Heterogeneous Theorem Proving, Certification, and Integrated Automation

John Harrison

Intel Corporation

PxTP and PSATTT workshops, CADE 2011

Wroclaw

1st August 2011 (14:00–15:00)

Table of contents

Do we need to integrate multiple proof tools?

Yes, current applications in both formal verification and the formalization of mathematics most naturally draw on a wide variety of tools.

- Formal verification uses a wide range of tools including SAT and SMT solvers, model checkers and theorem provers
- Some proofs in mathematics use linear programming, nonlinear optimization, computer algebra systems and other more ad hoc algorithms
- May want to combine work done in different theorem provers, e.g. ACL2, Coq, HOL, Isabelle.

Diversity at Intel

Intel is best known as a hardware company, and hardware is still the core of the company's business. However this entails much more:

- Microcode
- Firmware
- Protocols
- Software

A diversity of activities

Intel is best known as a hardware company, and hardware is still the core of the company's business. However this entails much more:

- Microcode
- Firmware
- Protocols
- Software

If the Intel® Software and Services Group (SSG) were split off as a separate company, it would be in the top 10 software companies worldwide.

A diversity of verification problems

This gives rise to a corresponding diversity of verification problems, and of verification solutions.

- Propositional tautology/equivalence checking (FEV)
- Symbolic simulation
- Symbolic trajectory evaluation (STE)
- Temporal logic model checking
- Combined decision procedures (SMT)
- First order automated theorem proving
- Interactive theorem proving

Integrating all these is a challenge!

Layers of verification

If we want to verify from the level of software down to the transistors, then it's useful to identify and specify intermediate layers.

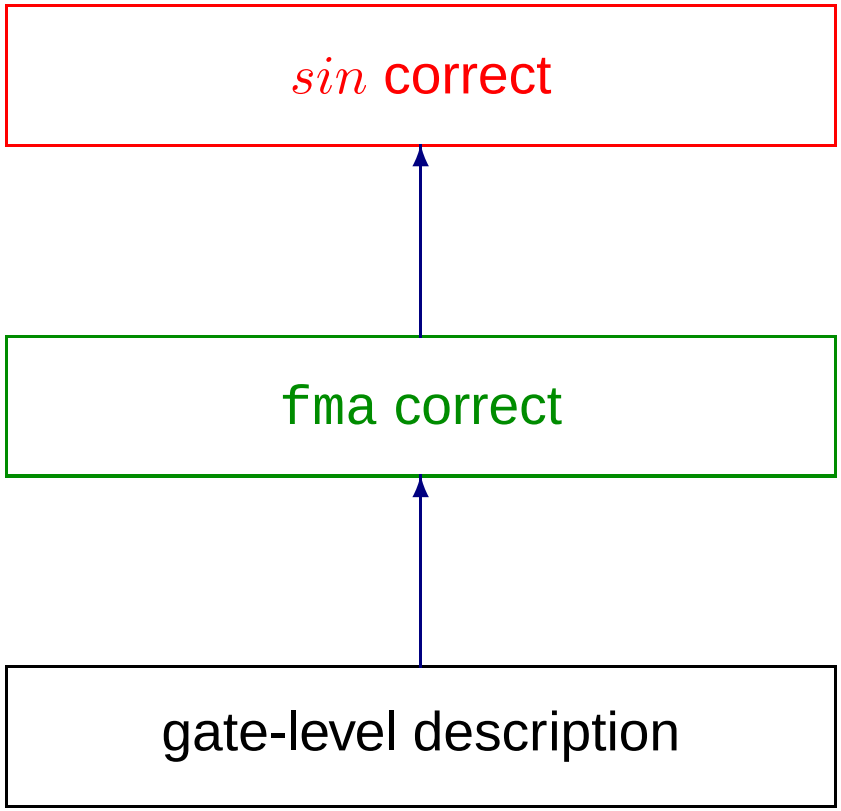
- Implement high-level floating-point algorithm assuming addition works correctly.
- Implement a cache coherence protocol assuming that the abstract protocol ensures coherence.

Many similar ideas all over computing: protocol stack, virtual machines etc.

If this clean separation starts to break down, we may face much worse verification problems. . .

Very often, different tools are better suited to different layers.

Example 1: floating-point algorithms



Example 1: floating-point algorithms

Formal proof of sin function assuming `fma` is correct:

> Harrison, *Formal verification of floating point trigonometric functions*, FMCAD 2000.

