 4 through 36.

Higher-dimensional sphere packing

Theorem (Theorem 3.2)

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ *is an admissible function satisfying the following three conditions:*

1. $f(0) = \hat{f}(0) > 0$.
2. $f(x) \leq 0$ *for* $|x| \geq r$, *and*
3. $\hat{f}(t) \geq 0$ *for all* t.

Then the center density of sphere packings in $\mathbb{R}^n$ *is bounded above by* $(r/2)^n$.

The proof uses Poisson summation, and is not hard.

Higher-dimensional sphere packing

The dialectic:

- All we need to do is find such f's.
- Heuristically, such f's should have certain properties.
- We can find good candidates for such f's using linear combinations of Laguerre polynomials.
- We used a simple hill-climbing algorithm to find good f's.
- There was no guarantee that the f's we would find this way would meet the constraints.
- But in practice, they did.
- In any event, the computations were fully rigorous.

Higher-dimensional sphere packing

We can find excellent functions for use in Proposition 7.1 as follows. Let $L_k^\alpha(x)$ be the Laguerre polynomials orthogonal with respect to the measure $e^{-x}x^\alpha\,dx$ on $[0,\infty)$. Set $\alpha = n/2 - 1$, and define

$$g_k(x) = L_k^\alpha(2\pi|x|^2)e^{-\pi|x|^2}$$

for $k \geq 0$; we suppress the dependence on n in our notation. These functions form a basis for the radial eigenfunctions of the Fourier transform, with eigenvalues $(-1)^k$ (see Section 4.23 and equation (4.20.3) of [Leb]).

To find a function g for use in Proposition 7.1, we consider a linear combination of $g_1, g_3, \ldots, g_{4m+3}$, and require it to have a root at 0 and m double roots at $z_1, \ldots, z_m$. (Counting degrees of freedom suggests that there should be a unique such function, up to scaling.) We then choose the locations of $z_1, \ldots, z_m$ to minimize the value r of the last sign change of g. To make this choice, we do a computer search. Specifically, we make an initial guess for the locations of $z_1, \ldots, z_m$, and then see whether we can perturb them to decrease r. We repeat the perturbations until reaching a local optimum. Strictly speaking, we cannot prove that it ever converges, or comes anywhere near the global optimum. However, it works well in practice. At any rate this cannot affect the validity of our bounds, only their optimality given m. As we increase m, this method should give better and better bounds, which should converge to the best bounds obtainable using Proposition 7.1.

This method gives good functions for use in Proposition 7.1, but naturally bounds on all sphere packings would be better than bounds only on isodual lattices. We can turn these functions into functions satisfying the hypotheses of Theorem 3.2, without changing r, as follows.

Let h be a linear combination of $g_0, g_2, \ldots, g_{4m+2}$ with double zeros at $z_1, \ldots, z_m$, such that $g + h$ has a double zero at r. Then in the examples we have computed, $g + h$ has constant sign, which we can take to be positive. We

Higher-dimensional sphere packing

end up with a function $f = -g + h$ that is nonpositive outside radius r, and whose Fourier transform $\widehat{f} = g + h$ is nonnegative everywhere; furthermore, $f(0) = \widehat{f}(0)$. Thus, we can apply Theorem 3.2 to get the same bound for general sphere packings that g proves for isodual lattices. Notice that it is *not* clear *a priori* that f must satisfy all the hypotheses of Theorem 3.2, but it happens in all of our numerical examples.

Figure 1 (from §1) and Table 3 (from Appendix C) illustrate the bounds this method produces, using $m = 6$ as the number of forced double zeros (except in dimension 7 and lower, where $m = 5$ suffices for the accuracy we desire). We also list for comparison Rogers' bound [Ro], which was previously the best bound known in dimensions 4 through 36. The choices of forced double roots are described in Table 4 (from Appendix C).

These bounds were calculated using a computer. However, the mathematics behind the calculations is rigorous. In particular, we use exact rational arithmetic, and apply Sturm's theorem to count real roots and make sure we do not miss any sign changes. More precisely, we take the quantities $2\pi z_i^2$ to be rational numbers. That is convenient, because the functions $g_k(x)$ are polynomials in $2\pi|x|^2$ with rational coefficients (times Gaussians, which have no sign changes). The one subtlety is that in constructing h, we want $g + h$ to have a double root at r, and $2\pi r^2$ is generally not rational. Instead of doing this computation using floating point arithmetic, we replace $2\pi r^2$ by a nearby rational number and then determine h exactly, using that value of r. To do so, we must increase r slightly, but of course we can make the increase as small as we wish.

Higher-dimensional sphere packing

In ordinary mathematics, we use all kinds of heuristics when search for a proof, or a solution to a problem.

Differences:

- We usually don't use computer search.
- We usually don't explain the heuristics.

