An LCF-Style Interface between HOL and First-Order Logic

Joe Hurd[⋆]

Computer Laboratory
University of Cambridge,
joe.hurd@cl.cam.ac.uk

1 Introduction

Performing interactive proof in the HOL theorem prover[1] [3] involves reducing goals to simpler subgoals. It turns out that many of these subgoals can be efficiently 'finished off' by an automatic first-order prover. To fill this niche, Harrison implemented a version of the MESON procedure [4] with the ability to translate proofs to higher-order logic. This was integrated as a HOL tactic in 1996, and has since become a standard workhorse of interactive proof. Today, building all the theories in the most recent distribution of HOL relies on MESON to prove 1726 subgoals.

Given this level of demand for automatic first-order proof by users performing interactive proof in HOL, it seems worthwhile to look for ways to narrow the gap between these two worlds. Consider the following high-level view of how a HOL goal g is proved using a first-order prover:

1. We first convert the negation of g to CNF; this results in a HOL theorem of the form
$$\vdash \neg g \iff \exists \boldsymbol{a}.\ (\forall \boldsymbol{v_1}.\ c_1) \wedge \cdots \wedge (\forall \boldsymbol{v_n}.\ c_n) \tag{1}$$
where each c_i is a HOL term having the form of a disjunction of literals, and may contain variables from the vectors $\boldsymbol{a}$ and $\boldsymbol{v_i}$.
2. Next, we create skolem constants for each variable in $\boldsymbol{a}$, and map each HOL term c_i to first-order logic. This produces the clause set
$$C = \{C_1, \ldots, C_n\}$$
3. The first-order prover runs on C, and finds a refutation ρ.
4. By proof translation, the refutation ρ is lifted to a HOL proof of the theorem
$$\{(\forall \boldsymbol{v_1}.\ c_1), \ldots, (\forall \boldsymbol{v_n}.\ c_n)\} \vdash \bot \tag{2}$$
5. Finally, some HOL primitive inferences use theorems (1) and (2) to derive
$$\vdash g \tag{3}$$

⋆ Supported by EPSRC project GR/R27105/01

1 HOL is available at `http://www.cl.cam.ac.uk/Research/HVG/FTP/`.

Various logical incompatibilities manifest themselves in steps 2 and 4, when formulas and proofs must be mirrored in both logics. In this paper we present a generic interface between HOL and first-order logic, offering:

– an expressive representation of HOL terms in unsorted first-order logic, permitting many 'higher-order' goals to be proved by standard first-order deduction calculi;
– an automatic conversion from first-order refutations to HOL proofs, reducing the effort needed to integrate existing first-order provers with the HOL theorem prover;
– with a strong guarantee that soundness will not be violated.

2 Mapping HOL Terms to First-Order Logic

Seemingly the hardest problem with mapping HOL terms to first-order logic—dealing with λ-abstractions—can be smoothly dealt with as part of the conversion to CNF. Any λ-abstraction at or beneath the literal level is rewritten to combinatory form, using the set of combinators $\{S, K, I, C, \circ\}$.[2]

The mapping that we use makes explicit function application, so that the HOL term $m + n$ maps to the first-order term $@(@(+, m), n)$. Since in HOL there is no distinction between terms and formulas, we model this in first-order logic by defining a special relation called B (short for Boolean) that converts a first-order term to a first-order formula. For example, the HOL boolean term $m \leq n$ can be translated to the first-order formula $B(@(@(\leq, m), n))$. The only exception to this rule is equality: the HOL term $x = y$ can be mapped to the first-order logic formula $=(x, y)$.

