First Order Logic in Practice

John Harrison
University of Cambridge Computer Laboratory
New Museums Site, Pembroke Street
Cambridge CB2 3QG, England
jrh@cl.cam.ac.uk
http://www.cl.cam.ac.uk/users/jrh

Abstract

There is a trend away from monolithic automated theorem provers towards using automation as a tool in support of interactive proof. We believe this is a fruitful drawing together of threads in automated reasoning. But it raises a number of issues that are often neglected in the classical first order theorem proving literature such as the following. Is first order automation actually useful, and if so, why? How can it be used for richer logics? What are the characteristic examples that require solution in practice? How do the traditional algorithms perform on these 'practical' examples — are they deficient or are they already too powerful? We discuss these and similar questions in the light of our own recent experience in this area using HOL [3].

1 Why do we need first order automation?

A taxonomy of computer theorem proving, after perhaps separating 'AI' and 'logic' approaches, would divide most contemporary systems into *automated* and *interactive* provers. Automatic provers (e.g. Otter, SETHEO, TPS) are typically applied to challenging but transparently stated problems in mathematics, and they can at times achieve striking success, as with McCune's recent solution of the 'Robbins conjecture' using EQP. Interactive provers (e.g. HOL, Isabelle, NQTHM/ACL2 and PVS) are normally used to build up, under human guidance, a body of formalized mathematics or a large system verification.[1]

Now, although interactive provers may require manual guidance, it's desirable to provide quite high levels of automation so that the user avoids the tedious filling in of trivial details. Indeed, the most effective recent systems such as PVS do provide quite powerful automation for special theories felt to be particularly important in practice, e.g. linear arithmetic and propositional tautology checking. But what about the automation of pure, typically first order, logic? There have been attempts since at least SAM [4] to harness automation of pure logic in interactive systems. Yet a common view today is that automation of theories like linear arithmetic is far more significant in practice.

There is some justification for this view. The logical structure of a typical verification or mathematics proof is sufficiently simple that users are content to create such a proof manually, whereas proving facts of (say) linear arithmetic is much less interesting, and

[1] We classify systems like NQTHM as interactive provers here because although they need no user selection of proof methods, they do require a careful manual grading of the theorems to be proved in their typical applications.

more time-consuming. For example, a manual proof that $|x - y| \leq ||x| - |y||$ is likely to consist of a series of case splits followed by tedious chaining together of inequalities.

However, we believe that first order automation *is* useful in practice, even in verification applications [8]. Indeed, we have relied quite extensively on first order automation to finish off otherwise tedious subgoals in recent work, e.g. in floating point verification [5] and classic metatheorems for embedded logics (as yet unpublished). It's especially handy when after a large case split there are many boring subgoals — a few would be acceptable to prove by hand, but a dozen or a hundred much less so. Admittedly, the verification example would have been all right without first order automation, whereas it would have been much more tedious without linear arithmetic. Nevertheless, even if first order automation isn't the *most* important tool, it is still useful.

This is not a novel idea: users of many systems such as EVES and Isabelle tend to use first order automation quite extensively. But we believe there is a deeper reason why the automation of pure logic is important, indicated by the Mizar system [10]. This is an interactive theorem prover that has been used for the formalization of an unparallelled amount of pure mathematics.[2] It has several interesting features, but we want to focus on its use of first order automation to provide a *declarative style of proof.*

We have said that the logical structures of typical theorems are reasonably simple and not uninteresting. 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 writeability — see [6] for a more detailed discussion of these points. The same advantages can be had in many other interactive systems, given adequate logical automation. For example, Syme [9] has recently used a prover with a declarative input language to prove type soundness for a part of Java.

2 First order automation for richer logics

It may be objected that, while Mizar is more or less based on a first order theory (Tarski-Grothendieck set theory), many of the leading interactive systems like HOL and PVS are based on a higher-order logic. So it might seem that special higher-order proof-automation methods are essential. Such methods exist, and are used for example in TPS [1]. Nevertheless, first order automation is at present better understood and easy to make efficient, and we believe it is satisfactory for a significant fragment of HOL proofs. We'll now try to explain why.

