Real Numbers in Real Applications

John Harrison

Intel Corporation

- Real numbers for fun and profit
- The phenomenon of transcendence
- Floating-point verification
- Context of the work
- HOL Light's real and floating-point theories
- Verifying a tangent algorithm
- Conclusions

Mathematics for verification

It's often thought that formal verification requires only "trivial" mathematics.

Much research in the 1970s was focused on automating as much as possible of this trivial mathematics.

However, some important verification applications require non-trivial mathematics.

This might once have been considered surprising, but is no longer particularly controversial.

We'll focus particularly on the role of real analysis in floating-point verifications. But there are other good examples.

Reals for fun

The earliest machine-checked developments of real analysis were not done with practical applications in mind.

- Jutting's formalization of Landau's "Grundlagen" in AUTOMATH
- Articles in the Mizar library by various authors

See also work by Bledsoe on automated proofs in nonstandard analysis.

From Volume 1 of the JFM

Basic Properties of Real Numbers

Real Sequences and Basic Operations on Them

Vectors in Real Linear Space

Subspaces and Cosets of Subspaces in Real Linear Arithmetic

Operations on Subspaces in Real Linear Space

Some Properties of Real Numbers

Monotone Real Sequences. Subsequences

Convergent Real Sequences. Upper and Lower Bound of Sets of Reals

Real Function Spaces

Linear Combinations in Real Linear Space

The Sum and Product of Finite Sequences of Real Numbers

The Lattice of Real Numbers. The Lattice of Real Functions.

Partial Functions from a Domain to the Set of Real Numbers

Topological Properties of Subsets in Real Numbers

Properties of Real Functions

Real Function Continuity

Real Function Uniform Continuity

Real Function Differentiability

Average Value Theorems for Real Functions of One Variable

Basis of Real Linear Space

Reals for verification

Now, the importance of real analysis in verification is widely accepted.

There are good developments of real analysis in at least the following provers:

- HOL
- PVS
- Coq
- ACL2r
- Isabelle

Most of these were developed, in the last decade, with applications in mind.

Indeed, reals were considered so important that ACL2's basic logic was extended to accommodate them.

The phenomenon of transcendence

In general, some applications may need mathematics going well beyond the obvious domain.

Consider an example from mathematics, the prime number theorem, stating that $\pi(n)$, the number of primes $\leq n$, has the limiting property

$$\pi(x)/(\frac{x}{ln(x)}) \rightarrow 1$$

All known proofs of this result use analysis.

Even finding a proof using just *real* analysis was a major accomplishment!

By the way, some deep results about the distribution of primes are used in the recent polynomial-time primality-testing algorithm ...

Floating-point verification

Floating-point arithmetic seems a particularly good target for formal verification:

- It's difficult to find efficient software/compiler workarounds for errors.
- Some of the algorithms are quite intricate and not feasible to verify by traditional simulation.
- Intel has already had a traumatic and expensive ($475M) experience with a floating-point division bug.
- There is a fairly clear and unambiguous specification, e.g. the IEEE 754-1985 Standard for the basic arithmetic operations.

Real reals?

All floating-point numbers are in fact rational. So it might seem that we only need a theory of rationals.

Indeed, there has been some work on formal verification of basic arithmetic operations in some AMD processors using the original ACL2 system, without real numbers.

However, even to specify square root, one of the basic IEEE operations, we're stretching things.

It seems hopeless in practice to specify, let alone verify, transcendental functions like sin, exp and log without real reals.

Context of this work

- We have applied formal verification to a number of algorithms used in the Intel® Itanium® processor family.
- The algorithms are used in hardware (microcode), firmware and software (math libraries and compiler inlining).
- Whatever the underlying implementation, the basic algorithms and the mathematical details involved are the same, and it makes sense to consider them at the algorithmic level.
- Verification covers division, square root and some major transcendental functions
- Division and square root are proved to obey the IEEE specification. Transcendental functions are proved to have an error within a fixed bound (e.g. $0.6ulp$).

