ADVENTURES IN THE VERIFICATION OF MATHEMATICS

by
Harvey M. Friedman
Computer Science Colloquium
Ohio State University
June 8, 2006

Verifying programs and verifying mathematics are two distinct enterprises which are becoming more and more closely related. Programs are to be verified by verifying associated mathematical statements.

Mathematical statements arising from program verification are believed to be much easier to deal with than statements coming from serious mathematics. At least this is true for "normal programming".

I am optimistic about adjusting existing tools for math verification to work wonders for program verification. This depends on work by computer scientists on the design of programming systems which automatically generate the simple mathematical statements to be verified.

1. COMING DOWN TO REALITY.

In the early days (1960's) there was the idea that computers could replace mathematicians, and prove serious mathematical theorems entirely on their own.

Even to this day, the success along these lines is extremely limited, and this idea has been all but abandoned. An exception is plane geometry, where beautiful things of interest to humans are done solely by computers. See, e.g.,

S.-C. Chou. *Mechanical geometry theorem proving*. Reidel, Dortrecht, 1988.
W.-T. Wu. *Mechanical Theorem Proving in Geometries*. Number 1 in Texts and Monographs in Symbolic Computation. Springer, Wien, 1994.

But this is still very far from general purpose ordinary mathematics, and worthless for program verification.

So why has this dream of unaided computer mathematics collapsed?

The original idea of Hilbert was that there should be a decision procedure for all of mathematics. This was refuted by Turing, Gödel, Church, et al, in very strong ways.

E.g., there is no decision procedure for deciding whether a sentence involving

□ integers, □, □, +, •

is true. This was later improved greatly (starting in 1970) by the result that there is no decision procedure for deciding whether a sentence

$(\square x_1,\ldots,x_9 \;\square\; N)(P(x_1,\ldots,x_n) = 0)$

is true, where P is a polynomial with integer coefficients.

However, this is not even close to the full story.

Pointing to possible victory for computers was a number of impressive decision procedures in very serious mathematical contexts. These were known well before the 1960's.

One is for all sentences involving

□ integers, □ integers, □, □, , □ , □ , +, -, <, 0, 1.

Another is for all sentences involving

□ reals, □ reals, □, □, , □ , □ , +, -, •, <, 0, 1.

Counterintuitive!, since reals are more sophisticated than the integers, and we are getting away with •.
On the other hand, computational complexity considerations enter in as follows.

The (Z,+,-,0,1,<) decision problem is known to be nondeterministic double exponential time complete. So all known algorithms run in triple exponential time.

The (¬,+,-,•,0,1,<) decision procedure is known to be nondeterministic exponential time hard and exponential space easy. Therefore, all known algorithms run in double exponential time.

These are bad news, but the computers can fight back.

First of all, the decision procedure for (Z,+,-,0,1,<) works pretty well on real world examples, thanks to some optimization work. It works splendidly on the universal sentences in (Z,+,-,0,1,<).

There is an exponential time decision procedure for the universal sentences in (¬,+,-,•,0,1,<), which is much better than for the full theory.

The bad news is that there are all sorts of real world examples of universal sentences in (¬,+,-,•,0,1,<) where the existing decision procedures blow up.

Exponential time algorithms themselves may or may not be practical. The most well known ones in computer science are those used to recognize satisfiability in propositional calculus.

Any practical satisfiability recognizer can also practically provide a truth assignment for any satisfiable formula.

The propositional satisfiability problems that naturally arise in the verification of mathematics are completely clobbered by existing algorithms.

One can also make inroads into restricted sentences in first order predicate calculus, with and without equality. But satisfiability of sentences with even just a few (two?) universal quantifiers in a binary function symbol, is not decidable.

2. WHY VERIFY MATHEMATICS?

The verification of mathematics, in any reasonably general purpose sense, now goes under the buzzword of

proof assistants

I.e., this is a human/machine interactive process.

But why do this at all? Some good reasons.

i. There is a subject called proof theory, in mathematical logic. But it doesn’t say much about the structure of actual proofs. We need to get our hands on actual mathematical proofs in standardized form. There is the belief that sophisticated proofs, which are much longer than toy proofs, have features that do not appear in toy proofs. This data and experience can only be conveniently obtained through proof assistants.

ii. To make good on a philosophical claim made in the foundations of mathematics. That there is an objective standard for whether or not something has really been proved. Justification of the special feature of mathematics – certainty.

iii. To refute (conscious and unconscious) skeptics among mathematicians, who, in some form, deny that mathematics is capable of formalization. Whereas there may be senses in which they are right, because of work on verification of mathematics, we know that there are clear senses in which they are wrong.

iv. To settle disputes as to whether or not something has really been proved. This occurs infrequently, but has occurred in connection with Kepler’s conjecture about sphere packing. Also with the classification of finite simple groups, there is unlikely to be any full record left from living mathematicians for the future, without formal verification.

v. To support the formal verification of software, and computer systems in general.

