A Mizar Mode for HOL

John Harrison

Åbo Akademi University, Department of Computer Science
Lemminkäisenkatu 14a, 20520 Turku, Finland

Abstract. The HOL theorem prover is implemented in the LCF manner. All inference is ultimately reduced to a collection of very simple (forward) primitive inference rules, but by programming it is possible to build alternative means of proving theorems on top, while preserving security. Existing HOL proofs styles are, however, very different from those used in textbooks. Here we describe the addition of another style, inspired by Mizar. We believe the resulting system combines the secure extensibility and interactivity of HOL with Mizar’s readability and lack of logical prescriptiveness. Part of our work involves adding new facilities to HOL for first order automation, since this allows HOL to be more flexible, as Mizar is, over the precise logical connection between steps.

1 HOL

The HOL theorem prover [13] is descended from Edinburgh LCF. While adding features of its own like stress on definitional extension as a reliable means of theory development, it remains true to the basic idea of the LCF project. This is to reduce all reasoning to a few simple primitive (usually forward) inference rules, but to allow a full programming language to automate higher level ‘derived rules’, broken down into these primitives. For example, HOL includes derived rules for linear arithmetic, tautologies and inductive definitions. Ordinary users can simply invoke them without understanding their implementation, but because they do ultimately decompose to simple primitives, can feel confident in their correctness. Should they need other, perhaps application-specific, proof procedures in the course of their work, they can write them using the same methodology.

This combination of reliability and flexibility is the outstanding feature of LCF systems, and there is usually not a serious loss of efficiency in derived rules [17]. Some of these derived rules may present the user with a quite different view of theorem proving from that implemented in the logical core. Even in the original LCF publication [14] we find the following:

The emphasis of the present project has been on discovering how to exploit the flexibility of the metalanguage to organise and structure the performance of proofs. The separation of the logic from its metalanguage is a crucial feature of this; different methodologies for performing proofs in the logic correspond to different programming styles in the metalanguage. Since our current research concerns experiments with proof methodologies – for example, forward proof versus goal-directed proof – it is essential that the system does not commit us to any fixed style.
To some extent, this theoretical flexibility is already a practical reality in HOL. In addition to the basic ‘machine code’ of forward primitive rules, there are several supported proof styles, all of which fit together smoothly:

- There are numerous more complicated forward proof rules, which can make the business of theorem proving much more palatable than it would be using the primitives. However, before each inference rule is applied, it’s necessary to muster all the required hypotheses exactly, and either include their proofs verbatim, or bind them to names and use those. It’s very hard to do proofs in this way unless the exact structure of the proof is already planned before starting to type.

- Backward, tactical proof was one of the most influential ideas in the LCF project. Most large HOL proofs are done in this way, perhaps because the required hypotheses appear naturally and determine the proof structure automatically. It also allows more convenient use of local assumptions and choosing variables. This flexibility is further increased if the tactic mechanism allows ‘metavarniables’ whose instantiations can be delayed [32, 26].

- Equational reasoning is one of the most widely used parts of the HOL system, largely thanks to an elegant and flexible implementation [25]. Depth conversions and rewriting tools allow the convenient iterated instantiation and use of equations. There are also straightforward means of handling associative and commutative operators.

- Window inference [29] is a methodology for organizing localized proof efforts. Users may focus on a particular subterm or subformula and transform it, exploiting contextual information that comes from its position in the whole formula. For example, when transforming \( \psi \) into an equivalent formula \( \psi' \) in the expression \( \phi \land \psi \), we may assume \( \phi \). Grundy [15] both mechanized window inference in HOL and generalized it to arbitrary preorder relations, such as implication and the refinement relation on programs.

- Prasetya [27] has written a package to support two features of textbook proofs: the use of a series of lemmas, and the use of iterated equations (we shall have more to say about this latter issue later).

- Specialized decision procedures for various particular domains such as linear arithmetic [6] are also available, as well as a number of derived definitional mechanisms [23, 24].

Most of these styles suffer from being rather low-level, making explicit too many details that are normally elided. More precisely, they are too logically prescriptive, demanding that even the most obvious steps be mediated by the exactly appropriate logical rule(s). This isn’t just a problem because doing it is tedious. A beginner might well simply not be able to drive the system well enough to get it to do the requisite steps. For example, many HOL users find manipulation of assumptions difficult. Decision procedures, on the other hand, are perhaps too high-level, compressing into one line substantial mathematical detail.

Whether too high-level or too low-level, all the proof styles suffer from one common failing: the proofs are expressed using complicated combinations of arcane higher order functions in a computer programming language. Though it’s easy to guess what many of