First Order Logic in Practice

John Harrison

University of Cambridge

http://www.cl.cam.ac.uk/users/jrh/

- Background: interaction and automation
- Why do we need first order automation?
- First order automation for richer logics
- Which problems arise in practice?
- Do the existing methods work?
- Final remarks

The spectrum of theorem provers

AUT MATH (de Bruijn)

Stanford LC (Milner)

Mizar (Trybulec)

. . .

. . .

VS (wre, Rushby, Shankar)

. . .

. . .

SETHE (Letz et al.)

tter (McCune)

Interaction plus Automation

It's a very natural idea for interactive theorem provers to include automation for filling in the intermediate steps.

The idea goes back at least to the SAM (semi-automated mathematics) project in the late 60s.

Nowadays many of the leading interactive systems include automation. There are many different aspects of reasoning that may be automated, e.g.

- Pure logic (first/higher order with/without equality)
- Linear arithmetic (or nonlinear arithmetic)
- Algebraic simplification
- Rewriting, completion and other equality reasoning
- Inductive proofs

What kind of automation?

Different interactive systems tend to focus on some of these in particular, because they are considered more important and/or easier to implement. For example:

- Isabelle — mainly automation of logical and equality reasoning. No decision procedures for arithmetic.
- PVS — decision procedures for important theories such as linear arithmetic, tightly coupled using congruence closure. Minimal support for pure logic.
- HOL — automation for logical and equality reasoning and linear arithmetic, as well as Boyer-Moore style automation of induction proofs. But minimal integration of these different provers.

Which are really the most important?

Logical v theory reasoning (1)

The simple answer is that all of these can be important, some more than others, depending on the application. Different applications might include:

1. Formalizing abstract algebra (e.g. general results about commutative rings)
2. Formalizing more concrete mathematics (e.g. particular Taylor expansions)
3. Verifying abstract system models (e.g. security protocols)
4. Verifying concrete system models (e.g. floating point arithmetic)

For example, logical reasoning is typically more important for (1) and (3), algebraic simplification for (2) and linear arithmetic for (4). Of course, these are just vague general rules.

Logical v theory reasoning (2)

But we can in general say that automating theory reasoning is more important. Why?

- Explicit proofs of, say, facts of linear arithmetic (e.g. $|x - y| \geq ||x| - |y||$) tend to be almost unbearably dull and tedious.
- The logical reasoning in an argument is usually relatively interesting, and fairly simple.

Our own recent work bears this out — we use both logical and theory reasoning but would much prefer to give up the former than the latter.

Why, then, should we be interested in logical automation? Well, even if it's not the *most* useful form, it is still useful. But there is a deeper reason why logical automation is particularly significant.

A Declarative Proof Style

We have said that the logical structures of typical theorems are reasonably simple and interesting. However sometimes the precise choreographing of logical steps is quite tedious when one theorem 'obviously' follows from a given set of premisses.

Mizar allows the user merely to state the premisses, and finds the proof itself, using an optimized special case of tableaux as well as simple techniques for equality reasoning.

This opens up the possibility of stating proofs in a much less prescriptive and more *declarative* style, which arguably leads to a number of advantages in readability, maintainability and indeed writability.

The same advantages can be had in many other interactive systems, given adequate logical automation.

Richer logics

Many of the leading interactive systems like HOL and PVS are based on a higher-order logic.

It would seem that we need to automate higher order logic, as in Andrews's system TPS, not first order logic.

Ideally yes, but (empirically) first order automation is sufficient for many of the problems that arise in practice, using the well-known mechanical reduction of higher order to first order logic.

First order logic has the advantage that there are well engineered 'off-the-shelf' techniques (and systems) to handle it.

HOL to FOL

There are some significant choices in the reduction of higher order to first order logic.

- How to deal with higher order features such as lambda abstractions. A translation of $P[\lambda\, x.\, t[x]]$ to $\forall f.\, (\forall x.\, f(x) = t[x]) \Rightarrow P[f]$?
- How to cope with the polymorphic types used in several higher order theorem provers. Preserve the type information or throw it away? How do we ensure soundness?
- How to reduce the problem to the normal form required by the first order prover. For example, there are many different ways of splitting up the problem into subproblems.
- How to handle equality reasoning, which is very important in practice. Naive equality axioms? Brand's transformation? Paramodulation in the first order prover?

Practical Problems

Traditionally, first order provers have been used for elegant examples in relatively simple axiomatic systems. Often the set of axioms, and even their formulation, is picked very carefully.

The current test suites for first order provers, e.g. TPTP, tend to reflect this bias.

The problems we need to solve in our work tend to be different. They are sometimes (not always) shallow, but involve relatively big and intricate terms, and large amounts of irrelevant information.

We suggest compiling a new list of problems from real applications of first order reasoning. It would be possible to do this semi-automatically.

We have already compiled a list of a few hundred examples from our own work. Preparing a TPTP-style public test suite would be quite possible, or adding them to the new FOF suite.

Do existing methods work?

But there would be little point in making different test suites unless they demanded significantly different qualities in a prover.

There is one obvious difference: we want to solve routine problems *quickly*, rather than very hard problems in hours or days.

Moreover, our problems may test the sensitivity of systems to very large terms, even when those terms are irrelevant to the proof, and the ability to discriminate among a large database of axioms.

Systematic testing of different systems on our problems would be interesting, but we haven't done this yet. We use a version of MESON (see CADE-13 paper).

One interesting point has come to light: we find that on average, naive equality axioms are better than Brand's transformation. Apparently on more standard test problems, the opposite is true.

Final remarks

- When there are well-established methods for handling a class of problems, e.g. first order theorem provers, model checkers, computer algebra systems and linear programming tools, it's always worth reflecting on the potential for using them as subsystems of interactive provers.

- Often the 'interactive' and 'first order automation' communities communicate too little. Interactive provers can provide real applications in which to put first order automation to work, and automation can be the key to some interesting new approaches to interactive proof such as a declarative proof style. If we try to create test suites of more 'practical' problems, we can still compare systems in a meaningful way.