3. Proof assistants (generalities).

Proof assistants are now very advanced in some respects, with thousands of man years in them, including lots of stuff thrown away. This has been going on since the 1960’s. Pat Suppes was a very early pioneer at Stanford.

By now, as we shall see later, very serious mathematical theorems continue to be formally verified through these proof assistants. The mathematician sits interactively with the proof assistant. The process is driven by the human, who tries to get the proof assistant to accept the human’s moves.

When successful, most proof assistants generate what is called a “proof object”, which is a file containing a proof in a very low level system. The file can be checked by an independent program of a simple sort – incomparably simpler than the code for the proof assistant itself.

Of course, then there is the question of how to verify this simpler code. Verifying this simpler code in the original assistant seems unsatisfactory. So a question is: in what precise sense can we achieve certainty or near certainty? Not clear.

1. The user orchestrates the refining of goals and hypotheses according to a natural deduction framework. This is in accordance with the general logical organization of actual mathematical proofs.
2. The user cites definitions and theorems from ‘libraries’. The proper construction of libraries is absolutely crucial in practice. Supports strong reusability.

3. It is also crucial that the proof assistant be able to make relatively trivial inferences on its own. Experience shows that otherwise the process is just too time consuming.

A lot of effort has gone into 1,2,3.

1) has stabilized long ago, although there is certainly a lot of room for improvement in terms of readability of output and user interfacing.
Readability of output has not been a high priority for proof assistants, and is only recently being seriously addressed.

3) uses a hodge podge of goodies that have been developed over decades. These include

a. General purpose. Various general purpose simplification procedures for expressions. These can be user directed, at least in Isabelle. User can say what simplifies to what. This is used to avoid having to enter simplified forms, and also internally when the proof assistant tries to fill in steps.

b. General purpose. Decision procedure for propositional calculus. Various decision procedures for fragments of predicate calculus with and without equality.

c. General purpose. Resolution theorem proving methods for predicate calculus. This is an old method due to J.A. Robinson, 1965. Since then it has been steadily improved, and the one I hear most often about is called Otter.

d. Special purpose. Domain specific decision procedures for various fragments of mathematics.

4. PROOF ASSISTANTS (MORE).

I am now going to go into some more detail about proof assistants. I have taken some material from

Little Engines of Proof, by Natarjan Shankar, FME 2002: Formal Methods - Getting it Right, Copengahen.
Available at
http://citeseer.ist.psu.edu/shankar02little.html

At the most general level, there have emerged two approaches. One is represented by Alan Robinson's general purpose resolution method, based on simple uniform procedures guided by heuristics.

The other school pioneered by Hao Wang, pushes problem specific combinations of decision and semi-decision procedures.

Current thinking is: abandon the first for the second. This also incorporates the first as just one tool.

(Shankar) State of the art:

i. high powered propositional satisfiability solvers.
ii. ground decision procedures for equality and arithmetic.
iii. decision procedures for integers and reals, and
iv. abstraction methods for nicely approximating problems over infinite domains.
There are also nice ways of combining different decision procedures over different domains (with serious limitations).

Not many relevant problems are stated in a form that is readily attackable with existing decision procedures.

However, humans can decompose these problems into decidable subproblems. Also to some extent, computers can too.

According to Shankar:

“The construction of modular inference procedures is a challenging research issue in automated reasoning.
Work on little engines of proof has been gathering steam lately. Many groups are actively engaged in the construction of little proof engines, while others are putting in place the train tracks on which these engines can run.

PVS itself can be seen as an attempt to unify many different inference procedures: typechecking, ground decision procedures, simplification, rewriting, MONA [EKM98], model checking [CGP99], abstraction, and static analysis, within a single system with an expressive language for writing mathematics.”

Along with SAT, two decision procedures stand out.

One is Presburger arithmetic, which in its full blown form is the theory of (¬,Q,Z,<,0, 1,+,-). This has a good decision procedure that works pretty well in practice.

Another that we haven’t mentioned yet is WS1S = weak monadic second-order logic with 1 successor. The domains are N and the collection of finite subsets of N, and we have the successor function on N. From this we can easily define <. This has a decision procedure that is indefinitely iterated exponential time complete in theory, OUCH!!, but is reasonably nice in practice.

In pure set theory, there is are a lot of implemented decision procedures coming out of the SETL group led by Jack Schwartz.