Quick introduction to HOL Light

The verifications are conducted using HOL Light, one of the family of theorem provers based on Mike Gordon's original HOL system.

- An LCF-style programmable proof checker written in CAML Light, which also serves as the interaction language.
- Supports classical higher order logic based on polymorphic simply typed lambda-calculus.
- Extremely simple logical core: 10 basic logical inference rules plus 2 definition mechanisms.
- More powerful proof procedures programmed on top, inheriting their reliability from the logical core. Fully programmable by the user.
- Well-developed mathematical theories including basic real analysis.

HOL Light is available for download from:

`http://www.cl.cam.ac.uk/users/jrh/hol-light`

HOL real analysis theory

- Definitional construction of real numbers
- Basic topology
- General limit operations
- Sequences and series
- Limits of real functions
- Differentiation
- Power series and Taylor expansions
- Transcendental functions
- Gauge integration

Examples of useful theorems

```
|- sin(x + y) =
   sin(x) * cos(y) + cos(x) * sin(y)

|- tan(&n * pi) = &0

|- &0 < x /\ &0 < y
   ==> (ln(x / y) = ln(x) - ln(y))

|- f contl x /\ g contl (f x)
   ==> (\x. g(f x)) contl x

|- (!x. a <= x /\ x <= b
        ==> (f diffl (f' x)) x) /\
   f(a) <= K /\ f(b) <= K /\
   (!x. a <= x /\ x <= b /\ (f'(x) = &0)
        ==> f(x) <= K)
   ==> !x. a <= x /\ x <= b ==> f(x) <= K
```

HOL floating point theory

We have formalized a generic floating point theory in HOL, which can be applied to all the required formats, and others supported in software such as quad precision.

A floating point format is identified by a triple of natural numbers `fmt`.

The corresponding set of real numbers is `format(fmt)`, or ignoring the upper limit on the exponent, `iformat(fmt)`.

Floating point rounding returns a floating point approximation to a real number, ignoring upper exponent limits. More precisely

```
round fmt rc x
```

returns the appropriate member of `iformat(fmt)` for an exact value `x`, depending on the rounding mode `rc`, which may be one of `Nearest`, `Down`, `Up` and `Zero`.

The $(1 + \epsilon)$ property

Most of the routine parts of floating point proofs rely on either an absolute or relative bound on the effect of floating point rounding. The key theorem underlying relative error analysis is the following:

```
|- normalizes fmt x /\
   ~(precision fmt = 0)
   ==> ?e. abs(e) <= mu rc /
                  &2 pow (precision fmt - 1) /\
           (round fmt rc x = x * (&1 + e))
```

This says that given that the value being rounded is in the range of normalized floating point numbers, then rounding perturbs the exact result by at most a relative error bound depending only on the floating point precision and rounding control.

Derived rules apply this result to computations in a floating point algorithm automatically, discharging the conditions as they go.

Cancellation theorems

Low-level mathematical algorithms often rely on special tricks to avoid rounding error, or compensate for it. Rounding is trivial when the value being rounded is already representable exactly:

```
|- a IN iformat fmt ==> (round fmt rc a = a)
```

Some special situations where this happens are as follows:

```
|- a IN iformat fmt /\ b IN iformat fmt /\
   a / &2 <= b /\ b <= &2 * a
   ==> (b - a) IN iformat fmt

|- x IN iformat fmt /\
   y IN iformat fmt /\
   abs(x) <= abs(y)
   ==> (round fmt Nearest (x + y) - y)
                  IN iformat fmt /\
       (round fmt Nearest (x + y) - (x + y))
                  IN iformat fmt
```

A tangent algorithm

An algorithm to calculate tangents works essentially as follows.

- The input number X is first reduced to r with approximately $|r| \leq \pi/4$ such that $X = r + N\pi/2$ for some integer N. We now need to calculate $\pm tan(r)$ or $\pm cot(r)$ depending on N modulo 4.
- If the reduced argument r is still not small enough, it is separated into its leading few bits B and the trailing part $x = r - B$, and the overall result computed from $tan(x)$ and pre-stored functions of B, e.g.

$$tan(B + x) = tan(B) + \frac{\frac{1}{sin(B)cos(B)}tan(x)}{cot(B) - tan(x)}$$

- Now a power series approximation is used for $tan(r)$, $cot(r)$ or $tan(x)$ as appropriate.