Computation in Algebraic Topology

In algebraic topology, one assigns algebraic invariants to spaces:

- Homotopy groups
- Homology groups
- Cohomology groups

The goal is then to "calculate" these algebraic objects, for particular spaces (like higher-dimensional spheres).

Often algorithms exist "in principle." For example, Brown 1956 showed given a simply connected polyhedron X, and can compute $\pi_n X$ for $n \geq 2$, but:

> *It must be emphasized that although the procedures developed for solving these problems are finite, they are much too complicated to be considered practical.*

Computation in Algebraic Topology

A program called *Kenzo*, developed by Francis Sergeraert and others, carries out such computations.

It is based on *constructive algebraic topology*, a way of representing homotopy types effectively, due to Sergeraert and Rubio.

A paper in *Algebraic and Geometric Topology* stated a theorem that $\pi_4(\Sigma K(A_4, 1)) = \mathbb{Z}/4$.

Kenzo computed $\pi_4(\Sigma K(A_4, 1)) = \mathbb{Z}/12$.

Kenzo was right.

Other examples

- Lyons-Sims sporadic simple group of order $2^8 \cdot 3^7 \cdot 5^6 \cdot 7 \cdot 11 \cdot 31 \cdot 37 \cdot 67$ (1973)
- Connect Four (first player has a winning strategy; Allis and Allen, independently, 1982)
- Feigenbaum's universality conjecture (Lanford, 1982)
- Nonexistence of a finite projective plane of order 10
- Catalan's conjecture (Mihăilescu, 2002)
- Optimal solutions to Rubik's cube (20 moves, Rokicki, Kociemba, Davidson, and Dethridge, 2010)
- Universal quadratic forms and the 290 theorem (Bhargava and Hanke, 2011)

Other examples

- Weak Goldbach conjecture (Helfgott, 2013)
- Maximal number of exceptional Dehn surgeries (Lackenby and Meyerhoff, 2013)
- Better bounds on gaps in the primes (Polymath 8a, 2014)

See Hales, "Mathematics in the Age of the Turing Machine," and Wikipedia, "Computer assisted proof."

See also computer assisted proofs in special functions (Wilf, Zeilberger, Borwein, Paule, Moll, Stan, Salvy, . . .).

See also anything published by Doron Zeilberger and Shalosh B. Ekhad.

Contents

"Conventional" computer-assisted proof:

- carrying out long, difficult, computations
- proof by exhaustion

Formal methods for discovery:

- finding mathematical objects
- finding proofs

Formal methods for verification:

- verifying ordinary mathematical proofs
- verifying computations.

Formal methods for discovery

Formal methods have *not* had a big impact on ordinary mathematics.

I expect that will change.

SAT solvers

Most modern SAT solvers are based on the DPLL algorithm.

Modern SAT solvers can handle tens of thousands of variables and millions of clauses.

SAT solvers are used for

- planning and scheduling problems
- bounded model checking
- fuzzing and test generation

The Erdős discrepancy problem

Consider $(x_i)_{i \in \mathbb{N}}$ with each $x_i = \pm 1$.

Consider partial sums,

$$x_1 + x_2 + x_3 + x_4 + \dots$$
$$x_2 + x_4 + x_6 + x_8 + \dots$$
$$x_3 + x_6 + x_9 + x_{12} + \dots$$

Given C, are there necessarily $k, d > 0$ such that $|\sum_{i=1}^{k} x_{id}| > C$?

Probably, but it is an open question.

The Erdős discrepancy problem

If the conjecture holds, then for every C, there is an n such that for every $x_1, \ldots, x_n$ there are $k, d > 0$ as above (with $kd < n$).

The *discrepancy* of $x_1, \ldots, x_n$ is the supremum of $|\sum_{i=1}^{k} x_{id}|$ for all such k, d.

Challenge: find long sequences with small discrepancy.

This problem was considered by the Polymath1 project.

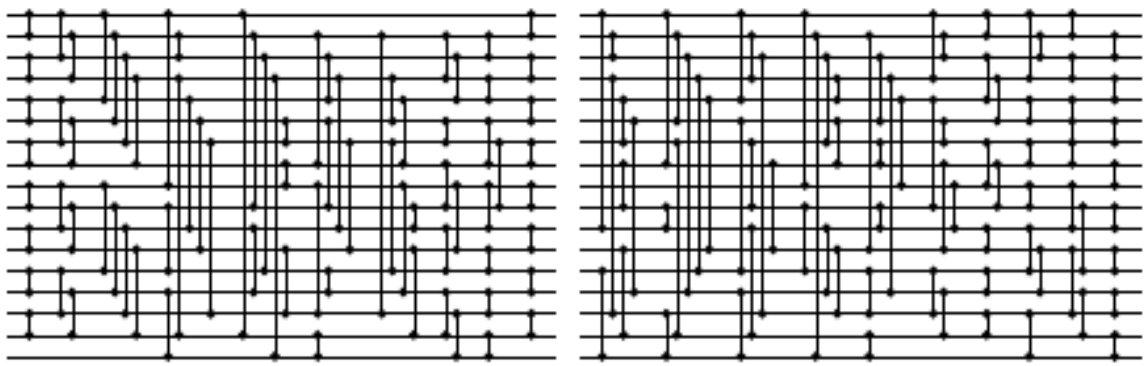
In 2014, Konev and Lisitsa used a SAT solver to find a sequence of length 1160 with discrepancy 2, and show that no such sequence exists with length 1161.