According to Shankar,

“WS1S is a natural formalism for many applications, particularly for parametric systems. The logic can be used to capture interesting datatypes such as regular expressions, lists, queues, and arrays.”

Since so few natural problems fall within a single one of these procedures, there has been considerable investigation

into how to combine different decision procedures.
The most well known and implemented method is the Nelson-Oppen procedure.

THEOREM (Nelson-Oppen). Suppose $T_1,...,T_n$ have disjoint languages (except for =), and no finite models. Suppose the universal fragments of each T_i are decidable. Then the universal fragment of T_1 □ ... □ T_n is decidable.

The procedure for the universal fragment of T_1 □ ... □ T_n may be very impractical, even though the procedures for the T_i individually are practical. Works much better if the theories have additional properties that are often met in existing decision procedures.

The big limitation for this method is the hypothesis of disjoint languages. Because of this, it turns out that, generally, T_1 □ ... □ T_n is too weak a theory to be all that useful.

The point of recent joint work with Avigad is to study a fundamental case of T_1 □ T_2 where the languages are not disjoint. We still obtain decidability of the universal consequences (with difficult-ty), and some undecidability results for more complicated consequences (also with difficulty).

Shankar lists some challenges.

"The Complexity Challenge. Many decision procedures are of exponential, superexponential, or nonelementary complexity. However, this complexity often does not manifest itself on practical examples. ... The challenge here is to understand the ways in which one can over-come complexity bounds on the problems that arise in practice through heuristic or algorithmic means.

"The Theory Challenge. Inference procedures are hard to build, extend, and maintain. The past experience has been that good theory leads to simpler decision procedures with greater efficiency. ... Methods derived by specializing general-purpose methods like resolution and rewriting can also simplify the construction of decision procedures.

"The Modularity Challenge. As we have already noted, inference procedures need rich programmer interfaces (APIs) [BM86,FORS01]. Boyer and Moore [BM86] write: . . . the

black box nature of the decision procedure is frequently destroyed by the need to integrate it. The integration forces into the theorem prover much knowledge of the inner workings of the procedure and forces into the procedure many features that are unnecessary when the problem is considered in isolation.
"The modularity challenge is a significant one. Butler Lampson has argued that software components have always failed at low levels of granularity (see http://research.microsoft.com/lampson/Slides/ReusableComponentsAbstract.htm). He says that successful software components are those at the level of a database, a compiler, or a theorem prover, but not decision procedures, constraint solvers, or unification procedures. For inter-operation between inference components, we also need compatible logics, languages, and term and proof representations.

"The Integration Challenge. The availability of good inference components is a prerequisite for integration, but we also need to find effective ways of combining these components in complementary ways. The combination of decision procedures with model checking in predicate and data abstraction is a case where such a complementary integration is remarkably effective. Other such examples include the combination of unification/matching procedures and constraint solving, and typechecking and ground decision procedures.

"The Verification Challenge. How do we know that our inference procedures are sound? This question is often asked by those who wish to apply inference procedures in contexts where a high level of manifest assurance is required. This question has been addressed in a number of ways. ... Proof objects have also been widely used as a way of validating inference procedures and securing mobile code [Nec97]. ...
The verification of decision procedures is actually well within the realm of feasible, and recently, there have been several successful attempts in this direction [Thie98,FS02].

5. SOME PARTICULAR PROOF ASSISTANTS.

Slides:
Formalization of Mathematics
Freek Wiedijk

Radboud University Nijmegen
TYPES Summer School 2005
GÄoteborg, Sweden
2005 08 23, 11:10

Freek lists four “prehistorical” proof assistants for mathematics:

1968 Automath
Netherlands, de Bruijn
1971 nqthm US, Boyer & Moore
1972 LCF   UK, Milner
1973 Mizar
Poland, Trybulec

Freek lists seven current systems for mathematics

Mizar ......most mathematical

LCF---→**HOL**--→**Isabelle**... most pure

Automath--→ **Coq**..most logical
NuPRL
PVS.........most popular

nqthm ----→ **ACL2**.....most computational

Arrows also point from Mizar to Isabelle, and from LCF to Coq and PVS. Also an arrow from nqthm to PVS.

From Freek:

Formalizing 100 Theorems
Theorems not formalized yet in italics.

The Irrationality of the Square Root of 2

Fundamental Theorem of Algebra

The Denumerability of the Rational Numbers

Pythagorean Theorem

Prime Number Theorem

Gödel's Incompleteness Theorem

Law of Quadratic Reciprocity

The Impossibility of Trisecting the Angle and Doubling the Cube

The Area of a Circle

Euler's Generalization of Fermat's Little Theorem

The Infinitude of Primes

The Independence of the Parallel Postulate

Polyhedron Formula

Euler's Summation of 1 + (1/2)^2 + (1/3)^2 + ...