In HOL, although higher order *features* are constantly used, many of the *proofs* are 'essentially first order'. We reduce higher order to first order logic in a well-known mechanical way: introduce a single binary function symbol a to represent 'application', and translate HOL's $f\ x$ into $a(f, x)$, etc.[3] Then it is often the case (empirically) that when a theorem is provable in higher order logic, the corresponding first order assertion is also provable. Proofs that cannot be done in the first order reduction are those that require the *instantiation* of higher order variables, i.e. the invention of lambda-abstractions. For

[2] See the Web page `http://web.cs.ualberta.ca:80/~piotr/Mizar/`.

[3] Actually we optimize this in various ways when we do it in HOL.

example, when trying to prove $\forall n.\, n + 0 = n$ by induction, the induction theorem needs to be specialized to the relation $\lambda n.\, n + 0 = n$, or equivalently, to the set $\{n \mid n + 0 = n\}$. Note however that if the appropriate term is already bound to some function, then just throwing in the definition is enough; the lambda-term is then expressible in a first order way. However, such a reduction of HOL to FOL allows a number of interesting questions.

- How much effort should we expend in eliminating higher order features? For example, although we reduce beta-redexes in the input, we don't make a more elaborate translation of $P[\lambda x.\, t[x]]$ to $\forall f.\, (\forall x.\, f(x) = t[x]) \Rightarrow P[f]$. Should we?
- HOL features instantiable polymorphic types. At present our translation simply throws away all type information, and regards differently-typed instances of the same constant as different when translating to FOL. To further cut down on the (already rare) cases where this makes the input assertion unprovable, we include some heuristics in a preprocessor to instantiate certain hypotheses with relevant-looking types.[4] An alternative, recently used by Paulson in Isabelle, is to regard all constants with the same name as identical, and backtrack in the rare cases where this causes the eventual HOL proof to fail due to ill-typed terms. Finally, of course, one can translate to many-sorted FOL and use the types during proof search.
- Typically, the eventual first order provers use negation normal form or even (for resolution and model elimination) clausal form. Shoehorning the initial problem into these forms can be done in many different ways. In particular, it is often the case that by splitting bi-implications $p \equiv q$ appropriately (either $(p \wedge \neg q) \vee (\neg p \wedge q)$ or $(p \vee q) \wedge (\neg p \vee \neg q)$ depending on sign) it is often possible to split the problem into much simpler problems, rather than reduce everything to a single clausal form directly.[5] For example, 'Andrews' Challenge':

$$\begin{array}{l}((\exists x.\, \forall y.\, Px \equiv Py) \equiv ((\exists x.\, Qx) \equiv (\forall y.\, Qy))) \\ \equiv ((\exists x.\, \forall y.\, Qx \equiv Qy) \equiv ((\exists x.\, Px) \equiv (\forall y.\, Py)))\end{array}$$

  can be split into 32 independent subgoals, each of which is fairly easy.
- In practice, first order automation seems much less useful without equality handling. Complete approaches include throwing in all the equality 'axioms', including congruence rules for all the relevant function and predicate symbols, and versions of Brand's transformation. There are also more sophisticated complete techniques as well as ad hoc variants like doing rewriting at certain places in standard procedures.

3 Which problems arise in practice?

Any selection of problems we make from our own work will suffer from self-selection. For example, the problems for which we have used first order automation are of course those that our own MESON-based tools could solve acceptably quickly. We haven't kept track of the problems we wished were solvable in a reasonable time but weren't. Nevertheless, we claim that, since we have used first order automation in a fairly wide range of applications, we have a sample of problems that would arise quite often in real work.

[4] We are not sure whether there is any complete preprocessing method — ours is not complete.

[5] It is also possible to use more sophisticated 'definitional' techniques to avoid blowup in such cases.

Typical first order test suites, notably the large TPTP library, tend to be quite different in character. The examples are typically results in quite simple axiomatic systems, e.g. that a group where $\forall x. x^2 = 1$ is Abelian. Compared with our problems they are sometimes easier, often much more difficult, but in any case tend to involve much smaller terms and very little irrelevant information. (There are exceptions, e.g. a substantial number of the TPTP problems in the `NUM` domain are based on a set of about 250 axioms for set theory.) Moreover, they have already been put into clausal form without exploiting possibilities for splitting (in cases where the canonical problem statement isn't already in clausal form).