Optimal sorting networks

A *sorting network* is made up of comparator gates.

It is a theorem that it suffices to show that the network works for $0, 1$ inputs.

Here are two sorting networks for 17 inputs:



Optimal sorting networks

For a given number of inputs, one can ask about the minimum number of comparators needed, or the minimum depth.

In 2015, Thorstein Ehlers and Mike Müller, students in Kiel, used SAT solvers (combined with some combinatorial cleverness), to give improved upper and lower bounds on depth for 17–20 inputs.

Zinovik, Kroening, and Chebiryak, 2008, similarly use SAT solvers to find Gray codes.

SMT solvers

SMT stands for *satisfiability modulo theories*.

SMT solvers extend SAT solvers by allowing not only propositional variables, but also atomic formulas from combinations of decidable theories, like

- linear arithmetic
- integer arithmetic
- uninterpreted functions.

Think of them as SAT solvers with arithmetic and "other stuff."

SMT solvers

Like SAT solvers, they are used for hardware and software verification, and are much more expressive.

They are also used for

- the analysis of biological systems
- synthesis of programs
- whitebox fuzzing

But I do not yet know of any striking applications to pure mathematics.

Quantifier elimination

SMT solvers:

- find solutions to quantifier-free / existential formulas.
- prove validity of quantifier-free / universal formulas.

Logic tells us we can do better.

There is full quantifier elimination for

- real linear arithmetic
- integer linear arithmetic (Presburger arithmetic)
- the reals as an ordered field
- the complex numbers

Quantifier elimination

Another way to think about it: linear, integer linear, and nonlinear programming and optimization deal with sets of constraints.

SMT technology adds propositional logic, essentially "or."

Logic tells us we can also add quantifiers.

Cylindrical algebraic decomposition (QE for RCF) is used in robotic motion planning, graphics, control theory.

Mathematica, Redlog, and some SMT solvers (Z3, CVC3) handle quantifier-elimination over the reals.

Presburger arithmetic has been used in verification.

I know of no applications to mathematics.

Logic and optimization

An observation: the Cohn-Elkies method amounts to an optimization problem, over a space of functions.

If we can synthesize programs, why can't we synthesize functions with certain properties?

Automated theorem provers

There are a number of first-order theorem provers, including E, Spass, Vampire, Prover9, Waldmeister.

There are various approaches:

- resolution theorem provers
- tableaux theorem provers
- instantiation-based methods
- model evolution

These are used in hardware and software verification.

The Robbins conjecture

In the 1930's, Robbins conjectured that these axioms characterize boolean algebras:

- Associativity: $a \vee (b \vee c) = (a \vee b) \vee c$
- Commutativity: $a \vee b = b \vee a$
- The Robbins equation: $\neg(\neg(a \vee b) \vee \neg(a \vee \neg b)) = a$

William McCune proved this in 1996, using his theorem prover *EQP*.

Model finders

Some software that searches for small finite models of first-order statements:

- Mace
- Finder (Finite Domain Enumerator)
- SEM (A System for Enumerating Models)
- Kodkod

Kodkod is based on SAT technology, and is used for "code checking, test-case generation, declarative execution, declarative configuration, and lightweight analysis of Alloy, UML, and Isabelle/HOL."

Loops

A *quasigroup* is a set with a binary operation such that for every a and b there is are unique x and y such that

- $a \cdot x = b$
- $y \cdot a = b$

A *loop* is a quasigroup with an identity.

Researchers like Ken Kunen and Simon Colton have used both model constructors and resolution theorem provers to prove things about loops and other nonassociative algebraic structures.

Model checking

Naïve idea:

- Describe a system as a finite state automaton.
- Characterize the initial states and "bad" states.
- Exhaustively search to see if we can reach a bad state.

Advances:

- Allow infinitely many states.
- Use clever representations for states and transitions.
- Use special purpose languages to describe properties.
- Use CEGAR: CounterExample-Guided Abstraction Refinement.

Model checking

Model checking methods have been enormously successful in industry.

CEGAR is mathematically familiar:

- To prove a theorem, consider possible counterexamples.
- Gradually develop more refined information about what a counterexample might look like.
- Eventually rule out all possibilities.