Overview of the verification

In order to verify this algorithm, we need to prove:

- The range reduction to obtain r is done accurately.
- The mathematical facts used to reconstruct the result from components are applicable.
- The pre-stored constants such as $tan(B)$ are sufficiently accurate.
- The power series approximation does not introduce too much error in approximation.
- The rounding errors involved in computing with floating point arithmetic are within bounds.

Most of these parts are non-trivial. Moreover, some of them require more pure mathematics than might be expected.

Range reduction (1)

Range reduction involves a fairly complicated computation, using various tricks to avoid rounding error. This can mostly be dealt with using the general lemmas given above. However, controlling the errors is harder the smaller the reduced argument is, so we need to answer the key mathematical question:

> How close can a floating point number be to an integer multiple of $\pi/2$?

To answer this question, we need to formalize in HOL some theorems about rational approximations. First of all, we have formalized some results allowing us to (provably) find arbitrarily good rational approximations to π, e.g. the series:

$$\pi = \Sigma_{m=0}^{\infty} \frac{1}{16^m}\left(\frac{4}{8m+1} - \frac{2}{8m+4} - \frac{1}{8m+5} - \frac{1}{8m+6}\right)$$

Range reduction (2)

We then formalize the proof that *convergents* to a real number x (rationals $p_1/q_1 < x < p_2/q_2$ with $p_2 q_1 = p_1 q_2 + 1$) are the best possible approximation with limited denominator.

```
|- (p2 * q1 = p1 * q2 + 1) /\
   (&p1 / &q1 < x /\ x < &p2 / &q2)
   ==> !b. ~(b = 0) /\ b < q1 /\ b < q2
           ==> abs(&a / &b - x)
                 > &1 / &(q1 * q2)
```

We find such convergents (outside the logic) using the Stern-Brocot tree, and by inserting the values into the approximation theorems, and can answer the above question for input numbers in the specified range:

```
|- integer(N) /\ ~(N = &0) /\
   a IN iformat (rformat Register) /\
   abs(a) < &2 pow 64
   ==> abs (a - N * pi / &2)
           >= &113 / &2 pow 76
```

Deriving the cotangent series (1)

The power series for tangent and cotangent are found in many mathematical handbooks. For example (for $0 < |x| < \pi$):

$$cot(x) = 1/x - \frac{1}{3}x - \frac{1}{45}x^3 - \frac{2}{945}x^5 - \ldots$$

However, such handbooks typically don't give any proof, while more rigorous works don't usually discuss such concrete results at all.

It's no accident that the proof we eventually found and formalized is in an older book: Knopp's *"Infinite Series"*. By a rather complicated limit argument we can prove:

$$\pi x\ cot(\pi x) = 1 + 2x^2 \Sigma_{k=1}^{\infty} \frac{1}{x^2 - k^2}$$

Deriving the cotangent series (2)

We can then expand the individual terms of the power series:

$$\frac{-x^2}{x^2 - k^2} = \Sigma_{n=1}^{\infty} (x^2/k^2)^n$$

Since all terms have the same sign, it's fairly easy to show that we can reverse the order of the summations. This gives us a power series with coefficients expressed in terms of the harmonic sums like $1 + 1/2^4 + 1/3^4 + 1/4^4 + \cdots$. By using the fact that $cot(x) - 2cot(2x) = tan(x)$ (for $0 < |x| < \pi/2$), we can compare the coefficients against the derivatives of tan and hence get them as rational numbers. As a byproduct, we derive various well-known theorems like:

$$\begin{aligned} 1 + 1/2^2 + 1/3^2 + 1/4^2 + \cdots &= \pi^2/6 \\ 1 + 1/2^4 + 1/3^4 + 1/4^4 + \cdots &= \pi^4/90 \end{aligned}$$

A Proof from the Book?