Therefore, we feel that a suite of examples compiled from work such as our own would be a valuable complement to the TPTP library. It would represent the kinds of problems that arise when using first order automation in a workaday capacity, rather than for isolated *tours de force*. The hope would be not so much that a system *can* solve them, but that it can solve them quickly enough to form part of a convenient interactive environment. If the problems seem too easy, it would be possible given certain proof systems (e.g. Mizar) to make them more difficult by automatically combining multiple steps into a single challenge problem.

4 Do the existing methods work?

We've generally found that a version of the MESON procedure (with a naive method for equality handling based on adding all the equality axioms) is a very useful tool, and in practice solves most of the problems that we would expect. It can even be too powerful, in that some results are proved automatically which perhaps deserve some thought from the user, and a human-style record of the proof.

However on our examples, we cannot always concur with some accepted wisdom. For example, it is usually held that Brand's transformation is a better way of coping with equality than our own naive approach. (A refinement of this method is used by SETHEO.) However our experiments indicate that while both have their strengths and weaknesses, our method actually solves more problems in a reasonable time. We conjecture that this is because in our examples some of the underlying terms, even where they are irrelevant to the proof, are quite large, and so Brand's transformation leads to an unnecessary explosion of clauses.

Of course, the ideal would be to combine first order automation and ubiquitous forms of theory reasoning such as linear arithmetic. This has long been a popular line of research — [2] describes recent work motivated very much by practical problems thrown up during interactive proof. But in any case, we have found that more-or-less standard first order automation is surprisingly valuable in itself.

5 A broader view

1. 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. This is hardly a new idea, but the point deserves emphasis. The main novelty (or handicap, depending on one's point of view) of our own approach is that we maintain soundness by creating all proofs using standard natural deduction steps. Quite often this can be done by having standard

off-the-shelf system record a solution or a proof trace which is then translated into a natural deduction proof. For example [7] discusses combining HOL with Maple in this fashion. It is not necessary to keep the entire system small and simple to ensure soundness.

2. We consider that 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. We must admit that for those interested in automation, solving hard problems is more exciting and helps to stimulate a competitive spirit. But if we try to create test suites of more 'practical' problems, we can still compare systems in a meaningful way. As well as the differences in the kind of problems that are significant, this may change the very qualities one looks for in a first order prover.

References

[1] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16:321–353, 1996.

[2] N. S. Bjørner, M. E. Stickel, and T. Uribe. A practical integration of first-order reasoning and decision procedures. In W. McCune, editor, *Automated Deduction — CADE-14*, volume 1249 of *Lecture Notes in Computer Science*, pages 101–115. Springer-Verlag.

[3] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[4] J. R. Guard, F. C. Oglesby, J. H. Bennett, and L. G. Settle. Semi-automated mathematics. *Journal of the ACM*, 16:49–62, 1969.

[5] J. Harrison. Floating point verification in HOL Light: The exponential function. Technical Report 428, University of Cambridge Computer Laboratory, 1997.

[6] J. Harrison. Proof style. Technical Report 410, University of Cambridge Computer Laboratory, 1997. To appear in the proceedings of TYPES'96.

[7] J. Harrison and L. Théry. A sceptic's approach to combining HOL and Maple. *Journal of Automated Reasoning*, To appear.

[8] R. Kumar, T. Kropf, and K. Schneider. Integrating a first-order automatic prover in the HOL environment. In M. Archer, J. J. Joyce, K. N. Levitt, and P. J. Windley, editors, *Proceedings of the 1991 International Workshop on the HOL theorem proving system and its Applications*, pages 170–176. IEEE Computer Society Press, 1991.

[9] D. Syme. Proving Java type soundness. Technical Report 427, University of Cambridge Computer Laboratory, 1997.

[10] A. Trybulec. The Mizar-QC/6000 logic information language. *ALLC Bulletin (Association for Literary and Linguistic Computing)*, 6:136–140, 1978.