Applications panel introduction

John Harrison
Intel Corporation

Grand Challenge Verification Workshop 2005

Tue 22nd February 2005 (15:30 – 16:30)

Summary

- Hardware or software?
- Model checking or theorem proving?
- Correct small systems or better large systems?
- Academia or Industry?
- Success or failure?
- What I do

Hardware or software?

Formal verification is currently better established in the hardware industry. Several plausible reasons:

- Hardware is designed in a more modular way than most software
- There is more scope for complete automation
- The potential consequences of a hardware error are greater

However, the conceptual gap between hardware and software is no longer so large, given high-level HDLs and various intermediate levels like microcode and PAL code.

Model checking or theorem proving?

The best-established formal verification techniques are the highly automated ones like equivalence checking and model checking.

Where applicable, these are very effective and can be used profitably by non-experts.

In some cases, more general theorem proving is essential either because:

- The number of states is infinite or too large for enumeration to be practical (e.g. cache coherence with many nodes).
- The problem cannot even be specified for a simpler system (e.g. floating-point transcendentals).

Considerable interest in using theorem proving as a ‘glue’ to reliably break down and abstract problems for model checking.

Correct small systems or better large systems?

Traditionally, much emphasis was placed on complete rigorous verification of a relatively simple system — for example the CLI stack.

Recently, there has been more interest in ‘partial’ verification of real complex systems

- Analysis of Windows device drivers using SLAM
- Non-overflow proof for A380 flight control software

Both approaches may have an important place in moving towards reliable large systems.

Academia or Industry?

In academia

- Easier to take time to analyze a problem thoroughly and find a general, elegant solution rather than a ‘quick fix’.
- Easier to collaborate and share, building on work of others

while in industry

- Stimulating drive of real problems, and simply an appreciation of the realities of the design process
- Opportunity to influence existing development processes.

The last can be a source of frustration as well as opportunity.

Success or failure?

There are plenty of verification success stories to brag about.

Nevertheless, formal verification is still a relatively small part of the hardware industry, and a minuscule part of the software industry.

Both academia and industry can do their bit to make formal verification more successful:

- Improve teaching of formal methods in CS curricula; avoid the general de-emphasis of ‘proof’
- Be more open to rational improvements in the development process, and avoid the rush to unmastered complexity.

What I do

I work on formal verification at Intel, almost entirely using the HOL Light theorem prover.

Most of my work has been on the correctness of floating-point algorithms (a sore spot for Intel after the FDIV bug):

- Proving that division and square root algorithms deliver correctly rounded results
- Proving rigorous error bounds for transcendental functions.

Generally quite successful. Many successful verifications, some bugs found, and some more efficient algorithms designed as a result of the formal analysis.

Example: tangent algorithm

- 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 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

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.
- 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.