High-Level Verification using Theorem Proving and Formalized Mathematics (Extended Abstract)

John Harrison

Intel Corporation, EY2-03
5200 NE Elam Young Parkway
Hillsboro, OR 97124, USA
`johnh@ichips.intel.com`

Abstract. Quite concrete problems in verification can throw up the need for a nontrivial body of formalized mathematics and draw on several special automated proof methods which can be soundly integrated into a general LCF-style theorem prover. We emphasize this point based on our own work on the formal verification in the HOL Light theorem prover of floating point algorithms.

1 Formalized mathematics in verification

Much of our PhD research [11] was devoted to developing formalized mathematics, in particular real analysis, with a view to its practical application in verification, and our current work in formally verifying floating point algorithms shows that this direction of research is quite justified.

First of all, it almost goes without saying that some basic facts about real numbers are useful. Admittedly, floating point verification *has* been successfully done in systems that do not support real numbers at all [16, 17, 19]. After all, floating point numbers in conventional formats are all rational (with denominators always a power of 2). Nevertheless, the whole point of floating point numbers is that they are approximations to reals, and the main standard governing floating point correctness [13] defines behavior in terms of real numbers. Without using real numbers it is already necessary to specify the square root function in an unnatural way, and for more complicated functions such as *sin* it seems hardly feasible to make good progress in specification or verification without using real numbers explicitly.

In fact, one needs a lot more than simple algebraic properties of the reals. Even to define the common transcendental functions and derive useful properties of them requires a reasonable body of analytical results about limits, power series, derivatives etc. In short, one needs a formalized version of a lot of elementary real analysis, an unusual mixture of the general and the special. A typical general result that is useful in verification is the following:

> If a function f is differentiable with derivative f' in an interval $[a, b]$, then a sufficient condition for $f(x) \leq K$ throughout the interval is that $f(x) \leq K$ at the endpoints a, b and at all points of zero derivative.

This theorem is used, for example, in finding a bound for the error incurred in approximating a transcendental function by a truncated power series. The formal HOL version of this theorem looks like this:

```
|- (!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)
```

A typical concrete result is a series expansion for π [1]:

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

This allows us to approximate π arbitrarily closely by rational numbers. Doing so is important both for detailed analysis of trigonometric range reduction (reducing an argument x to a trigonometric function to r where $x = r + N\pi/2$) and to dispose of trivial side-conditions. For example, an algorithm might rely on the fact that $sin(x)$ is positive for some particular x, and we can verify this by confirming that $0 < x < \pi$ using an approximation of π. In HOL, the formal theorem is as follows:

```
|- (\n. inv(&16 pow n) * (&4 / &(8 * n + 1) - &2 / &(8 * n + 4) -
                          &1 / &(8 * n + 5) - &1 / &(8 * n + 6))) sums pi
```

The mathematics needed in floating-point verification is an unusual mixture of these general and special facts, and it's sometimes the kind that isn't widely found in textbooks. For example, an important result we use is the power series expansion for the cotangent function (for $x \neq 0$):

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

To derive this straightforward-looking theorem, both getting a simple recurrence relation for the coefficients *and* a reasonably sharp bound on their size, is fairly non-trivial. A typical mathematics book either doesn't mention such a concrete result at all, or gives it without proof as part of a "cookbook" of well-known useful results. After some time browsing in a library, we eventually settled on formalizing a proof in Knopp's classic book on infinite series [14]. Formalizing this took several days of work, drawing extensively on existing analytical lemmas in HOL. A side-effect is that we derived a general result on harmonic sums, the simplest special cases of which are the well-known:

$$1 + 1/2^2 + 1/3^2 + 1/4^2 + \cdots = \pi^2/6$$

and

$$1 + 1/2^4 + 1/3^4 + 1/4^4 + \cdots = \pi^4/90$$

Knopp remarks

It is not superfluous to realize all that was needed to obtain even the first of these elegant formulae.

We may add that it is even more surprising that such extensive mathematical developments are used simply to verify that a floating point tangent function satisfies a certain error bound. Of course, one also needs plenty of specialized facts about floating point arithmetic, e.g. important properties of rounding. These theories have also been developed in HOL Light [12] but we will not go into more detail here.

