# First-Order Proof Tactics in Higher-Order Logic Theorem Provers

Joe Hurd[⋆]

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

**Abstract.** In this paper we evaluate the effectiveness of first-order proof procedures when used as tactics for proving subgoals in a higher-order logic interactive theorem prover. We first motivate why such first-order proof tactics are useful, and then describe the core integrating technology: an 'LCF-style' logical kernel for clausal first-order logic. This allows the choice of different logical mappings between higher-order logic and first-order logic to be used depending on the subgoal, and also enables several different first-order proof procedures to cooperate on constructing the proof. This work was carried out using the HOL4 theorem prover; we comment on the ease of transferring the technology to other higher-order logic theorem provers.

## 1 Introduction

Performing interactive proof in the HOL theorem prover [12] 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 [13] with the ability to translate proofs to higher-order logic. The original MESON procedure, due to Loveland [17], is a sound and complete calculus for finding proofs in full first-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 HOL4[1] relies on MESON to prove 1726 subgoals; the HOL formalization of probability theory, including an example verification of the Miller-Rabin primality test, contributes another 1953 subgoals to this total.

The primary goal of this paper is to evaluate the effectiveness of different first-order proof calculi for use as HOL tactics supporting interactive proof. We compare the performance of several first-order calculi on three different problem sets: the TPTP first-order problem set; and two problem sets derived from HOL subgoals proved by MESON. The TPTP (Thousands of Problems for Theorem Provers) problem set is a collection of first-order problems designed to test the limits of current automatic first-order provers [24]. This experiment allows us to directly compare the performance of the first-order proof procedures in the different environments of fully automatic proof of deep theorems and supporting a user engaged in driving an interactive theorem prover. In this paper we show that performance in these two environments is correlated. Therefore, if a new first-order prover is developed that can prove more TPTP problems than the existing state of the art, we can expect the same prover to prove more HOL subgoals, thus improving the user experience.

The most obvious difference between the fully-automatic and interactive environments is the real-time nature of interactive proof. Whether the cost of proof search is incurred each time the theory is loaded, compiled, or even just once when the theory is created, the user usually requires any HOL tactic to respond (almost) immediately. By contrast, fully automatic provers are generally judged on the number of theorems that they can prove within a much more relaxed time-limit. To simulate this environmental difference in our experiments, for the TPTP problem set we allow the provers 60 seconds per problem, and for the HOL problem sets we allow only 5 seconds per problem.

A limit of 5 seconds per problem suggests that the performance of a prover may be rather sensitive to the characteristics of initial proof search. Following this reasoning, it seems plausible that a combination

[⋆] This work was supported by EPSRC project GR/R27105/01

[1] HOL4 is available at `http://hol.sf.net/`.

of different proof procedures may perform better than any individual, on the grounds that any problems that are 'shallow' for one of the procedures may be quickly solved within the time limit. We therefore implement a proof procedure that combines resolution, model elimination, and the Delta preprocessor. All three procedures may be run in parallel (using time slicing), and they cooperate by sharing unit clauses.[2] It turns out that not only does this combination procedure significantly outperform each individual procedure on the HOL problem sets, but also on the TPTP problem set.

The device that allows the provers to share unit clauses is a small 'LCF-style' kernel for clausal first-order logic. As well as providing a convenient mechanism for detecting unit clauses derived by model elimination, it also provides a convenient place to install the proof recording in the event that it is necessary to translate them to HOL. In particular, it is the only place that needs to worry about keeping track of proofs, and this enabled broader experimentation with the first-order provers.

Since HOL4 is written in Standard ML, this is a convenient implementation language for our experiment, though in the past similar experiments have been performed by making calls to external C provers [14]. Therefore, this paper also provides a view of implementing first-order proof procedures in a functional programming language, and some interesting aspects of this are brought out in discussion.

