Special Issue Editorial

Higher Order Logic Theorem Proving and its Applications

The past twenty years have seen many advances in the use of computers to do mathematical and logical proofs. Computing scientists and engineers are increasingly discovering the utility of this work—especially in areas like Formal Methods or Semantics, where formal specification or formal reasoning is necessarily involved. Indeed, it is clear that the only feasible way of doing proofs about certain realistic applications, where there may be many thousands of deduction steps, is by computer.

There are several broad approaches to mechanized theorem-proving. In the algorithmic tradition, a computer program is employed to determine the truth of a proposition automatically, using a mathematically-justified decision procedure or perhaps heuristics. In the more interactive tradition, the user participates in proof discovery and construction. Most actual theorem proving systems lie somewhere between these two extremes, providing both interactive and automatic ways of generating proofs.

The topic of this special issue is applications of a theorem-proving technology that lies towards the more interactive end of the spectrum—namely, machine-assisted reasoning using higher order logic and the LCF approach to theorem proving. Higher order logic is a form of typed predicate calculus that is well suited to mechanization (it can be based on the λ-calculus) as well as powerful enough to make available the results of general mathematics. The LCF approach is one in which the system's command language is a strongly-typed functional programming language and an abstract data type of theorems is used to distinguish formulas that have been proved from arbitrary propositions [2].

Interaction with an LCF-style system can take the form of forward proof (applying inference rules to previously proved theorems until the desired theorem is obtained) or backward proof (working backwards from the statement to be proved to previously proved theorems that imply it). But because the system is built on top of a programming language, the user can also write arbitrarily complex programs to implement proof strategies—including fully automatic ones. This extensibility, combined with security and an expressive logic, makes these systems exceptionally well-suited to tailoring by users to an almost unlimited range of applications.

This special issue is intended to give the reader a sample of the range of possible applications of mechanized theorem proving using higher order logic. The papers chosen for this issue are all concerned with applications of one theorem-prover in the LCF tradition—the HOL system [1]. But the contents will also give the reader a good idea of higher order logic theorem proving in general, since there is much here that could be done equally well with similar existing systems.

A prominent application of theorem proving, with considerable scope for practical utility, is *hardware verification*—the use of formal reasoning to increase confidence in the correctness of hardware designs. Techniques for post-design proofs of correctness, at least for certain kinds of hardware, are both well-established and accessibly publicised. The first two papers selected for this issue therefore treat the related but rather less well-understood topics of proof maintenance and correct-by-construction designs. The third paper then presents some new methods for reasoning about RISC processor pipelines—designs for which the usual formal verification technology is inadequate.

In the first paper in this issue, *Tracking Design Changes with Formal Machine-Checked Proof*, Curzon presents a case study on the maintainability of proofs of correctness when designs are modified. Curzon's experience suggests that, with the right approach, it is possible both to verify the correctness of 'real-world' designs and to maintain the proofs (including input scripts for a theorem-prover) in times commensurate with the actual design, implementation and modification times.

Larsson's paper *An Engineering Approach to Formal Digital System Design* describes an approach to correct-by-construction hardware design based on correctness-preserving transformations. Larsson employs Grundy's HOL implementation of the 'window inference' method of interactive proof, which nicely encompasses both the forward and backward proof styles [3].

The third paper, *Formal Specification and Verification Techniques for RISC Pipeline Conflicts* by Tahar and Kumar, describes a method for proving that conflicts do not occur in the instruction pipelines of RISC machines. This work shows how the basic theories for hardware verification by mechanized deduction have now progressed to a stage where one can begin to develop very general theories for specific problem domains, with a view to making verifications within these domains almost routine.

The next three papers are concerned with mechanized formal reasoning about programming languages and programs. An enormously influential contribution to this field has been Milner's LCF project of the late 1970s and early 1980s [2,4]. LCF was designed for in-

teractive reasoning about higher-order recursively defined functions. Agerholm's paper *LCF Examples in HOL* shows how the essentially domain-theoretic basis of LCF can be built up within the HOL system, whose logic is based on set theory rather than domains. The result is a system that allows one to mix both domain and set-theoretic reasoning in a single framework.

Homeier and Martin's paper *A Mechanically Verified Verification Condition Generator* addresses the subject of proving programs correct using an algorithm that reduces a proof to a set of formulas called verification conditions, which (if true) are sufficient to establish correctness of the original program. Homeier and Martin show how to use the HOL theorem prover to formalize the semantics of a programming language, derive proof rules for it, and then prove the correctness of a verification condition generator for the language.

In *Studying the ML Module System in HOL*, Gunter and Maharaj use HOL to reason about the semantics of modules in the Standard ML programming language [5]. The authors go beyond merely encoding the existing dynamic semantics of modules by formalizing an extended semantics for higher-order functors. The HOL mechanization is used to prove theorems that inspire confidence in the reasonableness of this extension.

With the seventh paper, Chou's *Mechanical Verification of Distributed Algorithms in Higher-Order Logic*, the focus of this issue is shifted from hardware and software to distributed algorithms, which involve asynchrony and nondeterminism. Chou presents a framework for reasoning about distributed algorithms comprising a number of formal theories mechanized in HOL. The utility of this infrastructure is demonstrated by the verification of a simple but typical distributed algorithm.

As with all LCF-style systems, HOL is a 'fully-expansive' theorem prover, in the sense that theorems may be produced only by applications of primitive inferences rules. Harrison's paper *Binary Decision Diagrams as a HOL Derived Rule* is one of the few significant investigations into whether automatic decision procedures—in this case, based on BDDs—can be incorporated into this strictly fully-expansive world. Harrison's results are encouraging; he has devised a BDD algorithm that also does primitive inference in HOL and is only a constant factor slower than a direct implementation.

In the final paper, *Representing Higher-order Logic Proofs in HOL*, von Wright employs a theorem prover to reason about the properties of its own logic. Using the HOL system, the author formalizes in higher order logic the concepts of proof and provability for higher order logic itself. This gives a mechanized framework in which one can reason about soundness of derived inference rules, or the correctness of a stand-alone proof checker.

The papers selected for this issue do not, of course, cover all the possible applications of higher order logic theorem proving. Much less do they cover all existing theorem provers. Only one theorem proving system is represented, but there are numerous other systems for various different formulations of higher order logic. All of these have their own special strengths and weaknesses, though in basic facilities offered there is much overlap as well. I have concentrated on HOL because it is the most widely used and well-established system of this kind.

I thank both the authors and the referees for their contributions to this special issue. The referees and authors all worked to a very tight timetable, and their cooperation enabled virtually all correspondence and handling of papers to be done by electronic means.

T. F. Melham
University of Glasgow

REFERENCES

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

[2] M. J. Gordon, A. J. Milner, and C. P. Wadsworth, *Edinburgh LCF: A Mechanised Logic of Computation*, Lecture Notes in Computer Science, vol. 78 (Springer-Verlag, 1979).

[3] J. Grundy, 'Window Inference in the HOL System', in Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications, Davis, August 1991, edited by M. Archer, J. J. Joyce, K. N. Levitt, and P. J. Windley (IEEE Computer Society Press, 1992), pp. 177–189.

[4] R. Milner, 'The use of machines to assist in rigorous proof', in *Mathematical Logic and Programming Languages*, edited by C. A. R. Hoare and J. C. Shepherdson, Prentice-Hall International Series in Computer Science (Prentice-Hall, 1985), pp. 77–88.

[5] R. Milner, M. Tofte, and R. Harper, *The Definition of Standard ML* (MIT Press, 1990).