The mapping described thus far includes no type information, but is still a useful way to map HOL terms to first-order logic. We also experimented with including types in the first-order representation of a HOL term. Using this idea, the HOL term $m + n$ would map to the first-order term

$$@(@(+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}, m : \mathbb{N}) : \mathbb{N} \to \mathbb{N}, n) : \mathbb{N}$$

where ':' is a binary function symbol (written infix for readability), and higher-order logic types are encoded as first-order terms.[3] As might be expected, this mapping produces much larger first-order clauses than omitting the types, and this results in first-order deduction steps taking longer to perform. However, we cannot conclude that including types is definitely harmful: the extra information may pay for itself by cutting down the search space.

[2] In principle we could use more combinators to guarantee a more compact translation, but HOL goals are normally small enough that this extra complication is not worth the effort.

[3] Encoding type variables as first-order logic variables allows polymorphic types to be dealt with in a straightforward manner.

Using this mapping, we can use a first-order prover to prove several 'higher-order' goals, such as the classic derivation of an identity function from combinator theory:

$$\vdash (\forall x, y.\ \mathsf{K}\ x\ y = x) \land (\forall f, g, x.\ \mathsf{S}\ f\ g\ x = (f\ x)\ (g\ x)) \Rightarrow \exists f.\ \forall x.\ f\ x = x$$

3 Translating First-Order Refutations to HOL

At first sight it may appear that the necessity of translating first-order refutations to higher-order logic proofs imposes a burden that hampers free experimentation with the first-order provers. However, by applying the technology of the LCF project [2], we can isolate the proof translation and make it invisible to the developer of first-order proof procedures. We have implemented this automatic proof translation for the mapping that preserves type information, and it has been successfully used in combination with ML versions of first-order calculi to prove many subgoals in the HOL theorem prover.

$$\frac{}{A_1 \lor \cdots \lor A_n}\,\mathsf{AXIOM}\ [A_1, \ldots, A_n] \qquad \frac{}{L \lor \neg L}\,\mathsf{ASSUME}\ L$$

$$\frac{A_1 \lor \cdots \lor A_n}{A_1[\sigma] \lor \cdots \lor A_n[\sigma]}\,\mathsf{INST}\ \sigma \qquad \frac{A_1 \lor \cdots \lor A_n}{A_{i_1} \lor \cdots \lor A_{i_m}}\,\mathsf{FACTOR}$$

$$\frac{A_1 \lor \cdots \lor L \lor \cdots \lor A_m \qquad B_1 \lor \cdots \lor \neg L \lor \cdots \lor B_n}{A_1 \lor \cdots \lor A_m \lor B_1 \lor \cdots \lor B_n}\,\mathsf{RESOLVE}\ L$$

Fig. 1. The Primitive Rules of Inference of Clausal First-Order Logic.

This is achieved by defining a logical kernel of ML functions that execute a primitive set of deduction rules on first-order clauses. For our purposes, we use the five rules in Figure 1, which form a (refutation) complete proof system for clausal first-order logic.

The AXIOM rule is used to create a new axiom of the logical system; it takes as argument the list of literals in the axiom clause. The ASSUME rule takes a literal L and returns the theorem $L \lor \neg L$.[4] The INST rule takes a substitution σ and a theorem A, and applies the substitution to every literal in A.[5] The FACTOR rule takes a theorem and removes duplicate literals in the clause: note that no variable instantiation takes place here, two literals must be identical for one to be removed. Finally, the RESOLVE rule takes a literal L and two theorems A, B, and creates a theorem containing every literal except L from A and every literal except $\neg L$ from B. Again, no variable instantiation takes place here: only literals identical to L in A (or $\neg L$ in B) are removed.

[4] This rule is used to keep track of reductions in the model elimination procedure.

[5] In some presentations of logic, this uniform instantiation of variables in a theorem is called specialization.

```
signature Kernel =
sig
  type formula = Term.formula
  type subst   = Term.subst

  (* An ABSTRACT type for theorems *)
  eqtype thm

  (* Destruction of theorems is fine *)
  val dest_thm : thm -> formula list

  (* But creation is only allowed by these primitive rules *)
  val AXIOM   : formula list -> thm
  val ASSUME  : formula -> thm
  val INST    : subst -> thm -> thm
  val FACTOR  : thm -> thm
  val RESOLVE : formula -> thm -> thm -> thm
end
```

Fig. 2. The ML Signature of a Logical Kernel Implementing Clausal First-Order Logic.