The secondary goal of this paper is to serve as a 'HOW-TO guide' for would-be implementors of first-order proof tactics in higher-order theorem provers. We will present all the steps necessary to prove higher-order subgoals using automatic first-order provers: the initial conversion from higher-order subgoal to first-order clauses; the first-order proof search; and the final translation of the first-order refutation to a higher-order logic theorem.

The main contributions of this paper are as follows:

– A relative performace comparison of different first-order proof calculi in the two environments of proving deep first-order theorems (the TPTP problem set) and aiding the user engaged in interactive proof (the two problem sets extracted from HOL subgoals).
– A combination resolution and model elimination procedure that performs significantly better than either individually, in both the TPTP and the HOL environments.
– A detailed description of how to implement tactics for proving higher-order logic subgoals using first-order proof procedures.

The paper is structured as follows: in Section 2 we point out the interesting features of the mapping between higher-order and first-order logic; Section 3 examines the syntactic differences between the problem sets and presents our evaluation methodology; in Section 4 we describe the ML implementation of the first-order provers and their subsequent optimization; Section 5 presents the results of running different combinations of provers on the problem sets; Section 6 comments on how this technology might be ported to other higher-order logic theorem provers; and finally in Sections 7 and 8 we conclude and take a look at related work.

## 2 The HOL Interface to First-Order Logic

This is the high-level view of how we prove the HOL goal $g$ using a first-order prover:

1. We first convert the negation of $g$ to conjunctive normal form; this results in a HOL theorem of the form
$$\vdash \neg g \iff \exists \boldsymbol{a}.\ (\forall \boldsymbol{v_1}.\ c_1) \land \cdots \land (\forall \boldsymbol{v_n}.\ c_n) \tag{1}$$
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 map each HOL term $c_i$ to first-order logic, producing the clause set
$$C = \{C_1, \ldots, C_n\}$$

[2] Unit clauses are clauses with only one literal, and are used to simplify other clauses.

3. The first-order prover runs on $C$, and finds a refutation $\rho$.
4. By proof translation, the refutation $\rho$ 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

$$\vdash g \tag{3}$$

In the following subsections we examine the translation of formulas and proofs between higher-order logic and first-order logic, which plays a role in steps 2 and 4 of the above process. Much of this information appears in a previously published system description [15]; it is reproduced here because it is an essential part of our framework for creating first-order proof tactics.

Before getting into the details, we first give an extended example of the whole process with a typical HOL subgoal that we prove using a first-order proof tactic. Consider the subgoal

$$\forall\, x, y, z.\ \mathsf{divides}\ x\ y \Rightarrow \mathsf{divides}\ x\ (z * y) \tag{4}$$

where the predicate $\mathsf{divides}$ is defined as

$$\vdash \forall\, x, y.\ \mathsf{divides}\ x\ y \iff \exists\, z.\ y = z * x \tag{5}$$

To prove the subgoal, we also need the following theorems about multiplication:

$$\vdash \forall\, x, y.\ x * y = y * x \tag{6}$$
$$\vdash \forall\, x, y, z.\ (x * y) * z = x * (y * z) \tag{7}$$

The user invokes the first-order proof tactic on the subgoal (4), passing as arguments the definition (5) and theorems (6) and (7). Initially, the first-order proof tactic uses the arguments to set up the equivalent subgoal $g$:

$$(5) \wedge (6) \wedge (7) \Rightarrow (4)$$

The next step is to negate $g$ and convert to conjunctive normal form. This conversion is completely standard—negation normal form followed by pushing out quantifiers and Skolemization—and we refer the interested reader to a textbook such as Chang and Lee [6] for more details. In our example, this results in the theorem

$$\begin{aligned}
&\vdash \neg g \iff \\
&\quad \exists\, a, b, c, d. \\
&\qquad (\forall\, x, y.\ x * y = y * x)\ \wedge \\
&\qquad (\forall\, x, y, z.\ (x * y) * z = x * (y * z))\ \wedge \\
&\qquad (\forall\, x, y, z.\ \neg(y = z * x)\ \vee\ \mathsf{divides}\ x\ y)\ \wedge \\
&\qquad (\forall\, x, y.\ \neg\mathsf{divides}\ x\ y\ \vee\ y = d\ x\ y * x)\ \wedge \\
&\qquad \mathsf{divides}\ a\ b\ \wedge \\
&\qquad \neg\mathsf{divides}\ a\ (c * b)
\end{aligned}$$

