Combination and certification of proof tools

John Harrison

Intel Corporation

30th October 2013 (11:00–12:00)

Summary of talk

- Motivation for combining proof tools
  - Intel verification work
  - The Flyspeck project
- Combining tools and certifying results
  - Sharing results or sharing proofs?
  - Interfaces between interactive provers
  - Primality as a motivating example
- Survey of result certification
  - SAT, FOL, QBF
  - Linear arithmetic
  - Algebraically closed fields
  - Real-closed fields
  - Other possibilities
- Examples
  - Reciprocal algorithm
  - Flyspeck inequality

0: Motivation

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

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
- The Kepler proof uses linear programming, nonlinear optimization, and other more ad hoc algorithms

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
- The Kepler proof uses linear programming, nonlinear optimization, and other more ad hoc algorithms
- Many powerful facilities in computer algebra systems that we'd like to exploit

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
- The Kepler proof uses linear programming, nonlinear optimization, and other more ad hoc algorithms
- Many powerful facilities in computer algebra systems that we'd like to exploit
- 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

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

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.

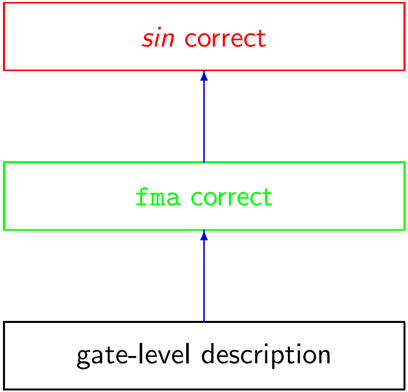
- Implement high-level floating-point algorithm assuming addition works correctly.
- Implement a cache coherence protocol assuming that the abstract protocol ensures coherence.

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: a diversity of methods

The Flyspeck proof combines large amounts of pure mathematics, optimization programs and special-purpose programs:

- Standard mathematics including Euclidean geometry and measure theory
- More specialized theoretical results on *hypermaps*, *fans* and packing.
- Enumeration procedure for 'tame' graphs
- Many linear programming problems.
- Many nonlinear programming problems.

1: Combining tools and certifying results

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 → Nuprl: Howe and Felty 1997
- ACL2 → HOL4: Gordon, Hunt, Kaufmann & Reynolds 2006

Transferring proofs:

- HOL4 → Isabelle/HOL: Skalberg 2006
- HOL Light → Isabelle/HOL: Obua 2006
- Isabelle/HOL → HOL Light: McLaughlin 2006
- HOL Light → Coq: Keller 2009

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

Certificates

We really want the various tools to be able to produce some kind of *certificate* that can be relatively easily checked in the prover.

Certificates

We really want the various tools to be able to produce some kind of *certificate* that can be relatively easily checked in the prover.

- We don't need to bring all the complicated and possibly buggy code in the various external tools into our formal world — we just check their work afterwards!

Certificates

We really want the various tools to be able to produce some kind of *certificate* that can be relatively easily checked in the prover.

- We don't need to bring all the complicated and possibly buggy code in the various external tools into our formal world — we just check their work afterwards!
- Example: suppose we want to prove formally that $2^{32} + 1$ is not prime.

Certificates

We really want the various tools to be able to produce some kind of *certificate* that can be relatively easily checked in the prover.

- We don't need to bring all the complicated and possibly buggy code in the various external tools into our formal world — we just check their work afterwards!
- Example: suppose we want to prove formally that $2^{32} + 1$ is not prime.
- Factorize it using external tools, giving the certificate (in this case just the answer) $2^{32} + 1 = 641 \times 6700417$

Certificates

We really want the various tools to be able to produce some kind of *certificate* that can be relatively easily checked in the prover.

- We don't need to bring all the complicated and possibly buggy code in the various external tools into our formal world — we just check their work afterwards!
- Example: suppose we want to prove formally that $2^{32} + 1$ is not prime.
- Factorize it using external tools, giving the certificate (in this case just the answer) $2^{32} + 1 = 641 \times 6700417$
- Factoring large numbers uses highly complex algorithms and optimized code, but to check the answer we just need to do simple integer arithmetic.