But I know of no applications to pure mathematics to date.

Contents

"Conventional" computer-assisted proof:

- carrying out long, difficult, computations
- proof by exhaustion

Formal methods for discovery:

- finding mathematical objects
- finding proofs

Formal methods for verification:

- verifying ordinary mathematical proofs
- verifying computations.

Formal verification in industry

Formal methods are commonly used:

- Intel and AMD use ITP to verify processors.
- Microsoft uses formal tools such as Boogie and SLAM to verify programs and drivers.
- Xavier Leroy has verified the correctness of a C compiler.
- Airbus uses formal methods to verify avionics software.
- Toyota uses formal methods for hybrid systems to verify control systems.
- Formal methods were used to verify Paris' driverless line 14 of the Metro.
- The NSA uses (it seems) formal methods to verify cryptographic algorithms.

Formal verification in mathematics

There is no sharp line between industrial and mathematical verification:

- Designs and specifications are expressed in mathematical terms.
- Claims rely on background mathematical knowledge.

I will focus, however, on verifying mathematics.

- Problems are conceptually deeper, less heterogeneous.
- More user interaction is needed.

Interactive theorem proving

Working with a proof assistant, users construct a formal axiomatic proof.

In most systems, this proof object can be extracted and verified independently.

Interactive theorem proving

Some systems with large mathematical libraries:

- Mizar (set theory)
- HOL (simple type theory)
- Isabelle (simple type theory)
- HOL light (simple type theory)
- Coq (constructive dependent type theory)
- ACL2 (primitive recursive arithmetic)
- PVS (classical dependent type theory)

Interactive theorem proving

Some theorems formalized to date:

- the prime number theorem
- the four-color theorem
- the Jordan curve theorem
- Gödel's first and second incompleteness theorems
- Dirichlet's theorem on primes in an arithmetic progression
- Cartan fixed-point theorems

There are good libraries for elementary number theory, real and complex analysis, point-set topology, measure-theoretic probability, abstract algebra, Galois theory, ...

Interactive theorem proving

Georges Gonthier and coworkers verified the Feit-Thompson Odd Order Theorem in Coq.

- The original 1963 journal publication ran 255 pages.
- The formal proof is constructive.
- The development includes libraries for finite group theory, linear algebra, and representation theory.

The project was completed on September 20, 2012, with roughly

- 150,000 lines of code,
- 4,000 definitions, and
- 13,000 lemmas and theorems.

Interactive theorem proving

Hales announced the completion of the formal verification of the Kepler conjecture (*Flyspeck*) in August 2014.

- Most of the proof was verified in HOL light.
- The classification of tame graphs was verified in Isabelle.
- Verifying several hundred nonlinear inequalities required roughly 5000 processor hours on the Microsoft Azure cloud.

Interactive theorem proving

Vladimir Voevodsky has launched a project to develop "univalent foundations."

- Constructive dependent type theory has natural homotopy-theoretic interpretations (Voevodsky, Awodey and Warren).
- Rules for equality characterize equivalence "up to homotopy."
- One can consistently add an axiom to the effect that "isomorphic structures are identical."

This makes it possible to reason "homotopically" in systems like Coq and Agda.

Verifying computation

Important computational proofs have been verified:

- The four color theorem (Gonthier, 2004)
- The Kepler conjecture (Hales et al., 2014)

Tucker's theorem is another obvious candidate.

Tucker has founded a group around developing methods and applications for validated numerics.

Verifying computation

Various aspects of Kenzo (computation in algebraic topology) have been verified:

- in ACL2 (simplicial sets, simplicial polynomials)
- in Isabelle (the basic perturbation lemma)
- in Coq/SSReflect (effective homology of bicomplexes, discrete vector fields)

Verifying computation

Some approaches:

- rewrite the computations to construct proofs as well (Flyspeck: nonlinear bounds)
- verify certificates (e.g. proof sketches, duality in linear programming)
- verify the algorithm, and then execute it with a specialized (trusted) evaluator (Four color theorem)
- verify the algorithm, extract code, and run it (trust a compiler or interpreter) (Flyspeck: enumeration of tame graphs)

Conclusions

Formal methods are valuable to mathematics:

- They provide additional precision.
- They provide strong guarantees of correctness.
- They make it possible to verify mathematical computation.
- They make it possible to use automation in a reliable way.
- They provide a language for specifying constraints and search problems.
- They provide powerful methods for traversing a search space efficiently.
- They assist in the storing, finding, and communicating mathematical knowledge.

Conclusions

Final message:

- The goal of mathematics is to get at the truth.
- Computers change the kinds of proofs that we can discover and verify.
- In the long run, formal methods *will* play an important role in mathematics.

A final thought

Generally, computer science, that no-nonsense child of mathematical logic, will exert growing influence on our thinking about the languages by which we express our vision of mathematics. — Yuri Manin