2 Proof in HOL Light

The theorem prover we are using in our work is HOL Light [8],[1] a version of the HOL prover [5]. HOL is a descendent of Edinburgh LCF [6] which first defined the 'LCF approach' that these systems take to formal proof. LCF provers explicitly generate proofs in terms of extremely low-level primitive inferences, in order to provide a high level of assurance that the proofs are valid. In HOL Light, as in most other LCF-style provers, the proofs (which can be very large) are not usually stored permanently, but the strict reduction to primitive inferences in maintained by the abstract type system of the interaction and implementation language, which for HOL Light is CAML Light [4, 23]. The primitive inference rules of HOL Light, which implements a simply typed classical higher order logic, are very simple, and will be summarized below.

$$\frac{}{\vdash t = t}\ \texttt{REFL}$$

$$\frac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u}\ \texttt{TRANS}$$

$$\frac{\Gamma \vdash s = t \quad \Delta \vdash u = v}{\Gamma \cup \Delta \vdash s(u) = t(v)}\ \texttt{MK\_COMB}$$

$$\frac{\Gamma \vdash s = t}{\Gamma \vdash (\lambda x.\, s) = (\lambda x.\, t)}\ \texttt{ABS}$$

$$\frac{}{\vdash (\lambda x.\, t)x = t}\ \texttt{BETA}$$

$$\frac{}{\{p\} \vdash p}\ \texttt{ASSUME}$$

$$\frac{\Gamma \vdash p = q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}\ \texttt{EQ\_MP}$$

[1] See `http://www.cl.cam.ac.uk/users/jrh/hol-light/index.html`

$$\frac{\Gamma \vdash p \quad \Delta \vdash q}{(\Gamma - \{q\}) \cup (\Delta - \{p\}) \vdash p = q}\ \texttt{DEDUCT\_ANTISYM\_RULE}$$

$$\frac{\Gamma[x_1, \ldots, x_n] \vdash p[x_1, \ldots, x_n]}{\Gamma[t_1, \ldots, t_n] \vdash p[t_1, \ldots, t_n]}\ \texttt{INST}$$

$$\frac{\Gamma[\alpha_1, \ldots, \alpha_n] \vdash p[\alpha_1, \ldots, \alpha_n]}{\Gamma[\gamma_1, \ldots, \gamma_n] \vdash p[\gamma_1, \ldots, \gamma_n]}\ \texttt{INST\_TYPE}$$

In `MK_COMB`, the types must agree, e.g. $s : \sigma \to \tau$, $t : \sigma \to \tau$, $u : \sigma$ and $v : \sigma$. In `ABS`, we require that x is not a free variable in any of the assumptions Γ. In `ASSUME`, p must be of Boolean type, i.e. a proposition.

All theorems in HOL are deduced using just the above rules, starting from three *axioms*: Extensionality, Choice and Infinity. There are also definitional mechanisms allowing the introduction of new constants and types, but these are easily seen to be logically conservative and thus avoidable in principle.

CAML Light also serves as a programming medium allowing higher-level derived rules (e.g. to automate linear arithmetic, first order logic or reasoning in other special domains) to be programmed as reductions to primitive inferences, so that proofs can be partially automated. This is very useful in practice. In floating point proofs we make extensive use of quite intricate facts of linear arithmetic, such as:

```
|- x <= a /\ y <= b /\
   abs(x - y) < abs(x - a) /\ abs(x - y) < abs(x - b) /\
   (x <= b ==> abs(x - a) <= abs(x - b)) /\
   (y <= a ==> abs(y - b) <= abs(y - a))
   ==> (a = b)
```

Proving these by low-level primitive inferences can be tedious in the extreme, so it is immensely valuable to have the process automated. Similarly, we often use first order automation to avoid tedious low-level reasoning (e.g. chaining together many inequalities) or exploit symmetries via lemmas such as:

```
|- (!x y. P x y = P y x) /\
   (!x y. Q x ==> P x y)
   ==> !x y. Q x \/ Q y ==> P x y
```

Because these are all programmed as reductions to primitive inferences, we have the security of knowing that any errors in the derived rule cannot result in false "theorems" as long as the few primitive rules are sound. This can be especially important in verification of real industrial systems, since an error in a 'proof' can invalidate the entire result.

The basic LCF approach of exploiting traditional automated techniques [3, 15] or high-level methods of proof description [9] by reducing them to primitive inferences in a single core logic seems to us a very fruitful one. Of course, it has an efficiency penalty, but as we argue in [7], it is not usually too severe except in a few special cases. Nevertheless, there is still much more work to be done to make systems like HOL Light really usable by a nonspecialist. In our opinion, the most impressive system for formalizing abstract mathematics is Mizar [18, 22], and importing the strengths of that system into LCF-style provers is a popular topic of research [10, 21, 24, 26].

The first sustained attempt to actually formalize a body of mathematics (concepts and proofs) was *Principia Mathematica* [25]. This successfully derived a body of fundamental mathematics from a small logical system. However, the task of doing so was extraordinarily painstaking, and indeed Russell [20] remarked that his own intellect 'never quite recovered from the strain of writing it'. The correctness theorems we are producing in our work often involve tens or hundreds of millions of applications of primitive inference rules, and build from foundational results about the natural numbers up to nontrivial and highly concrete applied mathematics. Yet using HOL Light, which can bridge the abyss between simple primitive inferences and the demands of real applications, doing so is quite feasible.

