# Mixing Computations and Proofs

Michael Beeson

July 18, 2014

# What's holding up the QED Singularity?

- The QED Singularity, mentioned in Freek's *Notices* article, is the future time when formal proofs will be the norm in mathematics.
- It is now only a gleam in the eye.
- Most mathematicians take the view that formal mathematics is either not even useful, or is not worth the cost (the time and energy would be better spent proving new theorems informally).
- Contrast this with the near-universal adoption of TEX.
- Evidently the cost-benefit analysis for TEX came out well.
- Imagine QED as an extension (or restriction, in some ways) of the TEX environment, in which a proof error would show up much as a TEX error does now.
- If it's too difficult, people won't use it. There are various ways it could be too difficult.

# Ways QED can be too difficult

- If I have to write too many steps in too much detail.
- If the system doesn't know facts at the undergraduate level.
- Referring to well-known theorems (meaning theorems so well-known that I would not have to cite a reference in a published paper) cannot cost a lot of time to look them up, and the system must know about them.
- If I have to write the proofs in a vastly different way than I would naturally write them in TEX.

All these are problems with all of today's systems.

# Logic and Computation

I will focus today on only one difficulty: the interplay between logic and computation.

- Mathematics consists of logic and computation, interwoven in tapestries of proofs.
- "Computation" refers to chains of formulas progressing towards an "answer", such as one makes when evaluating an integral or solving an equation.
- Typically computational steps move "forwards" (from the known facts further facts are derived) and logical steps move "backwards" (from the goal towards the hypothesis, as in *it would suffice to prove*.
- The mixture of logic and computation gives mathematics a rich structure that has not yet been captured, either in the formal systems of logic, or in computer programs.
- The proper way to mix proofs and computations will have to be found before the QED singularity will arrive.

# Outline

- Context; general remarks (9 slides)
- I will focus on my own contributions, as that is the only thing I know better than the rest of you.
- Symbolic computation with logical correctness in *MathXpert* (program released 1997). (4 slides)
- Reducing logic to computation I: Using infinitesimals in limit computations in*MathXpert*. (1995) (4 slides and a demo).
- Reducing logic to computation II: convergence tests for infinite series in *MathXpert*, and asymptotic inequalities. (4 slides and a demo).
- Linking proof to computation, by calling *MathXpert* from theorem-prover Otter-$\lambda$, with resulting applications to proof by mathematical induction. (2006) (4 slides).
- Theorem prover *Weierstrass* and the proof of irrationality of $e$ (2001). Computations from *MathXpert* combined with Gentzen-style inference rules. The right level of detail in a formal proof. (4 slides and a glimpse of the proof.)

# Kinds of Mathematical Reasoning

Librarians and journal editors are accustomed to classifying mathematics by subject matter, but that is not what we have in mind. Instead, we classify mathematics by the *kind of proofs* that are used:

- Purely logical
- Simple theory, as in geometry (one kind of object, few relations)
- Equational, as in the Robbins problem, or in group or ring theory.
- Uses calculations, as in algebra or calculus
- Uses natural numbers and mathematical induction
- Uses definitions (perhaps lots of them)
- Uses a little number theory and simple set theory (as in undergraduate algebra courses)
- Uses inequalities heavily (as in analysis)

# Obstacles

- Computation software doesn't track assumptions, or doesn't track them completely, and can give erroneous results.
- Computation performed by logic is inefficient.
- Justifying computations requires putting in too many steps.
- Computation performed by unverified software may be unreliable.
- Verified software may be inefficient.

# Several approaches to the problem

- Verify the algorithm, coded in the same language the proof system uses. Then you don't need to verify each result. (Coq does this.)
- Verify *each computation* step by step, rather than the algorithm. (HOL-Light does this, see the tutorial §3.4 for example.)
- Use unverified software, but check the result (not every step). E. g. if you find an indefinite integral, it doesn't matter how you got it, you can check it by differentiation.
- Use unverified software and just believe it. After all your proof-checker may have a bug too.

