Taking Theorem Proving Mainstream

John Harrison
Intel Corporation

Logic Colloquium 2006

Nijmegen

29 July 2006

Status of theorem proving

Theorem proving is quite widely used for correctness checking:

- Verifying correctness of computer systems
- Formalizing proofs in mathematics

In neither community is it yet a ‘mainstream’ activity.

Benefits and costs

Formalization in a theorem prover offers two main benefits:

- Confidence in correctness (if theorem prover is sound).
- Automatic assistance with tedious/routine parts of proof.

However, formalization and theorem proving is hard work, even for a specialist.

Current niches

We currently see use of theorem proving where:

- The cost of error is too high, e.g. $475M for the floating-point bug in the Intel®Pentium® processor.
- There are genuine doubts in the community about the correctness of a proof, e.g. Hales's proof of the Kepler Conjecture.

How might theorem proving expand beyond this niche?

Full automation?

Computers can occasionally prove interesting results automatically:

- SAM’s lemma
- Robbins conjecture

For most proofs however, we need an interactive combination of human and theorem prover.

Interactive theorem proving

The idea of a more ‘interactive’ approach was already anticipated by pioneers, e.g. Wang (1960):

> [...] the writer believes that perhaps machines may more quickly become of practical use in mathematical research, not by proving new theorems, but by formalizing and checking outlines of proofs, say, from textbooks to detailed formalizations more rigorous that *Principia* [Mathematica], from technical papers to textbooks, or from abstracts to technical papers.

However, constructing an effective combination is not so easy.

Effective interactive theorem proving

What makes a good interactive theorem prover? Most agree on:

- Library of existing results
- Intuitive input format
- Powerful automated steps

Several other characteristics are more controversial:

- Programmability
- Checkability of proofs

Library of existing results (1)

Most mathematical proofs rely on a certain amount of “machinery”. Duplicating all this for each proof is impractical.

Most theorem provers have a library of pre-proved mathematics; perhaps the most impressive is the MML.

However, sometimes unsystematic approach to library development.

Work is split among several different provers.

Library of existing results (2)

Possible ways forward:

- Consolidation round fewer theorem provers / logics
- Transfer of proofs at some level between systems
- More systematic approach to library development

Some of these things are already happening.

Intuitive input format (1)

We probably don’t want completely informal ‘natural language’ input, but what do we want?

- Declarative proof — emphasis on *what* is to be proved
- Procedural proof — emphasis on *how* it is to be proved

Each approach has its merits, and enthusiastic advocates.

Intuitive input format (2)

Ways forward:

- Analysis and refinement of declarative and procedural styles
- Support of multiple proof styles
- More feedback from “regular users”

Powerful automated steps (1)

Many powerful automated methods are known:

- General first-order proof search (tableaux, resolution, . . .)
- Decision procedures for theories (Presburger or Tarski arithmetic)

Several problems:

- Too general and not efficient enough for practical problems
- Complex algorithms have questionable soundness
- Too monolithic, difficult to combine and use as subroutines

Automated procedures are too general

Very often the problems arising in practice fall into limited subsets:

- The linear inequality reasoning in program verification often involves just 'UTVPI' cases: $ax \leq by + c$ for $a, b \in \{-1, 0, 1\}$.
- Much nonlinear inequality reasoning involves just proving universally quantified formulas, not general quantifier elimination

By focusing on these cases we may be able to find algorithms that are much more efficient or have other desirable characteristics . . .

Automated procedures may not be correct

Two solutions are

- Formally prove the correctness of the code (“reflection”)
- Arrange proof as a separation of search and checking

Example of latter: to prove

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

observe that

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

Automated procedures are too monolithic (1)

Consider proving "word problems" for rings, e.g.

$$\forall x_1, \ldots, x_n . p_1(x_1, \ldots, x_n) = 0 \wedge \cdots \wedge p_k(x_1, \ldots, x_n) = 0 \Rightarrow p(x_1, \ldots, x_n) = 0$$

This holds precisely if $p \in Id < p_1, \ldots, p_n >$ over $\mathbb{Z}[x_1, \ldots, x_n]$.

An efficient algorithm is to find cofactors for ideal membership (e.g. using integer Gröbner bases):

$$p(x_1, \ldots, x_n) = p_1(x_1, \ldots, x_n) \cdot q_1(x_1, \ldots, x_n) + \cdots p_k(x_1, \ldots, x_n) \cdot q_k(x_1, \ldots, x_n)$$

Automated procedures are too monolithic (2)

Two typical problems:

- Off-the-shelf Gröbner basis packages make it hard to get the cofactors — need to ‘roll our own’.
- Embeddings in theorem provers hide all the ideal membership reasoning

There are some interesting applications for which we want the cofactors directly.

Automated proofs of divisibility properties

Systematic approach to proofs of divisbility properties over integers:

- Take problem involving divisibility concepts, e.g.
  $ax \equiv ay \ (\mathsf{mod}\ n) \wedge \mathsf{coprime}(a, n) \Rightarrow x \equiv y \ (\mathsf{mod}\ n)$
- Expand definitions, e.g.
  $(\exists u.\ ax - ay = nu) \wedge (\exists v\ w.\ av + nw = 1) \Rightarrow \exists z.\ x - y = nz$
- Normalize
  $\forall u\ v\ w.\ ax - ay - nu = 0 \wedge av + nw - 1 = 0 \Rightarrow \exists z.\ x - y = nz$
- Find a witness for existential quantifier by cofactors for
  $x - y \in Id < ax - ay - nu, av + nw - 1, n >$
- Prove
  $\forall u\ v\ w. ax - ay - nu = 0 \wedge av + nw - 1 = 0 \Rightarrow n(uv + xw - yw) = x - y$

Conclusions

Theorem proving may be close to achieving “critical mass”, when it starts to become widely usable.

- More attention to library development and sharing
- More work on input formats
- More work on decision procedures
  - Study of important special cases
  - Study of finding/checking separation
  - Emphasis on non-monolithic code