References

1. D. Bailey, P. Borwein, and S. Plouffe. On the rapid computation of various polylogarithmic constants. *Mathematics of Computation*, 66:903–913, 1997.
2. Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors. volume 1690 of *Lecture Notes in Computer Science*, Nice, France, 1999. Springer-Verlag.
3. Richard John Boulton. Efficiency in a fully-expansive theorem prover. Technical Report 337, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK, 1993. Author's PhD thesis.
4. Guy Cousineau and Michel Mauny. *The Functional Approach to Programming.* Cambridge University Press, 1998.
5. Michael J. C. Gordon and Thomas F. Melham. *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.
6. Michael J. C. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science.* Springer-Verlag, 1979.
7. John Harrison. Metatheory and reflection in theorem proving: A survey and critique. Technical Report CRC-053, SRI Cambridge, Millers Yard, Cambridge, UK, 1995. Available on the Web as `http://www.cl.cam.ac.uk/users/jrh/papers/reflect.dvi.gz`.
8. John Harrison. HOL Light: A tutorial introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag, 1996.
9. John Harrison. A Mizar mode for HOL. In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 203–220, Turku, Finland, 1996. Springer-Verlag.

10. John Harrison. Proof style. In Eduardo Giménez and Christine Paulin-Mohring, editors, *Types for Proofs and Programs: International Workshop TYPES'96*, volume 1512 of *Lecture Notes in Computer Science*, pages 154–172, Aussois, France, 1996. Springer-Verlag.
11. John Harrison. *Theorem Proving with the Real Numbers.* Springer-Verlag, 1998. Revised version of author's PhD thesis.
12. John Harrison. A machine-checked theory of floating point arithmetic. In Bertot et al. [2], pages 113–130.
13. IEEE. Standard for binary floating point arithmetic. ANSI/IEEE Standard 754-1985, The Institute of Electrical and Electronic Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA, 1985.
14. Konrad Knopp. *Theory and Application of Infinite Series.* Blackie and Son Ltd., 2nd edition, 1951.
15. Ramaya Kumar, Thomas Kropf, and Klaus Schneider. Integrating a first-order automatic prover in the HOL environment. In Myla Archer, Jeffrey J. Joyce, Karl N. Levitt, and Phillip J. Windley, editors, *Proceedings of the 1991 International Workshop on the HOL theorem proving system and its Applications*, pages 170–176, University of California at Davis, Davis CA, USA, 1991. IEEE Computer Society Press.
16. J Strother Moore, Tom Lynch, and Matt Kaufmann. A mechanically checked proof of the correctness of the kernel of the $AMD5_K86$ floating-point division program. *IEEE Transactions on Computers*, 47:913–926, 1998.
17. John O'Leary, Xudong Zhao, Rob Gerth, and Carl-Johan H. Seger. Formally verifying IEEE compliance of floating-point hardware. *Intel Technology Journal*, 1999-Q1:1–14, 1999. Available on the Web as `http://developer.intel.com/technology/itj/q11999/articles/art_5.htm`.
18. Piotr Rudnicki. An overview of the MIZAR project. Available on the Web as `http://web.cs.ualberta.ca/~piotr/Mizar/MizarOverview.ps`, 1992.
19. David Rusinoff. A mechanically checked proof of IEEE compliance of a register-transfer-level specification of the AMD-K7 floating-point multiplication, division, and square root instructions. *LMS Journal of Computation and Mathematics*, 1:148–200, 1998. Available on the Web via `http://www.onr.com/user/russ/david/k7-div-sqrt.html`.
20. Bertrand Russell. *The autobiography of Bertrand Russell.* Allen & Unwin, 1968.
21. Don Syme. Three tactic theorem proving. In Bertot et al. [2], pages 203–220.
22. Andrzej Trybulec. The Mizar-QC/6000 logic information language. *ALLC Bulletin (Association for Literary and Linguistic Computing)*, 6:136–140, 1978.
23. Pierre Weis and Xavier Leroy. *Le langage Caml.* InterEditions, 1993. See also the CAML Web page: `http://pauillac.inria.fr/caml/`.
24. Markus Wenzel. Isar - a generic intepretive approach to readable formal proof documents. In Bertot et al. [2], pages 167–183.
25. Alfred North Whitehead and Bertrand Russell. *Principia Mathematica (3 vols).* Cambridge University Press, 1910.
26. Vincent Zammit. On the implementation of an extensible declarative proof language. In Bertot et al. [2], pages 185–202.