# Treatment of computations in QED

- When writing formal proofs that are intended to be human-readable, we do not want to see low-level justifications of tiny steps of a calculation.
- Do we want to see answer-only steps like the following?

```
sage: factor(t^119-1)
(t - 1) * (t^6 + t^5 + t^4 + t^3 + t^2 + t + 1)
 * (t^16 + t^15 + t^14 + t^13 + t^12 + t^11
 + t^10 + t^9 + t^8 + t^7 + t^6 + t^5 + t^4 + t^3 + t^2
  * (t^96 - t^95 + t^89 - t^88 + t^82 - t^81 + t^79 - t
  t^75 - t^74 + t^72 - t^71 + t^68 - t^67 + t^65 - t^64
  + t^62 - t^60 + t^58 - t^57 + t^55 - t^53 + t^51 - t^
  + t^48 - t^46 + t^45 - t^43 + t^41 - t^39 + t^38 - t^
  + t^34 - t^32 + t^31 - t^29 + t^28 - t^25 + t^24 - t^
  + t^21 - t^18 + t^17 - t^15 + t^14 - t^8 + t^7 - t +
```

- We almost certainly don't want to see that result verified by multiplying out the factorization and justifying each step from the associative and commutative laws.

# Verified computation, from Gonthier's 4-color proof

The program `check_reducibility` is proved to meet its specification `cf_reducible`. After that, any particular run of the program produces a correct result. For example (quoting the paper)

```
Lemma cfred232 : (cfreducible (Config 11 33 37
           H 2 H 13 Y 5 H 10 H 1 H 1 Y 3 H 11 Y 4 H
           9 H 1 Y 3 H 9 Y 6 Y 1 Y 1 Y 3 Y 1 Y Y 1 Y)).
```

[is proved] in just *two* logical steps, by applying `check_reducible_is_valid` to the concrete configuration above ... even though ... a longhand demonstration would need to go over 20 million cases. Of course the complexity does not disappear altogether Coq 7.3.1 needs an hour to check the validity of this trivial proof.

This is the opposite of seeing too many tiny steps.

# All that information is in there somewhere

As demonstrated by Claudio Sacerdoti Coen in *Declarative Representation of Proof Terms*, who shows how to extract a more human-readable proof with steps from a Matita proof script:

In the following example $H$ labels the fact $(x+y)^2 = x^2 + 2xy + y^2$:

**obtain** $H$

$$\begin{aligned}
(x+y)^2 &= (x+y)(x+y) \\
&= x(x+y) + y(x+y) \ \textbf{by} \ \textit{distributivity} \\
&= x^2 + xy + yx + y^2 \ \textbf{by} \ \textit{distributivity} \\
&= x^2 + 2xy + y^2
\end{aligned}$$

**done**

# MathXpert seen in this context

- MathXpert is computational software designed for education.
- But it keeps track of assumptions and has some simple logic internally.
- Hence (except for possible bugs) it can never derive an incorrect result.
- Hence it is safer to "just believe" than other computational software.
- There is no warranty against bugs (unlike in Coq).
- Still the design is interesting and permits one to explore the interface between proof and computation.

# Computation in a logical context

- Each line of a computation represents the right-hand side of a sequent.
- The left-hand side, which is not written, lists the assumptions in force at the moment.
- Computation rules generate new lines from old, but they have *side conditions*; that is, hypotheses that must be satisfied to apply the rule.
- When we want to apply a rule with a side condition, the algorithm is **infer-refute-assume**.
- We first try to infer the condition from the current assumptions.
- If that fails, we try to refute it. (In which case, the rule cannot be applied.)
- If that too fails, then we assume the required side condition and proceed.

# Infer, refute, assume

- "infer" and "refute" cannot be complete, both for theoretical reasons and because they must be nearly instantaneous.
- So, sometimes we will fail to refute a side condition that actually is false.
- Then the next line would appear false, but it also has generated a false assumption, so we technically have not derived a contradiction.
- Example: Divide $x(x-1)=0$ by $x$, generating the assumption $x \neq 0$. Then we find that the only solution of the equation is $x=1$, which looks wrong, but technically, under the assumption $x \neq 0$ it is OK.
- Of course we can't have that in educational software; so various warnings are generated in *MathXpert*, but that is irrelevant to today's general discussion.

# Treatment of bound variables

Example: determine the domain of (definedness conditions for)

$$\sum_{n=1}^{100} x^n$$

- The bound variable $n$ is an integer.
- $0^0$ is not defined.
- When the expression (tree) is traversed, we assume $n$ is an integer between 1 and 100, while traversing that part of the tree. So $n \neq 0$ and we generate no assumption.
- Note that this simple calculation can't be done in first order logic without a couple of quantifiers!

