Formal verification of floating point trigonometric functions

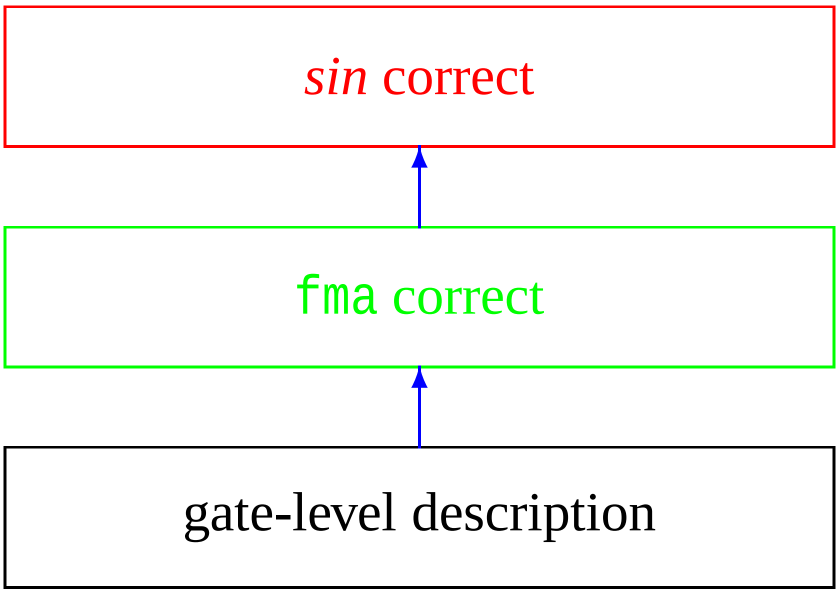
John Harrison

Intel Corporation

- Levels of verification
- HOL Light
- Theories of reals and floating point numbers
- Sine and cosine algorithms
- Mathematics of range reduction
- Other aspects of verification
- Conclusions

Levels of verification

Verifying higher-level floating-point algorithms based on assumed correct behavior of hardware primitives.



We will assume that all the operations used obey the underlying specifications as given in the Architecture Manual and the IEEE Standard for Binary Floating-Point Arithmetic.

This is a typical *specification* for lower-level verification.

Context

Specific work reported here is for the Intel® Itanium™ processor.

Similar work is underway on software libraries for the Intel Pentium® 4 processor.

Floating point algorithms for transcendental functions are used for:

- Software libraries (C `libm` etc.)
- Implementing x86 hardware intrinsics

The level at which the algorithms are modeled is similar in each case.

Infrastructure

What do we need to formally verify such mathematical software?

- Theorems about basic real analysis and properties of the transcendental functions.
- Theorems about special properties of floating point numbers, floating point rounding etc.
- Automation of as much tedious reasoning as possible.
- Ability to write special-purpose inference routines.
- A flexible framework in which these components can be developed and applied in a reliable way.

We use the HOL Light theorem prover. Other possibilities would include PVS and maybe ACL2.

Quick introduction to HOL Light

HOL Light is a member of the family of HOL theorem provers, demonstrated at FMCAD'96.

- 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 and 3 axioms.
- 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`

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

HOL floating point theory

Generic theory, applicable to all required formats (hardware-supported or not).

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 $(1+\varepsilon)$ property

Most routine floating point proofs just use results like 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))
```

Rounded result is true result perturbed by relative error.

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

Cancellation theorems

The present algorithms also rely on a number of low-level tricks.

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
```

Sine/cosine algorithm

Works roughly as follows (details simplified):

- The input number X is first reduced to $r+c$ with $|c| << |r|$ and approximately $|r| \leq \pi/4$ such that $X = r + N\pi/2$ for some integer N. We now need to calculate $\pm sin(r)$ or $\pm cos(r)$ depending on N modulo 4 and whether we want $sin(X)$ or $cos(X)$.

- Main function is evaluated by a fairly long power series, e.g. $sin(r+c) = r + P_1 r^3 + P_2 r^5 + \cdots + P_8 r^{17} + c(1 - r^2/2)$.

- To reduce the effect of rounding error, the second term of all the power series are split into a high part calculated exactly and a low part whose rounding error is less significant. For example $\frac{1}{2}r^2 = \frac{1}{2}r_{hi}^2 + \frac{1}{2}(r + r_{hi})(r - r_{hi})$.

Sources of error

The error in the result can be split into several components:

1. The error from range reduction: $|sin(r) - sin(X - N\frac{\pi}{2})$.
2. The polynomial approximation error $|p(r+c) - sin(r+c)|$.
3. The additional error in neglecting higher powers of c: $|p(r+c) - (p(r) + c(1 - r^2/2))|$.
4. The rounding error in actually computing $p(r) + c(1 - r^2/2))$.

Error analysis

1. Range reduction error requires some non-trivial mathematics to find how small r can be relative to X. If X is too close to a multiple of $\frac{\pi}{2}$, the reduced argument could be inaccurate.
2. Polynomial approximation error is determined automatically. However, it was a fair amount of work to automatically generate formal proofs for results of this kind.
3. Additional approximation error is bounded by straightforward analytical theorems.
4. A lot of the rounding error can be computed automatically. However the tricks used to ensure exact computation need to be proved with human intervention. This applies even more to the range reduction computation.

Example: mathematics of range reduction

We formalize the proof that *convergents* to a real number x, i.e. 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 without having a larger 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
```

Conclusions

- Formal verification of higher-level floating point algorithms is realistic with current theorem-proving technology.
- A large part of the work involves building up general theories about both pure mathematics and special properties of floating point numbers.
- It is easy to underestimate the amount of pure mathematics needed for obtaining very practical results.
- Using HOL Light, we can confidently integrate all the different aspects of the proof, using programmability to automate tedious parts.