Fundamental Theorem of Integral Calculus

Insolvability of General Higher Degree Equations

De Moivre's Theorem

Liouville's Theorem and the Construction of Trancendental Numbers

Four Squares Theorem

All Primes (1 mod 4) Equal the Sum of Two Squares

Green's Theorem

The Non-Denumerability of the Continuum

Formula for Pythagorean Triples

The Undecidability of the Continuum Hypothesis

Schroeder-Bernstein Theorem

Leibnitz's Series for Pi

Sum of the Angles of a Triangle

Pascal's Hexagon Theorem

Feuerbach's Theorem

The Ballot Problem

Ramsey's Theorem

The Four Color Problem

Fermat's Last Theorem

Divergence of the Harmonic Series

Taylor's Theorem

Brouwer Fixed Point Theorem

The Solution of a Cubic

Arithmetic Mean/Geometric Mean

Solutions to Pell's Equation

Minkowski's Fundamental Theorem

Puiseux's Theorem

Sum of the Reciprocals of the Triangular Numbers

The Isoperimetric Theorem

The Binomial Theorem

The Partition Theorem

The Solution of the General Quartic Equation

The Central Limit Theorem

Dirichlet's Theorem

The Cayley-Hamilton Thoerem

The Number of Platonic Solids

Wilson's Theorem

The Number of Subsets of a Set

Pi is Trancendental

Konigsberg Bridges Problem

Product of Segments of Chords

*The Hermite-Lindemann
Transcendence Theorem*

Heron's Formula

Formula for the Number of Combinations

The Laws of Large Numbers

Bezout's Theorem

Theorem of Ceva

Fair Games Theorem

Cantor's Theorem

L'Hôpital's Rule

Isosceles Triangle Theorem

Sum of a Geometric Series

e is Transcendental

Sum of an arithmetic series

Greatest Common Divisor Algorithm

The Perfect Number Theorem

Order of a Subgroup

Sylow's Theorem

Ascending or Descending Sequences

The Principle of Mathematical Induction

The Mean Value Theorem

Fourier Series

Sum of kth powers

The Cauchy-Schwarz Inequality

The Intermediate Value Theorem

The Fundamental Theorem of Arithmetic

Divergence of the Prime Reciprocal Series

Dissection of Cubes (J.E. Littlewood's "elegant" proof)

The Friendship Theorem

Morley's Theorem

Divisibility by 3 Rule

Lebesgue Measure and Integration

Desargues's Theorem

Derangements Formula

The Factor and Remainder Theorems

Stirling's Formula

The Triangle Inequality

Pick's Theorem

The Birthday Problem

The Law of Cosines

Ptolemy's Theorem

Principle of Inclusion/Exclusion

Cramer's Rule

Bertrand's Postulate

Buffon Needle Problem

Additions from Freek:

Atiyah-Singer Index Theorem
Baker's Theorem on Linear Forms in Logarithms
Black-Scholes Formula
Borsuk-Ulam Theorem
Cauchy's Integral Theorem
Cauchy's Residue Theorem
Chen's theorem
Classification of Finite Simple Groups
Gödel's Completeness Theorem
Gödel's Second Incompleteness Theorem
Green-Tao Theorem
Fundamental Theorem of Galois Theory
Heine-Borel Theorem
Hilbert Basis Theorem
Hilbert Nullstellensatz
Hilbert-Waring theorem
Invariance of Dimension
Jordan Curve Theorem
Lie's work relating Algebras and Groups
Nash's Theorem
Perelman's proof of the Poincaré Conjecture
Stoke's Theorem
Stone-Weierstrass Theorem
Thales' Theorem
Yoneda lemma

(last modification 2006-05-23)

State of the art: recent big formalizations.

PRIME NUMBER THEOREM
Jeremy Avigad e.a.
1 megabyte = 30,000 lines = 42 files of Isabelle/HOL
the elementary proof by Selberg from 1948

FOUR COLOR THEOREM
Georges Gonthier:
(2.5 megabytes = 60,000 lines = 132 files of Coq 7.3.1
streamlined proof by Robertson, Sanders, Seymour & Thomas
from 1996
Contains very interesting `own' proof language on top of
Coq heavily relies on reflection `this formalization really

needs Coq'

<u>JORDAN CURVE THEOREM</u>
Tom Hales:
2.1 megabytes = 75,000 lines = 15 files of HOL Light.
Proof thru the Kuratowski characterization ofplanarity.
Current *Biggies*:

Formalization of a complete `advanced' mathematics textbook:

A Compendium of Continuous Lattices, by Gierz et al.