We map each line of this formula to a first-order clause. The existential variables $a, b, c, d$ are mapped to first-order function symbols, and the universal variables $x, y, z$ are mapped to first-order variables. The first-order prover runs, finds a refutation, and this is translated to a HOL theorem from which the first-order tactic derives $\vdash g$, thus proving the initial goal.

## 2.1 Mapping HOL Terms to First-Order Logic

Seemingly the hardest problem with mapping HOL terms to first-order logic—dealing with $\lambda$-abstractions—can be smoothly dealt with as part of the conversion to CNF. Any $\lambda$-abstraction at or beneath the literal level is rewritten to combinatory form, using the set of combinators $\{S, K, I, C, \circ\}$. Using this set of combinators prevents the exponential blow-up that is encountered when only $\{S, K, I\}$ are used [25].[3]

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 $m \leq n$ is mapped to the first-order formula $B(@(@(\leq, m), n))$. The only exception to this rule is equality: the HOL term $x = y$ is mapped to the first-order logic formula $=(x, y)$.

As described thus far, this mapping is used to generate the **uHOL** first-order problem set from HOL subgoals sent to MESON. uHOL stands for untyped HOL, because no type information is included in this representation. However, we also experimented with including higher-order logic types in the first-order mapping 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.[4] This mapping is used to produce the **HOL** problem set from HOL subgoals. As might be expected, this 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. This hypothesis is examined in Section 5.

## 2.2 Translating First-Order Refutations to HOL

When the first-order prover has found a refutation of a set of clauses, the HOL tactic must translate the refutation to a HOL theorem, thus ensuring that no soundness bugs in the first-order prover are propagated into 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 logical kernel idea of the LCF project [11], we can make the proof translation invisible to the developer of first-order proof procedures, leaving him free to experiment with new calculi. We have implemented this automatic proof translation for both the mapping with type information and the one without, and they have been successfully used to prove many HOL subgoals.

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

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

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

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

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

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

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 only need the five rules in Figure 1.

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 $L \lor \neg L$.[5] The INST rule takes a substitution $\sigma$ and a theorem $A$, and applies the substitution to every literal in $A$.[6] 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.

