COMPUTER ASSISTED CERTAINTY

by
Harvey M. Friedman
Computer Science Lecture Series
University of Kentucky
Ohio State University
January 25, 2007

ABSTRACT

Certainty (and the lack thereof) is a major issue in mathematics and computer science. Mathematicians strongly believe in a special kind of certainty for their theorems.

Computer scientists, programmers, and especially their clients, want much more certainty than they (know that they) presently have.

Theoretical certainty was achieved in mathematics through the development of set theory, and the epsilon/delta and related methods in the late 1900's, followed by formal axioms of set theory in the early 2000's (ZFC).

Practical certainty has only been achieved for any substantial variety of mathematics over the last few decades through the complex interactive computer systems now called proof assistants - such as Mizar and Isabelle.

We discuss a number of theoretical and practical issues that have arisen in the design and application of these proof assistants.

A major effective idea has been the development and application of decision procedures for (very) restricted kinds of mathematics. Recently, experience with the practical difficulties surrounding real number algebra has led to the theoretical study of real number algebra without distributivity. Also, promising applications to program verification in limited contexts are being developed.

Currently, proof verification is far too difficult and expensive. We will discuss a plan of attack aimed towards meeting this challenge.

VERIFYING MATHEMATICS AND VERIFYING PROGRAMS

It is important to distinguish these two distinct aims. We will mostly talk about the first.

Current thinking is that verification of general purpose software must rely on the verification of automatically generated mathematical statements that arise from the software -- preferably in the course of development of that software.

In general purpose practical setups affecting practice, human intervention in the verification process must be kept to an absolute minimum.

Yet human intervention for the verification of substantial mathematics has proved to be extremely demanding.

FORTUNATELY, the mathematical statements properly arising from program verification are believed to be very much easier to deal with than statements coming from serious mathematics.

I am optimistic about adjusting existing tools for math verification to work wonders for program verification.

This will depend 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.

Success along these lines is very limited, and this idea has been nearly 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.

However, this is very far from ordinary mathematics, worthless for program verification.

Why has 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, in very strong ways.

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

□ integers, □, □, +, •

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

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

is true, where P is a polynomial with integer coefficients. (Unsolvability of Diophantine problems over the nonnegative integers).

However, this is NOT close to the full story!

Computers fought back with impressive decision procedures in very serious mathematical contexts.

One (Presburger) is for all sentences involving

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

Another (Tarski) is for all sentences involving

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

This is very counterintuitive, since the reals are much more sophisticated than the integers, and we are getting away with using •!!

HOWEVER, computational complexity considerations enter in as follows.

The (Z,+,-,0,1,<) decision procedure 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 fight back.

Decision procedure for (Z,+,-,0,1,<) works well in practice, thanks to optimization work. Splendid for universal sentences in (Z,+,-,0,1,<).

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

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

Exponential time may/may not be practical. Most well known: those used to recognize satisfiability in propositional calculus. SAT.

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

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

One can also make inroads into restricted sentences in first order predicate calcu-lus, with and without equali-ty. But satisfiability of sentences with even just one universal quantifier and one binary function symbol, or two unary function symbols, with equality, is not decidable

(Gurevich).

2. WHY VERIFY MATHEMATICS?

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

proof assistants

I.e., an interactive process.

But why do this at all?

i. There is a subject called proof theory, in logic. It doesn't deal much with actual proofs. We need to get our hands on actual mathematical proofs in standardized form. Sophisticated proofs have features that do not appear in toy proofs. This data 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. I.e., 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 regards to the classification of finite simple groups, there is growing concern that there is unlikely to be any full record left from living mathematicians, without formal verification. (The existing proof is being de-bugged by mathematicians pushing 60 years of age, who may not finish their work).

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. Development since the 1960’s.

By now, 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 very low level form. The file can be checked by an independent program of a simple sort - incomparably simpler than the code for the proof assistant itself.

DIGRESSION: 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 certain-ty or near certainty? Not clear. A deeper look at this moves us into sophisticated logical and philosophical issues. END.

How do proof assistants work?

1. The user orchestrates the refining of goals and hypotheses according to a natural deduction framework. This is very much like the general logical organization of actual mathematical proofs.

2. The user cites definitions and theorems (some in the form of rules) from ‘libraries’. The proper construction of libraries is absolutely crucial in practice. They support 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.

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. Used to avoid having to enter simplified forms, and also internally.

b. General purpose. Decision procedure for propositional calculus (SAT). Various decision procedures for fragments of predicate calculus.

c. General purpose. Resolution theorem proving methods for predicate calculus. Goes back to J.A. Robinson, 1965. Has been steadily improved, e.g., with the program Otter.

d. Special purpose. Domain specific decision procedures for various fragments of mathematics. There is a lot of current excitement, prom-ise, and expectation. [Particularly useful are quanitifer free forms of quantified formulas. HMF]

4. PROOF ASSISTANTS (MORE).

I have taken material from

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

At most general level, two approaches. One exemplified by J.A. Robinson's general purpose resolution method - simple uniform procedures guided by heuristics.

Second pioneered by Hao Wang - pushes problem specific combinations of decision and semi-decision procedures.

Current thinking: abandon first for second. Incorporate first as just one tool.

(Shankar) State of the art:

i. High powered propositional satisfiability solvers (SAT).
ii. Ground decision proced-ures for equality and arithmetic.
iii. Decision procedures for integers and reals.
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. BUT: modularity.

"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."

Along with SAT, two decision procedures stand out.

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

Another 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 get <.

"WS1S is a natural formalism for many applications. It can be used to capture interesting datatypes such as regular expressions, lists, queues, and arrays."

In pure set theory, working decision procedures from SETL group led by Jack Schwartz.

BUT, few problems fall within just one of these: how to combine different decision procedures?