The latest edition of "Proofs from the Book" presents an allegedly simple proof of the cotangent expansion based on a trick due to Herglotz.

The key insight is indeed very easy to formalize:

```
|- !h. (!x. h contl x) /\
      (!x. h(x + &1) = h(x)) /\
      (!x. h(--x) = --h(x)) /\
      (!x. h(x / &2) + h((x + &1) / &2) =
           &2 * h(x))
     ==> !x. h(x) = &0`,
```

However, the application to the cotangent series uses some additional tricks, in particular extension by continuity over isolated singularities.

By the time these were made precise, the HOL proof script was almost exactly the same size as the Knopp version!

Error in the actual power series

In fact, the power series for tan and cot used in the algorithm are not quite the standard Taylor/Laurent expansions.

This would not be possible anyway since the coefficients are not all representable exactly as floating-point numbers.

A *minimax* approximation is used, whose coefficients are derived numerically using Remez's algorithm.

This means we need to bound $tan(x) - p(x)$ for an "arbitrary" polynomial $p(x)$.

We start by finding a standard Taylor series $t(x)$ with several more terms, so the difference $tan(x) - t(x)$ is negligible.

This reduces the problem to bounding a polynomial $q(x) = p(x) - t(x)$ over the appropriate interval.

Bounding functions

We have a theorem in HOL that a function attains its extrema either at endpoints of the interval concerned, or at a point of zero derivative:

```
|- (!x. a <= x /\ x <= b ==> (f diffl (f' x)) x) /\
   f(a) <= K /\
   f(b) <= K /\
   (!x. a <= x /\ x <= b /\ (f'(x) = &0) ==> f(x) <= K)
   ==> (!x. a <= x /\ x <= b ==> f(x) <= K)
```

So it suffices to isolate the points of zero derivative quite closely, evaluate the function there and add on an error term to compensate for the fact that we don't generally know the *exact* point of zero derivative:

```
|- (!x. a <= x /\ x <= b ==> (f diffl (f' x)) x) /\
   (!x. a <= x /\ x <= b ==> (f' diffl (f'' x)) x) /\
   (!x. a <= x /\ x <= b ==> abs(f''(x)) <= K) /\
   a <= c /\ c <= x /\ x <= d /\ d <= b /\ (f'(x) = &0)
   ==> abs(f(x)) <= abs(f(d)) + (K / &2) * (d - c) pow 2
```

Root isolation of polynomials

We just need to isolate the zeros of the derivative.

We can accept conservativeness, so we don't need to spend energy eliminating multiple roots etc. The key theorem is that between zeros of $f'(x)$, there can be at most one root of $f(x)$, and there can be none at all if the function doesn't change sign:

```
|- (!x. a <= x /\ x <= b ==> (f diffl f'(x))(x)) /\
   (!x. a < x /\ x < b ==> ~(f'(x) = &0)) /\
   f(a) * f(b) >= &0
   ==> !x. a < x /\ x < b ==> ~(f(x) = &0)

|- (!x. a <= x /\ x <= b ==> (f diffl f'(x))(x)) /\
   (!x. a < x /\ x < b ==> ~(f'(x) = &0))
   ==> !c d. a <= c /\ c <= d /\ d <= b /\
             f(c) * f(d) <= &0
             ==> !x. a < x /\ x < b /\ (f(x) = &0)
                     ==> c <= x /\ x <= d`
```

So we can recursively bound and isolate all the derivatives, starting at the trivial n^{th} derivative. We program a derived rule in HOL that does this automatically.

Conclusions

- Traditionally, real analysis was formalized for general intellectual interest, but now can be used in real applications.
- Quite abstractly, we might expect to observe a phenomenon of "transcendence" in some applications, just as we do in mathematics.
- Concretely, floating-point verification is one important application domain where quite a lot of real analysis is used.
- No doubt future applications will generate the need for more formalized mathematics.
- Or conversely, more formalized mathematics will make possible new applications!