Formal proof of `fma` correctness at the gate level:

> Slobodova, *Challenges for Formal Verification in Industrial Setting*, FMCAD 2007.

Yet these verifications were done in different proof systems and do not even share a common `fma` specification.

Example 2: protocol verification

Many successes with Chou-Mannava-Park method for parametrized systems:

Chou, Mannava and Park: *A simple method for parameterized verification of cache coherence protocols*, FMCAD 2004.

Krstic, *Parametrized System Verification with Guard Strengthening and Parameter Abstraction*, AVIS 2005.

Talupur, Krstic, O'Leary and Tuttle, *Parametric Verification of Industrial Strength Cache Coherence Protocols*, DCC 2008.

Bingham, *Automatic non-interference lemmas for parameterized model checking*, FMCAD 2008.

Talupur and Tuttle, *Going with the Flow: Parameterized Verification Using Message Flows*, FMCAD 2008.

Example 2: protocol verification

The CMP method applies to *parametrized systems* with N equivalent replicated components, so the state space involves some Cartesian product

$$\Sigma = \Sigma_0 \times \overbrace{\Sigma_1 \times \cdots \times \Sigma_1}^{N \text{ times}}$$

The method abstracts the system to a finite-state one and then uses a conventional model checker to prove the abstraction.

Currently, the abstraction is done by ad hoc programs, even though it would be desirable to encompass it all in a formal proof system.

Pure mathematics: the Kepler conjecture

The *Kepler conjecture* states that no arrangement of identical balls in ordinary 3-dimensional space has a higher packing density than the obvious ‘cannonball’ arrangement.

Hales, working with Ferguson, arrived at a proof in 1998:

- 300 pages of mathematics: geometry, measure, graph theory and related combinatorics, . . .
- 40,000 lines of supporting computer code: graph enumeration, nonlinear optimization and linear programming.

Hales submitted his proof to *Annals of Mathematics* . . .

The response of the reviewers

After a full four years of deliberation, the reviewers returned:

> “The news from the referees is bad, from my perspective. They have not been able to certify the correctness of the proof, and will not be able to certify it in the future, because they have run out of energy to devote to the problem. This is not what I had hoped for.
>
> Fejes Toth thinks that this situation will occur more and more often in mathematics. He says it is similar to the situation in experimental science — other scientists acting as referees can’t certify the correctness of an experiment, they can only subject the paper to consistency checks. He thinks that the mathematical community will have to get used to this state of affairs.”

The birth of Flyspeck

Hales's proof was eventually published, and no significant error has been found in it. Nevertheless, the verdict is disappointingly lacking in clarity and finality.

As a result of this experience, the journal changed its editorial policy on computer proof so that it will no longer even try to check the correctness of computer code.

Dissatisfied with this state of affairs, Hales initiated a project called *Flyspeck* to completely formalize the proof.

Flyspeck

Flyspeck = ‘Formal Proof of the Kepler Conjecture’.

> “In truth, my motivations for the project are far more complex than a simple hope of removing residual doubt from the minds of few referees. Indeed, I see formal methods as fundamental to the long-term growth of mathematics. (Hales, *The Kepler Conjecture*)

The formalization effort has been running for a few years now with a significant group of people involved, some doing their PhD on Flyspeck-related formalization.

In parallel, Hales has simplified the non-formal proof using ideas from Marchal, significantly cutting down on the formalization work.

Flyspeck: current status

- Almost all the ordinary mathematics has been formalized in HOL Light: Euclidean geometry, measure theory, *hypermaps*, *fans*, results on packings.
- Many of the linear programs have been verified in Isabelle/HOL by Steven Obua. Alexey Solovyev has recently developed a faster HOL Light formalization.
- The graph enumeration process has been verified (and improved in the process) by Tobias Nipkow in Isabelle/HOL
- An approach to formalizing the nonlinear programming based on Bernstein polynomials has been developed by Roland Zumkeller, initially using Coq.

Flyspeck: the challenges

Besides its sheer size, a key challenge of the Flyspeck proof is the use of multiple computer-based methods together with a traditional paper proof.

The Flyspeck project seems to have solved most of these integration problems, but the nonlinear optimization part remains difficult.

Moreover, the successful use of many different proof systems in working on the project has led to new issues around the integration of different interactive theorem provers.

Sharing results or sharing proofs?

A key dichotomy is whether we want to simply:

- Transfer *results*, effectively assuming the soundness of tools
- Transfer *proofs* or other ‘certificates’ and actually check them in a systematic way.