# Eliminating a quantifier alternation using infinitesimals

- Example problem:

$$\lim_{x \to 0} \frac{\sin x^2}{x} + \frac{1}{x - 1}$$

- Since $x$ is a bound variable, no assumptions involving $x$ are appropriate.
- Traversing the expression tree, when we go inside the limit, we assume $x$ is infinitesimally near the limit point (in this case 0), but not equal to 0.
- Then we can infer that the denominators are not zero.
- The expression is therefore defined.
- Asymptotic inequalities and the dependence of $\delta$ on $\epsilon$ are avoided.

# Example 2 in working with limits

To calculate the derivative of $\sqrt{x}$ from the definition, we consider

$$\lim_{h \to 0} \frac{\sqrt{x+h} - \sqrt{x}}{h}$$

- We need to rewrite $(\sqrt{x+h})^2$ as $x+h$.
- That is only legal when $x + h \geq 0$, but we should not generate an assumption depending on the bound variable $h$.
- The proper assumption to generate is $x > 0$. Note $\sqrt{x}$ is not differentiable at 0.
- How can that be done by a general-purpose algorithm?
- Answer: $x + h \geq 0$ for all infinitesimal $h$ if and only if $x > 0$.
- The implicit universal quantifier over infinitesimals is expressed in the computation rule that rewrites $x + h \geq 0$ as $x > 0$.
- Quantifiers eliminated in favor of computation.

# Infinitesimal elimination and its correctness

- Infinitesimal elimination algorithm eliminates an infinitesimal introduced when entering a limit term.
- Interval semantics: A formula $\phi(\alpha)$ involving a nonstandard variable $\alpha$ means that $\phi(x)$ is true for all $x$ in every sufficiently small punctured neighborhood of a limit point.
- One-sided neighborhoods for one-sided limits.
- Obvious modifications for limits at infinity.
- Infinitesimal elimination algorithm is correct for this semantics.
- At least for one infinitesimal: nested limits have not been treated (and are not in-scope for MathXpert, or for freshman calculus).

# Remarks on limit problems

- Before the infinitesimal elimination algorithm, early *MathXpert* used a form of second-order logic, similar to interval semantics, but coded directly in C.
- The infinitesimal elimination algorithms saved six or seven thousand lines of code.
- Students have great difficulties understanding the semantics of limits. Epsilon-delta semantics was not properly understood by even one of hundreds of students I taught.
- Such understanding as some do achieve is based (in effect) on a direct axiomatization of the undefined notion of limit.
- In other words, they learn certain laws to manipulate limits and work with those computation rules rather than with logic.
- I think mathematicians too prefer to work with computation rules; logic is a last resort.

# Infinite series

- Calculus students are asked to know certain basic series and to be able to use them to sum other series.
- Also, they are supposed to be able to calculate the first few terms of a Taylor series by differentiating the given function.
- Finally, they (are supposed to) learn the classical convergence tests: ratio test, root test, alternating series with decreasing terms test, and possibly some fancier ones.
- Infinite series is the only topic on the American AP Calculus exam that was not supported by MathXpert originally.
- Since June 2013 it is supported (to the first-year college level only).
- In a subsequent demo I will show some examples.

# QED issues about infinite series

- We want to write them down without knowing whether or not they converge (i.e., are defined).
- Many computational operations have complicated side conditions: rearrange order of terms, regroup terms, differentiate or integrate term-by-term, multiply or divide.
- The side conditions often involve two quantifiers: absolute convergence, for example.
- The hypotheses for the convergence tests reduce to asymptotic inequalities.
- Infinite series are widespread in mathematics and the failure to work with them in a natural way will hold up the arrival of the "QED singularity".
- Infinite series is the only topic on the American AP Calculus exam that was not supported by MathXpert originally.

# A Feynman story

When I was an undergraduate at Caltech, Feynman told us not ever to worry about whether an infinite series converges. He said,

*Just add it up!* After you've got the answer, there's plenty of time to worry about whether it converges.

Feynman wouldn't have appreciated the goals of QED.

# A John Harrison story

From his paper on formalizing the analytic proof of the prime number theorem:

$$f(z) = \sum_{n=1}^{\infty} \left( \sum_{p \leq n} \frac{\log p}{p} \right) = \sum_{p} \frac{\log p}{p} \left[ \sum_{n \geq p} \frac{1}{n^z} \right]$$

The implied inequality between these different orders of summation of the double series, simply stated without comment by Newman, also occurs in our proofs, and we needed a 116-line proof script to justify it.