Proving primality

What about the dual problem of proving that a large number *is* prime? It's not so obvious how to certify this.

Proving primality

What about the dual problem of proving that a large number *is* prime? It's not so obvious how to certify this.

- There are suitable certificates that p is prime, based on a factorization of $p-1$, using Lucas's theorem from number theory.

Proving primality

What about the dual problem of proving that a large number *is* prime? It's not so obvious how to certify this.

- There are suitable certificates that p is prime, based on a factorization of $p - 1$, using Lucas's theorem from number theory.
- Pratt, "Every prime has a succinct certificate", SIAM J. Computing 1975. This was the first proof that primality is NP (we now know it's in P).

Proving primality

What about the dual problem of proving that a large number *is* prime? It's not so obvious how to certify this.

- There are suitable certificates that p is prime, based on a factorization of $p - 1$, using Lucas's theorem from number theory.
- Pratt, "Every prime has a succinct certificate", SIAM J. Computing 1975. This was the first proof that primality is NP (we now know it's in P).
- A somewhat more efficient refinement using Pocklington's theorem was implemented in Coq by Caprotti and Oostdijk, "Formal and efficient primality proofs by computer algebra oracles"

Pocklington's thoerem

In HOL Light, we also generate a 'certificate of primality' based on Pocklington's theorem:

```
2 ≤ n ∧
(n - 1 = q * r) ∧
n ≤ q EXP 2 ∧
(a EXP (n - 1) == 1) (mod n) ∧
(∀p. prime(p) ∧ p divides q ⇒ coprime(a EXP ((n - 1) DIV p) - 1,n))
⇒ prime(n)
```

The certificate is generated 'extra-logically', using the factorizations produced by PARI/GP.

The certificate is then checked by formal proof, using the above theorem.

2: Survey of result certification

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 $x < y \to x + 1 \leq y$ 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 $23x^2 + 6xy + 3y^2 - 20x + 5 = 5 \cdot (2x - 1)^2 + 3 \cdot (x + y)^2 \geq 0$ or

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

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.

3: Examples

Results on reciprocal algorithm

We use prime factor certification to derive critical values that need to be checked for the correctness of a reciprocal algorithm:

0xFFFFFFFFFFFFFFFF	0xFFFFFFFFFFFFFFFD	0xFE421D63446A3B34	0xFBFC17DFE0BEFF04	0xFB940B119826E598
0xFB0089D7241D10FC	0xFA0BF7D05FBE82FC	0xF912590F016D6D04	0xF774DD7F912E1F54	0xF7444DFBF7B20EAC
0xF39EB657E24734AC	0xF36EE790DE069D54	0xF286AD7943D79434	0xEDF09CCC53942014	0xEC4B058D0F7155BC
0xEC1CA6DB6D7BD444	0xE775FF856986AE74	0xE5CB972E5CB972E4	0xE58469F0234F72C4	0xE511C4648E2332C4
0xE3FC771FE3B8FF1C	0xE318DE3C8E6370E4	0xE23B9711DCB88EE4	0xE159BE4A8763011C	0xDF738B7CF7F482E4
0xDEE256F712B7B894	0xDEE24908EDB7B894	0xDE86505A77F81B25	0xDE03D5F96C8A976C	0xDDFF059997C451E5
0xDB73060F0C3B6170	0xDB6DB6DB6DB6DB6C	0xDB6DA92492B6DB6C	0xDA92B6A4ADA92B6C	0xD9986492DD18DB7C
0xD72F32D1C0CC4094	0xD6329033D6329033	0xD5A004AE261AB3DC	0xD4D43A30F2645D7C	0xD33131D2408C6084
0xD23F53B88EADABB4	0xCCCE6669999CCCD0	0xCCCE666666633330	0xCCCCCCCCCCCCCCD0	0xCBC489A1DBB2F124
0xCB21076817350724	0xCAF92AC7A6F19EDC	0xC9A8364D41B26A0C	0xC687D6343EB1A1F4	0xC54EDD8E76EC6764
0xC4EC4EC362762764	0xC3FCF61FE7B0FF3C	0xC3FCE9E018B0FF3C	0xC344F8A627C53D74	0xC27B1613D8B09EC4
0xC27B09EC27B09EC4	0xC07756F170EAFBEC	0xBDF3CD1B9E68E8D4	0xBD5EAF57ABD5EAF4	0xBCA1AF286BCA1AF4
0xB9B501C68DD6D90C	0xB880B72F050B57FC	0xB85C824924643204	0xB7C8928A28749804	0xB7A481C71C43DDFC
0xB7938C6947D97303	0xB38A7755BB835F24	0xB152958A94AC54A4	0xAFF5757FABABFD5C	0xAF4D99ADFEFCAAFC
0xAF2B32F270835F04	0xAE235074CF5BAE64	0xAE0866F90799F954	0xADCC548E46756E64	0xAD5AB56AD5AB56AC
0xAD5AAA952AAB56AC	0xAB55AAD56AB55AAC	0xAAAAB55555AAAAAC	0xAAAAAAAAAAAAAAAC	0xAAAAA00000555554
0xA93CFF3E629F347D	0xA80555402AAA0154	0xA8054ABFD5AA0154	0xA7F94913CA4893D4	0xA62E84F95819C3BC
0xA5889F09A0152C44	0xA4E75446CA6A1A44	0xA442B4F8DCDEF5BC	0xA27E096B503396EE	0x9E9B8FFFFFD8591C
0x9E9B8B0B23A7A6E4	0x9E7C6B0C1CA79F1C	0x9DFC78A4EEEE4DCB	0x9C15954988E121AB	0x9A585968B4F4D2C4
0x99D0C486A0FAD481	0x99B831EEE01FB16C	0x990C8B8926172254	0x990825E0CD75297C	0x989E556CADAC2D7F
0x97DAD92107E19484	0x9756156041DBBA94	0x95C4C0A72F501BDC	0x94E1AE991B4B4EB4	0x949DE0B0664FD224
0x942755353AA9A094	0x9349AE0703CB65B4	0x92B6A4ADA92B6A4C	0x9101187A01C04E4C	0x907056B6E018E1B4
0x8F808E79E77A99C4	0x8F64655555317C3C	0x8E988B8B3BA3A624	0x8E05E117D9E786D5	0x8BEB067D130382A4
0x8B679E2B7FB0532C	0x887C8B2B1F1081C4	0x8858CCDCA9E0F6C4	0x881BB1CAB40AE884	0x87715550DCDE29E4
0x875BDE4FE977C1EC	0x86F71861FDF38714	0x85DBEE9FB93EA864	0x8542A9A4D2ABD5EC	0x8542A150A8542A14
0x84BDA12F684BDA14	0x83AB6A090756D410	0x83AB6A06F8A92BF0	0x83A7B5D13DAE81B4	0x8365F2672F9341B4
0x8331C0CFE9341614	0x82A5F5692FAB4154	0x8140A05028140A04	0x8042251A9D6EF7FC	

Results on a Flyspeck inequality

Some simple Flyspeck inequalities, after being expressed componentwise, can be proved efficiently by SOS certification, e.g. this one in HOL Light syntax:

```
!u v w:real^3.dist(u,v) >= &2 /\
      dist(u,w) >= &2 /\
      dist(v,w) >= &2 /\
      norm(u - v) < sqrt(&8)
      ==> norm(w - &1 / &2 % (u + v))
          > norm(u - v) / &2
```

Results on a Flyspeck inequality

Some simple Flyspeck inequalities, after being expressed componentwise, can be proved efficiently by SOS certification, e.g. this one in HOL Light syntax:

```
!u v w:real^3.dist(u,v) >= &2 /\
     dist(u,w) >= &2 /\
     dist(v,w) >= &2 /\
     norm(u - v) < sqrt(&8)
     ==> norm(w - &1 / &2 % (u + v))
         > norm(u - v) / &2
```

However, some of the more complex ones seem to be out of reach of current SOS implementations.

Conclusions

- There is a real need for combining different proof tools, for applications both in formal verification and pure mathematics

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.

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.