Most well known 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 $T_1 \cup ... \cup T_n$ is decidable.

BIG limitation: hypothesis of disjoint languages. Generally, $T_1 \cup ... \cup T_n$ is too weak to be all that useful.

Joint work with Avigad: to study a fundamental case of $T_1 \cup T_2$ where the languages are not disjoint.

We still obtain decidability of the universal consequences (with difficulty), and some undecidability results for more complicated consequences (also with difficulty).

Shankar lists some challenges - paraphrased here.

"The Complexity Challenge. Many decision procedures have very high complexity in theory but are good in practice. Why? How can we overcome high complexity?

"The Theory Challenge. Inference procedures are hard to build, extend, and maintain. Need to specialize general purpose methods like resolution and rewriting.

"The Modularity Challenge. Black box nature of a decision procedure often destroyed by the need to integrate it. Integration forces one to work with inner workings."

"The Integration Challenge. Need effective ways to com-bine inference components. Combining decision procedures with model checking is effective. Combining unification/ matching procedures and constraint solving, and type checking with ground decision procedures, is effective.

"The Verification Challenge. How do we know that our inference procedures are sound?

Proof objects have been widely used for validation. Outright verification of decision procedures has recent success.

5. SOME PARTICULAR PROOF ASSISTANTS

Freek Wiedijk, Formalization of Mathematics, http://www.cs.ru.nl/~freek/talks/index.html manuscript 35.

Freek lists four "prehistorical" proof assistants:

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.

HOL, Isabelle. Most pure.
Coq, NuPRL. Most logical.
PVS. Most popular.
ACL2. Most computational.

Formalizing 100 Theorems

Theorems not formalized yet in italics. (Freek).

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 Theorem
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

State of the art: recent big formalizations. (Freek).

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

FOUR COLOR THEOREM
Georges Gonthier:
(2.5 megabytes = 60,000 lines = 132 files of Coq 7.3.1
Via Robertson, Sanders, Seymour & Thomas from 1996.

JORDAN CURVE THEOREM
Tom Hales:
2.1 megabytes = 75,000 lines = 15 files of HOL Light.
Proof thru Kuratowski characterization of planarity.
Current *Biggies*:

Formalization of a complete "advanced" mathematics textbook:

A Compendium of Continuous Lattices, by Gierz et al.

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

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

Proof relies on running 3 gigabytes programs & data, 2 months.
FlysPecK project:
"Formal Proof of Kepler"
Estimated time: 10 years.

So why hasn't proof checking really taken off?

Freek:

Reason ONE: incompatible systems. set theory type theory higher order logic classical constructive etc.

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

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

. . . and the gain is too little
NOT impossibly expensive:

formalizing all undergrad math = 140 man years: the price of one Hollywood movie.

BUT: after formalization we just have a big incomprehensible file. Need good argument yet for spending that money.

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

Mizar Math Library: the biggest library of formalized mathematics
49,588 lemmas
1,820,879 lines of 'code'
64 megabytes
165 `authors'
912 `articles'
Will proof assistants ever become common among mathematicians?

Insider's answer: 50 years.

6. REAL ALGEBRA WITHOUT DISTRIBUTIVITY.

with J. Avigad, Combining decision procedures for the reals, Logical Methods in Computer Science 2(4:4), 2006.

Decision procedures for reals with +,• exist (Tarski) but are bad in practice. A number of reasons for this.

One is that it is hard to automate the judgment of whether or not to apply distributivity. $r(s+t) = rs + rt$.

Sometimes yes, sometimes no.

Idea: User controls use of distributivity. Computer controls everything else.

Leads to fragments of the usual theory of the field of reals, where distributivity is dropped.

One theory we study is:

$T[Q] = T_{add}[Q]$ □ $T_{mult}[Q]$, where

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

$$0,1,+,-,<,f_a,\ a \text{ □ } Q$$

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

$$0,1,\bullet,\div,<,f_a,\ a \text{ □ } Q$$

Here f_a is scalar multiplication by the rational a.

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

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. But:

THEOREM 2. Theorem 1 for equations can be made efficient, by practical standards.

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] is 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].

7. NEW PLAN OF ATTACK.

Proof verification is much too painful, still. What to do about it?

Overriding problem is:

computer cannot come up with various "obvious" things.

First separate the friendly obvious from the unfriendly obvious. Even heavy doses of human input of the friendly obvious is OK. Windowing and dialog boxes can minimize typing and searching.

We do **NOT** want to work hard to get the computer to do the friendly obvious.

We **WANT** to work hard to get the computer to do the unfriendly obvious.

We find that lots of purely logical manipulation is friendly obvious, including some inputting of terms. When the terms are (relatively) not obvious, inputting them in the proper positions is friendly obvious. This is good, because coming up with terms to use, and when to use them, has been the subject of a lot of research effort.

The unfriendly obvious is when the user must become distracted by details that are not germane to the proof, but are more general.

These will generally take the form of a few applications of very low level rules and facts that should be in the library.

An immediate challenge is theoretical/practical support for library creation.

We believe in an appropriate notion of “small fact” and “small rule”, of fundamental theoretical/practical significance in connection with diverse basic contexts.

This requires new kinds of “small” completeness theorems, and also new practical algorithmic studies for find-ing short paths from one small item to another.

In high level design, the computer maintains a finite set of windows, each devoted to a set of haves, and a single want.
Each window has a have/want proof: have’s cumulate and wants override. The computer runs the window splitting.

The user directs the logic by simple mouse clicks. Terms are entered by dialog boxes.

The user has a number of other options, such as library lookup, and just entering a low level step which should be gotten by the computer applying a few low level rules from the library.

There are various ways in which user/computer can help each other.

Development of “ideal elementary mathematics books”.