# Infinite series convergence tests

## Asymptotic inequalities are the key

- *Comparison test*. If asymptotically $|b_n| \leq |a_n|$ and $\sum_{n=1}^{\infty} a_n$ converges, then so does $\sum_{n=1}^{\infty} b_n$.
- *Divergence test*. If asymptotically $|b_n| \leq |a_n|$ and $\sum_{n=1}^{\infty} b_n$ diverges, then so does $\sum_{n=1}^{\infty} b_n$.
- *Root test*. If asymptotically $a_n \leq a^n < 1$ then $\sum_{n=1}^{\infty} a_n$ converges.

# Computing with asymptotic inequalities

As $n$ goes to $\infty$:

- $x^n < x^m$ if $n < m$ and $x > 1$.
- $x^m < x^n$ if $n < m$ and $x < 1$.
- $x^k < 2^n$ for any fixed $k$.

# Infinite series in MathXpert

Examples will be run live.

# Mathematical induction

- Say you want to prove the formula for the sum of the first $n$ squares.
- The "proof plan" for mathematical induction reduces this to two algebra problems (base case and induction step).
- The induction step is equational reasoning with an equational assumption.

# Otter-$\lambda$ and MathXpert

- Otter-$\lambda$ added code for (a certain kind of) second-order unification to OTTER.
- With second-order unification, you can give Otter a second-order axiom (schema) like induction, and it can try to find an instance of induction that will work.
- It did well enough at this to do all the interesting examples of proofs by induction that Bundy's provers could do.
- The relevance to QED is that, in order to do simple high-school examples, Otter needed high-school algebra.
- So I linked Otter-$\lambda$ to MathXpert (literally, linked the source codes producing a single executable).
- Otter-$\lambda$ passed MathXpert the assumptions from the proof, and retrieved the result (including any changed assumptions). Unless MathXpert has a bug, this result will be correct, so I just let Otter-$\lambda$ take such steps.

# Results of the Otter-$\lambda$ experiment

- This combination (Otter-$\lambda$ plus MathXpert) can indeed do high-school proofs by induction–a successful proof of concept.
- To mention specific examples, it could prove the formulas for $\sum_{n=1}^{\infty} n$ and $\sum_{n=1}^{\infty} n^2$.

# Relevance of the Otter-$\lambda$ experiment to QED

- The proofs produced had about the desired QED level of detail for the computations.
- However, as resolution proofs, they were not very readable (reading resolution proofs requires some practice).
- They relied on trusting that MathXpert had no relevant bugs.
- But because MathXpert is logically sound, there was no other soundness risk.

# The *Weierstrass* prover and the irrationality of $e$

- The irrationality of $e = \sum \frac{1}{n!}$ is a famous result.
- It is not trivial, but not terribly difficult either.
- It mixes logic and computation very thoroughly.
- I wrote a theorem-prover that used a form of backwards search for a Gentzen sequent-calculus proof, and added new inference rules that incorporated calculation rules.
- The code to implement these rules came from MathXpert.
- After enough tinkering the program could automatically find a proof that $e$ is irrational. (Probably in 2000, the paper was published in 2001.)
- Although the automatic part is not relevant to QED, other aspects of this work are.

# Irrationality of $e$ as a QED example

The proof involves

- inequalities
- bounds on infinite series
- type distinctions (between real and natural numbers)
- simplification of expressions involving factorials
- summing an infinite geometrical series
- proving a formula $2^{n-q-1}(q+1)! \leq n!$ by induction on $n$.

# Reception of Weierstrass's proof that $e$ is irrational

- I gave a couple of mathematics colloquium talks about this proof in 2000 or 2001.
- The questions asked indicated that people were very willing to believe that soon computers would be proving much more complicated theorems, perhaps new ones.
- I am not sure of the reasons for this.

# The proof, as an example of a QED proof with the right level of detail

- It was typeset (in TEX) by *Weierstrass*
- It has a logical structure, shown typographically with indentation when an assumption is made, and a corresponding exdentation (the opposite of indentation) when that assumption is discharged.
- Some steps are justified by "simplification". Not every internal detail of simplification is shown.
- People will perhaps be happy to accept "simplification" as a justification, if they trust the simplifier, and the missing steps are not too many (Is 20 million too many?)