Project led by Grzegorz Bancerek
about 70% formalized
4.4 megabytes = 127,000 lines = 58 files of Mizar.

Flyspeck project.
Kepler, 1661:
Is the way we customarily stacks oranges the most efficient way to stack spheres?
Tom Hales, 1998: yes !

Proof depends on computer checking 3 gigabytes programs & data, couple of months of computer time. FlysPecK project: `Formal Proof of Kepler'

So why did the qed project not take off as intended?

Reason ONE: incompatible systems. set theory type theory
higher order logic PRA classical constructive
extensional intensional
impredicative predicative
choice only countable choice

There are projects underway to try to put systems together. Michael Kohlhase, and Carsten Schurmann.

Reason TWO: why mathematicians are not interested (yet)
the cost is too high. . .

de Bruijn factor =

size of formalization ÷
size of normal text

question: is this a constant?

experimental: around 4

de Bruijn factor in time =

time to formalize ÷
time to understand

much larger than 4.

formalizing one textbook page = 1 man/week = 40 man hours

. . . and the gain is too little

NOT impossibly expensive
formalizing all of undergraduate mathematics = 140 man
years the price of about one Hollywood movie.

BUT: after formalization we just have a big
incomprehensible file. We don't have a good argument yet
for spending that money.

AND: it does not look like mathematics. Even in Mizar and
Isar: still looks like code.

Mizar is based on set theory but it is a typed system.

Mizar types are soft types:
M : N(t1; : : : ; tn)
should really be read as a predicate.

Think of Mizar types as predicates that the system keeps
track of for you.

Mizar Mathematical Library
the biggest library of formalized mathematics
49,588 lemmas
1,820,879 lines of `code'
64 megabytes
165 `authors'
912 `articles'

Mizar versus Isar
some reasons to prefer Mizar over Isar

the set theory of Mizar is much more powerful and
expressive than the HOL logic of Isabelle/HOL

Mizar is much more able to talk about abstract mathematics, and in particular about algebraic structures, with nice notation

dependent types are way cool

some reasons to prefer Isar over Mizar

Isabelle gives you an interactive system

Isabelle allows you to mix declarative and procedural proof

Isabelle has much more possibilities of automation

Isabelle allows you to define binders

Mizar is about as complex as the Pascal programming language

Will proof assistants ever become common among mathematicians?

The experienced user's answer: 50 years.

6. Real algebra without distributivity.

Decision procedures for the reals with both addition and multiplication exist (Tarski) but are quite bad in practice. There are probably a number of reasons for this.

One particular reason is that it is very hard to automate the judgment of whether or not distributivity should be applied. $r(s+t) = rs + rt$. Sometimes yes, sometimes no.

So one idea is to have the user always control use of distributivity, and let the computer try to handle everything else.

This leads to fragments of the usual theory of the reals, where distributivity is dropped.

One of the theories that we study is written $T[Q] = T_{add}[Q]$ □ $T_{mult}[Q]$, where

$T_{add}[Q]$ is based on the symbols

$0,1,+,-,<,f_a$, a □ Q

and T_{mult}[Q] is based on the symbols

0,1,•,÷,<,f_a, a □ Q

Here f_a is scalar multiplication by the rational Q.

T_{add}[Q] and T_{mult}[Q] consist of the true sentences in their respective languages. They have very elegant complete axiomatizations.

It is not at all clear just what T[Q] proves or doesn't. Even the purely universal sentences.

THEOREM 1. There is a decision procedure for determining whether a universal sentence in the language of T[Q] is provable in T[Q].

It is not clear whether this can be made efficient. However

THEOREM 2. Theorem 1 for equations can be done efficiently.

THEOREM 3. If Hilbert's 10th problem fails for Q (believed) then there is no decision procedure for determining whether an existential sentence in the language of T[Q] provable in T[Q].

THEOREM 4. There is no decision procedure for determining whether a □□□□...□ sentence in the language of T[Q] is provable in T[Q].

REFERENCES

Formalization of mathematics, Freek Wiediuk, Radboud University Nijmegen, TYPES Summer School 2005, GÄoteborg, Sweden, 2005 08 23, 11:10, lecture notes.

Little Engines of Proof, by Natarjan Shankar, FME 2002: Formal Methods - Getting it Right, Copengahen, http://citeseer.ist.psu.edu/shankar02little.html

J. Avigad, and H. Friedman, Combining decision procedures for the reals, January 31, 2006, submitted, http://www.math.ohio-state.edu/%7Efriedman/