The first is general speaking easier and still useful. The latter gives better assurance and is the approach I, and probably most people here, are interested in.

Matching semantics

Even for the relatively easy case of transferring results, we need a precise match between the semantics of the tools.

In the case of importing a tool in some specific mathematical domain (e.g. an integer programming package) into a general theorem prover, this is usually pretty easy, though there can be subtle corners.

It becomes much more complex and difficult if we want to transfer results between general mathematical frameworks with significantly different foundations.

Interfaces between interactive provers

Transferring results:

- hol90 $\rightarrow$ Nuprl: Howe and Felty 1997
- ACL2 $\rightarrow$ HOL4: Gordon, Hunt, Kaufmann & Reynolds 2006

Transferring proofs:

- HOL4 $\rightarrow$ Isabelle/HOL: Skalberg 2006
- HOL Light $\rightarrow$ Isabelle/HOL: Obua 2006
- Isabelle/HOL $\rightarrow$ HOL Light: McLaughlin 2006
- HOL Light $\rightarrow$ Coq: Keller 2009

More comprehensive solutions for exchange between HOL-like provers include work by Hurd et al. (OpenTheory) and Adams (importing into HOL Zero).

Pure logic: SAT

SAT is particularly important nowadays given the power of modern SAT solvers and the fact that they get used as components in other systems (QBF solvers, bounded model checkers, . . .)

For *satisfiable* problems it's generally easy to get a satisfying valuation out of a SAT solver and check it relatively efficiently.

For *unsatisfiable* problems, some SAT checkers are capable of emitting a resolution proof, and this can be checked.

> Weber and Amjad, *Efficiently Checking Propositional Refutations in HOL Theorem Provers*

This is feasible, though depending on the problem it can still take rather more time to check the solution than the SAT solver took to find it. Usually not too much longer, though.

Pure logic: FOL

In principle, relatively easy: often much faster to check a proof even in a slow prover than to perform the extensive search that led to it.

Even 'internal' automated provers like `MESON` in HOL Light and `blast` in Isabelle have long used a separate search phase.

Main difficulties of interfacing to mainstream ATP systems are:

- Getting a sufficiently explicit proof out of certain provers in the first place. For example, Vampire is generally more powerful than prover9, but it's much easier to get proofs from the latter.
- When formulating a problem in a higher-order polymorphically typed setting, making a suitable reduction to the monomorphic first-order logic supported by most ATPs.

Much more detail in Jasmin Blanchette's talk . . .

Pure logic: QBF

Quantified Boolean formulas are a useful representation for some classes of problem. There have been successful projects to check traces from QBF provers:

- Invalid QBF formulas: Weber 2010
- Valid QBF formulas: Kuncar 2011, Kumar and Weber 2011

While these work, the process of checking incurs a sometimes dramatic slowdown, often several orders of magnitude.

These setups also seem very sensitive to the implementation details of the target prover (e.g. name carrying versus de Bruijn terms).

Arithmetical theories: linear arithmetic

Generally works quite well for universal formulas over $\mathbb{R}$ or $\mathbb{Q}$.

The key is Farkas's Lemma, which implies that for any unsatisfiable set of inequalities, there's a linear combination of them that's 'obviously false' like $1 < 0$.

Alexey Solovyev's highly optimized implementation of this is essential for Flyspeck.

More challenging if we have (i) quantifier alternations, or (ii) non-trivial use of a discrete structures like $\mathbb{Z}$ or $\mathbb{N}$. (Simple tricks like $x < y \to x + 1 \leqslant y$ go some way.)

For example, there are implementations of Cooper's algorithm inside theorem provers, but none that can efficiently check traces from any external tool.

Arithmetical theories: algebraically closed fields

Again, the universal theory is easiest, and this coincides with the universal theory of fields or integral domains (when the characteristic is fixed).

Using the Rabinowitsch trick $p \neq 0 \rightarrow \exists y.\, py - 1 = 0$, we just need to refute a conjunction of equations. Then we can appeal to the Hilbert Nullstellensatz:

The polynomial equations $p_1(\overline{x}) = 0, \ldots, p_k(\overline{x}) = 0$ in an algebraically closed field have *no* common solution iff there are polynomials $q_1(\overline{x}), \ldots, q_k(\overline{x})$ such that the following polynomial identity holds:

$$q_1(\overline{x}) \cdot p_1(\overline{x}) + \cdots + q_k(\overline{x}) \cdot p_k(\overline{x}) = 1$$

Thus we can reduce equation-solving to ideal membership.