These five primitive rules define a (refutation) complete proof system for clausal first-order logic. To see this, recall that a complete proof system results from Factor and Resolve rules that perform unification [6]. However, we can simulate these rules by first instantiating appropriately using the INST rule, and then applying our identical-match versions FACTOR and RESOLVE.

```
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.[7] 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. The final ingredient needed to translate a proof is a HOL theorem corresponding to each of the first-order axioms.

[5] This rule is used to keep track of reductions in the model elimination procedure.
[6] In some presentations of logic, this uniform instantiation of variables in a theorem is called specialization.
[7] Indeed, the ability to define an abstract theorem type was the original reason that the ML type system was created.

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.

### 2.3 The Scope of a First-Order Prover for HOL

Using the mapping in Section 2.1, we can use a first-order prover to prove some higher-order HOL goals, such as the classic derivation of an identity function from combinator theory:

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

Similarly, the framework for translating refutations in Section 2.2 is general enough to translate any first-order theorem to HOL. Therefore, we can use a first-order prover to solve for HOL terms satisfying a set of HOL formulas, just as Prolog does for Horn formulas.[8]

However, our method of embedding higher-order in first-order logic is not without danger. The whole reason for adding types to higher-order logic is to avoid the Russell paradox, and so if we choose to remove them in our translation we must beware of unsoundness. Defining a 'Russell combinator' $R$ as

$$R = (\lambda x.\ \neg(x\ x)) = \mathsf{S}\ (\mathsf{K}\ (\neg))\ (\mathsf{S}\ \mathsf{I}\ \mathsf{I})$$

we find that we can use the reduction rules for $\mathsf{S}$ and $\mathsf{K}$ to prove

$$R\ R = \neg(R\ R)$$

and thus derive a contradiction.

Of course, a first-order refutation that is unsound in this way cannot be successfully translated to a HOL proof. It is therefore trivial to discover any problems that occur due to the lack of type information in the first-order representation. When unsoundness is discovered, the subgoal is simply tried again with the type information included. Fortunately, this phenomenon occurs in less than 1% of all HOL subgoals.

## 3 Problem Sets and Evaluation Methodology

In the next section we describe the ML implementation of our combination of first-order provers. To evaluate and compare different procedures, we use the following three problem sets:

**TPTP** This consists of all the problems classified as 'unsatisfiable' in version 2.4.1 of TPTP.[9]

**uHOL** This problem set consists of all subgoals proved by MESON when building: the standard theories included with version Kananaskis-0 of the HOL4 theorem prover; the HOL formalization of probability theory; and the example verification of the Miller-Rabin primality test.[10] The HOL subgoals are mapped to first-order logic without type information (uHOL = untyped HOL).

**HOL** The same problem set as uHOL, but the HOL subgoals are mapped to first-order logic with type information included.

Table 1 profiles the three problem sets. For each problem set, we show the number of problems (**N**), and the median of several statistics for each problem: number of clauses (**C**), number of literals (**L**), number of symbols[11] (**S**), mean literals per clause (**L**/**C**), mean symbols per clause (**S**/**C**), and mean symbols per literal (**S**/**L**).

[8] The model elimination procedure has the capability to solve for terms in this way.
[9] The TPTP problem set is available at `http://www.cs.miami.edu/~tptp/`.
[10] Available at `http://www.cl.cam.ac.uk/~jeh1004/research/problems/`.
[11] By symbols we mean variables, functions, relations and logical connectives.

**Table 1.** Profiles of the Problem Sets.

| Set | N | C | L | S | L/C | S/C | S/L |
|---|---|---|---|---|---|---|---|
| **TPTP** | 2752 | 31.0 | 65.0 | 229.0 | 2.07 | 8.17 | 4.00 |
| **uHOL** | 3679 | 11.0 | 19.0 | 146.0 | 1.78 | 12.86 | 7.14 |
| **HOL** | 3679 | 11.0 | 19.0 | 701.0 | 1.78 | 63.71 | 35.30 |

Comparing the TPTP and HOL problem sets, it can be seen that the average TPTP problem has more clauses, while the average HOL problem has more symbols per literal. However, by looking at the uHOL row, it is apparent that most of the symbols in the HOL problems come from type information. One similarity between all three problem sets is the average number of literals per clause: around two.

