Formalizing Mathematics

John Harrison
Intel Corporation

Seminar, University of Pittsburgh

March 22, 2007

What is formalization of mathematics?

Two aspects, corresponding to Leibniz's *characteristica universalis* and *calculus ratiocinator*.

- Express *statement* of theorems in a formal language, typically in terms of primitive notions such as sets.
- Write *proofs* using a fixed set of formal inference rules, whose correct form can be checked algorithmically.

Correctness of a formal proof is an objective question, algorithmically checkable in principle.

Mathematics is reduced to sets

The explication of mathematical concepts in terms of sets is now quite widely accepted (see *Bourbaki*).

- A real number is a set of rational numbers . . .
- A Turing machine is a quintuple $(\Sigma, A, \ldots)$

Statements in such terms are generally considered clearer and more objective. (Consider pathological functions from real analysis . . .)

Symbolism is important

The use of symbolism in mathematics has been steadily increasing over the centuries:

> [Symbols] have invariably been introduced to make things easy. [. . .] by the aid of symbolism, we can make transitions in reasoning almost mechanically by the eye, which otherwise would call into play the higher faculties of the brain. [. . .] Civilisation advances by extending the number of important operations which can be performed without thinking about them.

Formalization is the key to rigour

Formalization now has a important conceptual role in principle:

> As to precision, we have now stated an absolute standard of rigor: A Mathematical proof is rigorous when it is (or could be) written out in the first-order predicate language $L(\in)$ as a sequence of inferences from the axioms ZFC, each inference made according to one of the stated rules. [. . .] When a proof is in doubt, its repair is usually just a partial approximation to the fully formal version.

What about in practice?

Logical symbolism in practice

Variables were used in logic long before they appeared in mathematics, but logical symbolism is rare in current mathematics.

Yet now, apart from the odd '$\Rightarrow$', logical relationships are usually expressed in natural language, with all its subtlety and ambiguity.

> "as far as the mathematical community is concerned George Boole has lived in vain"

Many mathematicians are probably unable to understand typical logical notation.

Formal proof in practice

Very few people do formal proofs:

> "this mechanical method of deducing some mathematical theorems has no practical value because it is too complicated in practice."

and those who do usually regret it:

> "my intellect never quite recovered from the strain of writing [*Principia Mathematica*]. I have been ever since definitely less capable of dealing with difficult abstractions than I was before."

However, now we have computers to check and even automatically generate formal proofs . . .

Why formalize?

There are two main reasons for formalizing mathematics:

- To show that it is possible, perhaps in pursuit of a philosophical thesis such as logicism
- To really improve the rigour and objectivity of mathematical proofs.

Only for the second objective do we need to *actually* formalize proofs.

Are proofs in doubt?

Mathematical proofs are subjected to peer review, but errors often escape unnoticed.

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

A book by Lecat gave 130 pages of errors made by major mathematicians up to 1900.

A similar book today would no doubt fill many volumes.

Most doubtful informal proofs

What are the proofs where we do in practice worry about correctness?

- Those that are just very long and involved. Classification of finite simple groups, Seymour-Robertson graph minor theorem
- Those that involve extensive computer checking that cannot in practice be verified by hand. Four-colour theorem, Hales's proof of the Kepler conjecture
- Those that are about very technical areas where complete rigour is painful. Some branches of proof theory, formal verification of hardware or software

Formal verification

In most software and hardware development, we lack even *informal* proofs of correctness.

Correctness of hardware, software, protocols etc. is routinely "established" by testing.

However, exhaustive testing is impossible and subtle bugs often escape detection until it's too late.

The consequences of bugs in the wild can be serious, even deadly.

Formal verification (*proving* correctness) seems the most satisfactory solution, but gives rise to large, ugly proofs.

The FDIV bug

A great stimulus to formal verification at Intel:

- Error in the floating-point division (FDIV) instruction on some early Intel®Pentium® processors in 1994
- Very rarely encountered, but was hit by a mathematician doing research in number theory.
- Intel eventually set aside US $\$475$ million to cover the costs of replacements.

We don't want something like that to happen again!