The ML type system can be used to ensure that these primitive rules of inference represent the only way to create elements of an abstract `thm` type.[6] In Figure 2 we show the signature of an ML `Kernel` module that implements the logical kernel. We insist that the programmer of a first-order provers derive refutations by creating an empty clause of type `thm`. The only way to do this is to use the primitive rules of inference in the `Kernel` module: this is both easy and efficient for all the standard first-order proof procedures.

At this point it is simple to translate first-order refutations to HOL proofs. We add proof logs into the representation of theorems in the `Kernel`, so that each theorem remembers the primitive rule and theorems that were used to create it. When we complete a refutation, we therefore have a chain of proof steps starting at the empty clause and leading back to axioms. In addition, for each primitive rule of inference in `Kernel`, we create a higher-order logic version that works on HOL terms, substitutions and theorems.[7] The final ingredient needed to translate a proof is a HOL theorem corresponding to each of the first-order axioms. These theorems are the HOL clauses in the CNF representation of the original (negated) goal, which we mapped to first-order logic and axiomatized.

To summarize: by requiring the programmer of a first-order proof procedure to derive refutations using a logical kernel, lifting these refutations to HOL proofs can be done completely automatically.

[6] Indeed, the ability to define an abstract theorem type was the original reason that the ML type system was created.

[7] If we omit the types from our mapping of HOL terms to first-order logic, it is possible that the first-order refutation cannot be translated to a valid HOL proof. In this case we can either abort, or restart the whole procedure with types included.

4 Conclusions and Related Work

We tested the LCF-style kernel for clausal first-order logic by implementing ML versions of various first-order calculi, and found it to be more than just a convenient interface to a proof translator. Reducing the steps of proof procedures to primitive inferences clarified their behaviour, and also helped catch bugs early. Also, assuming the (52 line) ML `Kernel` module is correctly implemented and the programmer is careful about asserting axioms, loss of soundness arising from 'prover optimizations' can be completely avoided.

In addition to MESON in HOL, there have been many other examples of automatic first-order provers being used to prove problems in an interactive theorem prover, including: FAUST in HOL [5]; Paulson's `blast` in Isabelle [6]; and Bliksem in Coq [1].

In these link-ups, various mappings are used from theorem prover subgoals into problems of first-order logic, defining the class of subgoals that may be feasibly proved using the underlying first-order prover. The architecture presented in this paper for translating first-order refutations allows different first-order provers to be 'plugged-in' to the theorem prover, with small marginal effort. Moreover, if first-order provers emitted proofs in a standardized 'LCF-style' logical kernel for clausal first-order logic, then this would further simplify their integration into interactive theorem provers.

References

1. Marc Bezem, Dimitri Hendriks, and Hans de Nivelle. Automated proof construction in type theory using resolution. In David A. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *Lecture Notes in Computer Science*, pages 148–163, Pittsburgh, PA, USA, June 2000. Springer.
2. M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer Verlag, 1979.
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. John Harrison. Optimizing proof search in model elimination. In Michael A. McRobbie and John K. Slaney, editors, *13th International Conference on Automated Deduction (CADE-13)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 313–327, New Brunswick, NJ, USA, July 1996. Springer.
5. R. Kumar, T. Kropf, and K. Schneider. Integrating a first-order automatic prover in the HOL environment. In Myla Archer, Jeffrey J. Joyce, Karl N. Levitt, and Phillip J. Windley, editors, *Proceedings of the 1991 International Workshop on the HOL Theorem Proving System and its Applications (HOL '91), August 1991*, pages 170–176, Davis, CA, USA, 1992. IEEE Computer Society Press.
6. L. C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3), March 1999.