As previously mentioned, we allow 60 seconds per TPTP problem and 5 seconds per HOL problem, to simulate the difference in requirements between fully automatic and interactive proof. All experiments were run on Athlon 1.4GHz processors with at least 512Mb of memory, using version 2.00 of Moscow ML[12] on RedHat Linux 7.1.

So that we can use statistical methods to compare the first-order provers, we randomly split each problem set into 10 equally sized sections. By counting the number of problems in each section that any two provers solve within the time limit, we can use the *t*-test to compute the statistical significance that one prover is better than the other [9]. Here is an example results table where we compare two hypothetical provers, **foo** and **bar**:

| | **foo** | **bar** |
|---|---|---|
| **foo** | $*$ | $\overset{+95}{99.5\%}$ |
| **bar** | $\overset{+7}{\text{———}}$ | $*$ |

Since we do not compare provers with themselves, the diagonal entries are marked with $*$. The 99.5% in the upper right entry means that **foo** is statistically better than **bar** with 99.5% confidence. The +95 above this means that, over the whole problem set, **foo** proved 95 problems that **bar** could not. The lower left entry in the table means that **bar** is *not* significantly better than **foo**, but it did prove 7 problems that **foo** could not.

## 4 Implementing the First-Order Provers

In Sections 2 and 3 we presented a mechanism for mapping HOL subgoals to first-order problems and a way to evaluate first-order provers. In this section we will describe an ML implementation of a collection of first-order proof procedures, using our evaluation method to select optimum parameters and justify optimizations.

Since the literature contains such an abundance of strategies and techniques for first-order proof, it was necessary to select just a few for the purpose of our present experiment. In particular, we do not treat equality at all, and add equality axioms as part of our mapping to first-order logic. However, in the future there is nothing to stop us including more sophisticated methods for handling equality, say by adding a `PARAMODULATION` primitive inference rule.

### 4.1 Model Elimination Procedure

The first proof procedure that we implement is the model elimination procedure of Loveland [17]; our prover is essentially a ground-up reimplementation of Harrison's MESON [13], incorporating some optimizations of Astrachan, Loveland and Stickel [2, 3].

[12] Moscow ML is available at `http://www.dina.dk/~sestoft/mosml.html`.

Our strategy is to first produce a naive implemention, and then incrementally optimize it. The starting point is a version of model elimination called **m-0** that treats every input clause as an initial clause. A clause must contain at least one negative literal for **m-1** to treat it as initial, and initial clauses in **m-n** must contain all negative literals.

Building upon **m-n**, we add ancestor pruning to get **m-a**, and then ancestor cutting to get **m-x**. Ancestor cutting means that if the negation of an ancestor exactly matches the current goal, we do a reduction on that ancestor and disallow backtracking. Incorporating Harrison's divide-and-conquer search strategy brings us to **m-d**, and trying to match the goal from the clause set before trying unification is called **m-s**.[13] The optimization in **m-c** is slightly dubious, incorporating a limited form of caching to stop us attempting the same goal twice from a given point in the search. The overhead of this pays off on the TPTP problem set with a time limit of 60 seconds, but not on the HOL problem sets where the limit is 5 seconds.

Finally, **m-u** incorporates unit lemmaizing, where the use of a unit lemma contributes size 1 to the proof. Since we use iterative deepening to search for proofs in order of size, the penalty of using a unit lemma is an important factor in the optimization. However, we cannot make the penalty depend on the proof size of the unit lemma, since later we will import unit lemmas from a resolution prover where proof sizes do not play any part.

**Table 2.** Comparing Model Elimination Optimizations on the TPTP Problem Set.

| | **m-u** | **m-c** | **m-s** | **m-d** | **m-x** | **m-a** | **m-n** | **m-1** | **m-0** |
|---|---|---|---|---|---|---|---|---|---|
| **m-u** | ∗ | +89 99.5% | +91 99.5% | +99 99.5% | +180 99.5% | +186 99.5% | +198 99.5% | +345 99.5% | +371 99.5% |
| **m-c** | +13 | ∗ | +2 80.0% | +14 97.5% | +103 99.5% | +109 99.5% | +121 99.5% | +269 99.5% | +295 99.5% |
| **m-s** | +13 | +0 | ∗ | +13 95.0% | +101 99.5% | +107 99.5% | +119 99.5% | +267 99.5% | +293 99.5% |