The trends are worrying . . .

Recent Intel processor generations (Pentium, P6 and Pentium 4) indicate:

- A 4-fold increase in overall complexity (lines of RTL . . .) per generation
- A 4-fold increase in design bugs per generation.
- Approximately 8000 bugs introduced during design of the Pentium 4.

Fortunately, pre-silicon detection rates are now very close to 100%, partly thanks to formal verification.

4-colour Theorem

Early history indicates fallibility of the traditional social process:

- Proof claimed by Kempe in 1879
- Flaw only point out in print by Heaywood in 1890

Later proof by Appel and Haken was apparently correct, but gave rise to a new worry:

- How to assess the correctness of a proof where many explicit configurations are checked by a computer program?

Most worries finally dispelled by Gonthier's formal proof in Coq.

Who checks the checker?

Why should we believe that a formally checked proof is more reliable than a hand proof or one supported by ad-hoc programs?

- What if the underlying logic is inconsistent? Many notable logicians from Curry to Martin-Löf have proposed systems that turned out to be inconsistent.
- What if the inference rules of the logic are specified incorrectly? It’s easy and common to make mistakes connected with variable capture.
- What if the proof checker has a bug? They are often large and complex pieces of software not developed to high standards of rigour

Who cares?

The robust view:

- Bugs in theorem provers do happen, but are unlikely to produce apparent “proofs” of real results.
- Even the flakiest theorem provers are far more reliable than most human hand proofs.
- Problems in specification and modelling are more likely.
- Nothing is ever 100% certain, and a foundational death spiral adds little value.

We may care

The hawkish view:

- There has been at least one false “proof” of a real result.
- It’s unsatisfactory that we urge formality on others while developing provers so casually.
- It should be beyond reasonable doubt that we do or don’t have a formal proof.
- A quest for perfection is worthy, even if the goal is unattainable.

Prover architecture

The reliability of a theorem prover increases dramatically if its correctness depends only on a small amount of code.

- de Bruijn approach — generate proofs that can be certified by a simple, separate checker.
- LCF approach — reduce all rules to sequences of primitive inferences implemented by a small logical kernel.

The checker or kernel can be much simpler than the prover as a whole.

Nothing is ever certain, but we can potentially achieve very high levels of reliability in this way.

HOL Light

HOL Light is an extreme case of the LCF approach. The entire critical core is 430 lines of code:

- 10 rather simple primitive inference rules
- 2 conservative definitional extension principles
- 3 mathematical axioms (infinity, extensionality, choice)

Everything, even arithmetic on numbers, is done by reduction to the primitive basis.

Automation versus interaction

Most theorem provers can be classified somewhere between two extremes:

- Automatic — User states a conjecture, and the system tries to prove it without further user intervention (e.g. Otter).
- Interactive — User gives an explicit step-by-step proof and the system merely checks its correctness (e.g. AUTOMATH).

Best seems a combination where the user specifies the overall sketch of the proof and the machine fills in the gaps automatically.

Choice of foundations

What kind of logic?

- Classical — easier and more familiar
- Constructive — natural link with computation
- Partial functions — perhaps more intuitive

What kind of mathematical framework?

- Untyped set theory
- Simple type theory
- Rich dependent type theory

Prover architecture

How to organize the construction of the prover?

- Arbitrary programming
- Based on fixed primitive inferences
- Extensible by reflection principles

Coq uses a combination of approaches:

- The set of inference rules is fixed
- Evaluation is optimized for execution inside the logic

Proof style

Directly invoking the primitive or derived rules tends to give proofs that are *procedural*. This can be quite compact and efficient.

But in some ways a *declarative* style (*what* is to be proved, not *how*) is more attractive: easier to understand independent of the prover.

Mizar pioneered the declarative style of proof, and it is now being adopted in some other systems.

There is still no consensus on what is best. Perhaps we need to be able to combine both?