Arithmetical theories: ideal membership

One can solve ideal membership problems using various methods, e.g. linear algebra. But the most standard method is Gröbner bases, which are implemented by many computer algebra systems.

Given polynomials $p_1(\overline{x}), \ldots, p_k(\overline{x})$ and $r(x)$, these can return explicit cofactor polynomials $q_k(\overline{x})$ when they exist such that

$$q_1(\overline{x}) \cdot p_1(\overline{x}) + \cdots + q_k(\overline{x}) \cdot p_k(\overline{x}) = r(\overline{x})$$

However, in contrast to Farkas's Lemma, the cofactors are not just numbers and can be huge expressions.

Often more efficient to use HOL Light's simple internal implementation of Gröbner bases than appeal to external tools.

However, can return the cofactors in more efficient forms using shared subterms.

Arithmetical theories: universal theory of reals (1)

There is an analogous way of certifying universal formulas over $\mathbb{R}$ using the Real Nullstellensatz, which involves sums of squares (SOS):

The polynomial equations $p_1(\overline{x}) = 0, \ldots, p_k(\overline{x}) = 0$ in a real closed closed field have *no* common solution iff there are polynomials $q_1(\overline{x})$, $\ldots, q_k(\overline{x})$, $s_1(\overline{x}), \ldots, s_m(\overline{x})$ such that

$$q_1(\overline{x}) \cdot p_1(\overline{x}) + \cdots + q_k(\overline{x}) \cdot p_k(\overline{x}) + s_1(\overline{x})^2 + \cdots + s_m(\overline{x})^2 = -1$$

The similar but more intricate Positivstellensatz generalizes this to inequalities of all kinds.

Arithmetical theories: universal theory of reals (2)

The appropriate certificates can be found in practice via semidefinite programming (SDP). For example
$23x^2 + 6xy + 3y^2 - 20x + 5 = 5 \cdot (2x - 1)^2 + 3 \cdot (x + y)^2 \geqslant 0$ or

$$\forall a\ b\ c\ x.\ ax^2 + bx + c = 0 \Rightarrow b^2 - 4ac \geqslant 0$$

because

$$b^2 - 4ac = (2ax + b)^2 - 4a(ax^2 + bx + c)$$

However, most standard nonlinear solvers do not return such certificates, and this approach does not obviously generalize to formulas with richer quantifier structure.

Other examples

There has been some research on at least the following:

- SMT: seems feasible to combine and generalize methods for SAT and theories. Much current research, some reported at this workshop.
- Explicit-state or BDD-based symbolic model checking: seems hard to separately certify and emulation is slow.
- Computer algebra: some easy case like factorization, indefinite integrals. Others like definite integrals are much harder.

Major research challenge: which algorithms lend themselves to this kind of efficient checking? Which ones seem essentially not to? Some analogies with the class NP.

Fully integrated automation?

Suppose we have many efficient decision procedures implemented by external tools. How can we put them together?

Effectively combination methods like Nelson-Oppen and Shostak solve this problem for quantifier-free theories.

But even mild extensions with quantifiers rapidly become undecidable, such as linear integer arithmetic with one function symbol, when we can characterize squaring:

$$(\forall n. f(-n) = f(n)) \wedge f(0) = 0 \wedge (\forall n. 0 \leqslant n \Rightarrow f(n+1) = f(n)+n+n+1)$$

and then multiplication by $m = n \cdot p \Leftrightarrow (n+p)^2 = n^2 + p^2 + 2m$

Quantifiers + theories

At present, we still seem to need human-driven interactive proof to formulate lemmas that can be solved by automated tools and tie them together.

One of the primary research problems in automated theorem proving is to find a practically effective combination of quantifier and theory reasoning.

We see this being approached from both sides:

- First-order provers are adding theory reasoning (SPASS+T)
- SMT solvers are improving their ability to instantiate quantifiers

Can sometimes exploit types to instantiate quantifiers systematically.

However, there is much active research on other heuristics that often seem to work well in practice.

Conclusions

- There is a real need for combining different proof tools, for applications both in formal verification and pure mathematics
- Effective exchange and checking of proofs between tools seems to be the best way of ensuring soundness and intellectual manageability of such connections.
- Several significant problems still seem hard to treat effectively via a certification, including model checking state enumeration and full quantifier elimination or general nonlinear optimization.
- The final challenge will probably lie in the effective combination of a variety of certified techniques, which broadly involves the combination of quantifier and theory reasoning.