| **m-d** | +9 | +0 | +1 | ∗ | +90 99.5% | +96 99.5% | +108 99.5% | +255 99.5% | +281 99.5% |
| **m-x** | +12 | +11 | +11 | +12 | ∗ | +7 95.0% | +25 99.5% | +178 99.5% | +204 99.5% |
| **m-a** | +12 | +11 | +11 | +12 | +1 | ∗ | +19 97.5% | +174 99.5% | +200 99.5% |
| **m-n** | +5 | +4 | +4 | +5 | +0 | +0 | ∗ | +158 99.5% | +184 99.5% |
| **m-1** | +0 | +0 | +0 | +0 | +1 | +3 | +6 | ∗ | +29 99.5% |
| **m-0** | +0 | +0 | +0 | +0 | +1 | +3 | +6 | +3 | ∗ |

Table 2 shows the result of a pairwise comparison between each step in the evolution of our model elimination prover, from the humble **m-0** to the hi-tech **m-u** which proves 371 more TPTP problems. Similar results occur on the HOL problem sets.

## 4.2 Resolution Procedure

The second proof procedure that we implemented is the resolution procedure of Robinson [21]. Our version uses the given clause algorithm, and we implement term nets to improve the speed of unification and subsumption checking. Additionally, unit clauses are used whenever possible to simplify clauses. In contrast with the incremental sequence of optimizations that we used for model elimination, our resolution procedure has several independent parameters that control the search strategy.

[13] The fact that we are implementing this in ML might help to explain why this is such an effective optimization. If matching succeeds then there is no need to update the substitution context, and this results in less allocation and reduced garbage collection times. These reductions range between 20% and 80% on TPTP problems.

The first parameter controls how much subsumption checking is done. By default, as clauses are taken from the unused list they are checked to see if they are subsumed by a clause in the used list. If so, they are dropped and the next clause is chosen from the unused list. We also implement a higher level of subsumption, indicated by an extra **s** in the prover name. Here, when we lift a clause from the unused list, we immediately see if it is subsumed by another clause in the unused list. If so, we use that clause instead.

The second parameter is a number $n$, and controls the order that we pick clauses from the unused list. For every clause picked from the unused list in FIFO order, we pick $n$ clauses with the smallest symbol count.[14] The number $n$ is one of **1**, **2**, **3**, **4** or **5**, and is part of the prover name. This is called the ratio strategy, originally used in the Otter theorem prover [26].

The final parameter is Robinson's positive refinement [20], which we indicate with a final '$^+$' in the prover name.

We found that the best prover for all three problem sets is $\mathbf{r3}^+$: the default level of subsumption; picking 3 smallest clauses for every clause at the head of the queue; and using positive resolution. On the TPTP problem set, this parameter setting is better than any other with confidence at least 95%.

### 4.3 Delta-style Procedure

The third and final proof procedure that we implemented is based on the Delta preprocessor of Schumann [23]. Put simply, for every $n$-ary relation $R$ present in the problem, we generate the 'Delta goals' $R(X_1, \ldots, X_n)$ and $\neg R(Y_1, \ldots, Y_n)$ (with fresh variables $X_i$ and $Y_i$). We then use the model elimination procedure with iterative deepening to search for solutions to the Delta goals. Every unit clause that is derived during this process is shared with the other proof procedures.

The Delta procedure takes the same optimization parameters as model elimination, and so it is not necessary to separately optimize this procedure. In any case, since it is not designed to directly solve the goal, but rather to help the other procedures by finding useful unit clauses, it only makes sense to use it when unit lemmaizing is switched on.

## 5 Combining the First-Order Provers

As already mentioned, when we run different proof procedures together they can cooperate by sharing unit clauses. Whenever a unit clause is derived, it is inserted into a global store that is available to every proof procedure. The way that the individual proof procedures make use of unit clauses was described in the previous sections.

Each proof procedure runs for a time-slice,[15] and a scheduler decides which proof procedure to run based on the cost of the execution time it has already consumed. For model elimination and resolution, the cost of execution time is simply the number of seconds, but for the Delta procedure it was empirically found to be better to use the square of the number of seconds.