A few notable general-purpose theorem provers

Different systems with various strengths and weaknesses:

- ACL2
- Coq
- HOL Light, HOL4 and ProofPower
- IMPS
- Isabelle
- Mizar
- Nuprl
- PVS

State of the art

Three notable recent proofs:

- Prime Number Theorem — Jeremy Avigad et al (Isabelle/HOL)
- Four-colour theorem — Georges Gonthier (Coq)
- Jordan Curve Theorem — Tom Hales (HOL Light)

These indicate that highly non-trivial results are within reach.
However these all required months/years of work.

Theorem provers and computer algebra systems

Both are systems for symbolic computation, but in practice they are very different:

- CASs are generally easier to use and more efficient
- Theorem provers are more logically flexible and rigorous

Some systems like MathXpert blur the distinction somewhat . . .

Semantics of expressions

Consider an equation $(x^2 - 1)/(x - 1) = x + 1$ from a CAS. What does it mean?

- Universally valid identity (albeit not quite valid)?
- Identity true when both sides are defined
- Identity over the field of rational functions
- . . .

Lack of rigour in many CASs

CASs often apply simplifications even when they are not strictly valid.

Hence they can return wrong results.

Consider the evaluation of this integral in Maple:

$$\int_0^\infty \frac{e^{-(x-1)^2}}{\sqrt{x}}\,dx$$

We try it two different ways:

An integral in Maple

```
> int(exp(-(x-t)^2)/sqrt(x), x=0..infinity);
```

$$\frac{1}{2}\frac{e^{-t^2}\left(-\frac{3(t^2)^{\frac{1}{4}}\pi^{\frac{1}{2}}2^{\frac{1}{2}}e^{\frac{t^2}{2}}K_{\frac{3}{4}}(\frac{t^2}{2})}{t^2}+(t^2)^{\frac{1}{4}}\pi^{\frac{1}{2}}2^{\frac{1}{2}}e^{\frac{t^2}{2}}K_{\frac{7}{4}}(\frac{t^2}{2})\right)}{\pi^{\frac{1}{2}}}$$

```
> subs(t=1,%);
```

$$\frac{1}{2}\frac{e^{-1}\left(-3\pi^{\frac{1}{2}}2^{\frac{1}{2}}e^{\frac{1}{2}}K_{\frac{3}{4}}(\frac{1}{2})+\pi^{\frac{1}{2}}2^{\frac{1}{2}}e^{\frac{1}{2}}K_{\frac{7}{4}}(\frac{1}{2})\right)}{\pi^{\frac{1}{2}}}$$

```
> evalf(%);
```

$$0.4118623312$$

```
> evalf(int(exp(-(x-1)^2)/sqrt(x), x=0..infinity));
```

$$1.973732150$$

Combining theorem provers and computer algebra systems

We can combine CASs and other useful bits of mathematical software with theorem provers.

But how can we gain the power of a CAS without sacrificing logical rigour?

Use the CAS as an oracle but check its “certificates”.

$$\forall a\ b\ c\ x.\ ax^2 + bx + c = 0 \Rightarrow b^2 - 4ac \geq 0$$

can easily be proved by considering the certificate:

$$b^2 - 4ac = (2ax + b)^2 - 4a(ax^2 + bx + c)$$

however the magic identity was arrived at.

Example of external linkage

Here's a combination of HOL Light and PARI/GP in action:

```
# time PRIME_CONV ‘prime 1234567891‘;;
Reading GPRC: /home/knoppix/.gprc ...Done.

proving that 1234567891 is prime
Reading GPRC: /home/knoppix/.gprc ...Done.

Reading GPRC: /home/knoppix/.gprc ...Done.

CPU time (user): 0.09
val it : thm = |- prime 1234567891 <=> T
```

Certificate-producing mathematical software

We'd like computer algebra systems and other mathematical software to produce these certificates when possible.

Example: when a polynomial is in an ideal, give the cofactors or a derivation tree.

It's apparently quite hard to get this information from many computer algebra systems, yet it is very useful.

Conclusions

- Formalization of mathematics is feasible with modern computer technology and sofware.
- Useful in formal verification and arguably in pure mathematics too.
- Still many provers and many different design choices without clear consensus.
- We may be able to exploit other bits of mathematical software without sacrificing rigour.