For example, if each proof procedure has consumed 1/3 second of CPU time, the model elimination and resolution cost is 1/3, while the Delta cost is $(1/3)^2 = 1/9$. Therefore, Delta will be scheduled as the cheapest procedure. If each proof procedure has consumed 2 seconds, the model elimination and resolution cost is 2, while the Delta cost is $2^2 = 4$, and so one of model elimination and resolution will be scheduled to run for a time-slice.

Tables 3, 4, and 5 show the result of running different proof procedure combinations on the TPTP, uHOL and HOL problem sets, respectively.

In every case, the combined model elimination and resolution procedure performed significantly better than either individually, with the highest level of confidence (99.5%). For the TPTP problem set, we can use some arithmetic to see that there must exist at least 166 problems that were proved by the combined procedure but were not proved by either acting alone. This is compelling evidence that the combined procedure

[14] We efficiently implement this alternation in ML by storing unused clauses as both queues and (leftist) heaps. Okasaki [18] implements functional versions of these and many more data structures.

[15] In our experiments we set each time slice to be 1/3 second long.

**Table 3.** Comparing Combinations of Provers on the TPTP Problem Set.

| | **mrd** | **mr** | **md** | **m** | **rd** | **r** |
|---|---|---|---|---|---|---|
| **mrd** | ∗ | +22 99.5% | +160 99.5% | +200 99.5% | +291 99.5% | +322 99.5% |
| **mr** | +9 — | ∗ | +161 99.5% | +189 99.5% | +283 99.5% | +307 99.5% |
| **md** | +22 — | +36 — | ∗ | +56 99.5% | +274 99.5% | +298 99.5% |
| **m** | +13 — | +15 — | +7 — | ∗ | +243 99.5% | +264 99.5% |
| **rd** | +25 — | +30 — | +146 — | +164 — | ∗ | +42 99.5% |
| **r** | +25 — | +23 — | +139 — | +154 — | +11 — | ∗ |

**Table 4.** Comparing Combinations of Provers on the uHOL Problem Set.

| | **mrd** | **mr** | **md** | **m** | **rd** | **r** |
|---|---|---|---|---|---|---|
| **mrd** | ∗ | +16 70.0% | +111 99.5% | +187 99.5% | +164 99.5% | +148 99.5% |
| **mr** | +13 — | ∗ | +119 99.5% | +182 99.5% | +156 99.5% | +137 99.5% |
| **md** | +11 — | +22 — | ∗ | +91 99.5% | +152 97.5% | +133 90.0% |
| **m** | +14 — | +12 — | +18 — | ∗ | +133 — | +113 — |
| **rd** | +21 — | +16 — | +109 — | +163 90.0% | ∗ | +33 — |
| **r** | +22 — | +14 — | +107 — | +160 99.0% | +50 90.0% | ∗ |

does more than simply harvest the problems that are 'shallow' for one of model elimination and resolution. Rather, the sharing of unit clauses creates a whole new procedure that is better than either.

Comparing the combinations of proof procedures across the three problem sets, as we move from TPTP through uHOL to HOL we find resolution becoming better relative to model elimination. Similarly, the Delta procedure helps the combined procedure less as we move from TPTP to HOL. This latter effect is probably due to the cost functions we chose, which favours Delta in the first 3 seconds of CPU time. When the total limit is 5 seconds, this represents a serious bias.

Because the proof procedures share unit clauses, some rather counter-intuitive effects can arise from combining proof procedures. For example, there is a class of 7 TPTP problems that model elimination can prove, but model elimination and Delta together fail to prove within 60 seconds. One of these is `GRP128_4_003`, which model elimination acting alone proves in about 12 seconds! The only explanation is that Delta finds some 'helpful' unit clauses that lead model elimination into an unprofitable area of the search space. However, these kind of events are rare: the other 6 problems in this class take model elimination acting alone more than 45 seconds to prove.

Finally, we can compare the performance of provers on the different versions of each problem in the uHOL and HOL problem sets. The best prover in this domain is the combination of model elimination and resolution, and there were 142 problems that it could prove in uHOL but not in HOL, and 13 problems that it could prove in HOL but not in uHOL. Therefore, this confirms our expectations that the much smaller versions of the problems in uHOL can be proved more efficiently, though there are a few examples where the types cut down the search space enough to make the difference between finding a proof within the time limit and not.

**Table 5.** Comparing Combinations of Provers on the HOL Problem Set.

| | **mrd** | **mr** | **md** | **m** | **rd** | **r** |
|---|---|---|---|---|---|---|
| **mrd** | $*$ | +9 — | +171 99.5% | +246 99.5% | +174 99.5% | +127 99.5% |
| **mr** | +22 95.0% | $*$ | +184 99.5% | +258 99.5% | +184 99.5% | +131 99.5% |
| **md** | +25 — | +25 — | $*$ | +102 99.5% | +168 75.0% | +124 — |
| **m** | +23 — | +22 — | +25 — | $*$ | +146 — | +105 — |
| **rd** | +14 — | +11 — | +154 — | +209 99.5% | $*$ | +20 — |
| **r** | +32 — | +23 — | +175 99.5% | +233 99.5% | +85 99.5% | $*$ |

## 6 Transferring the Technology to Other Higher-Order Logic Theorem Provers

The results of the previous section show that we can create combination first-order proof tactics that are more powerful than individual proof procedures, and the LCF-style logical kernel allows us to easily translate first-order refutations to higher-order logic theorems. The existing implementation in HOL4 has been found to be a useful proof tool when used in proofs by the HOL developers.

The general architecture is easily ported to other higher-order logic theorem provers such as PVS or Isabelle. Indeed, the first-order proof procedures complete with logical kernel are a standalone library in Standard ML, so could be imported without any change at all.[16] The only part that needs to be created afresh for each higher-order logic theorem prover is a rule of inference that translates first-order refutations to higher-order logic theorems. This will differ slightly according to the particular details of the higher-order logic.

There is one difference between HOL and PVS that may be significant here. When translating a first-order refutation to higher-order logic, it is often necessary to translate a first-order term to higher-order logic. In HOL this is straightforward, but in PVS some auxiliary theorems may be necessary to establish the TCCs of the translated term. Additionally, there may be no type information in the first-order term, in which case we generate a HOL term and infer its principal type. In PVS this may not be possible in general, which may jeopardize the possibility of an untyped mapping to first-order logic. More investigation is necessary to establish whether the required information can be reconstructed in some way.

Finally, for some interactive theorem proving applications, it may be appropriate to simply trust that the subgoal is valid if the first-order prover detects unsatisfiability of the corresponding clauses. There is already strong pressure on the implementors of first-order provers to make sure their system is sound; at the annual CADE automatic system competition, provers face disqualification if they fail any soundness test. Abandoning the idea of translating refutations gives an additional benefit: many first-order provers perform much better if they are not required to keep track of the refutation as they search.

## 7 Conclusions

In this paper we described a framework for implementing first-order proof tactics in higher-order logic theorem provers, which uses an LCF-style logical kernel to create a modular interface between the two logics. The architecture we presented is not specific to a particular higher-order theorem prover, and we have sketched out how it could be ported to a new theorem prover, notwithstanding the potential problem with type reconstruction in PVS. We have implemented a version in HOL4, with two working mappings: one that preserves type information from higher-order logic, and one that reconstructs it while translating the first-order refutation.

[16] Available at `http://www.cl.cam.ac.uk/~jeh1004/research/metis/`.

We also implemented a combination of first-order proof procedures in Standard ML, and compared their performance on three different problem sets. Based on our experiments, we tentatively conclude that good performance from a first-order prover in one domain suggests that it will also perform well in other domains. Optimizations we made to improve performance on the TPTP problem set usually also improved performance on the HOL problem sets, though there was a significant shift from model elimination in the TPTP domain towards resolution in the HOL domain. This was extremely surprising, since the HOL problems are self-selected for the MESON prover in HOL. During interactive proof in HOL, if MESON cannot prove a subgoal within a reasonable time, then a user can perform a manual inference step and then try again. Further investigation is needed to establish why resolution seems to do better on HOL subgoals.

We found the LCF-style kernel for clausal first-order logic 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.

Finally, on all three problem sets the combination of model elimination and resolution was found to perform significantly better than either individually. This supports the hypothesis that first-order search spaces have a structure that rewards the use of a variety of search methods, despite the extra redundancy that is entailed; a view neatly summarized by Astrachan and Loveland [2]:

> "Unlike chess, theorems are a very diverse lot and different proof methods may excel in different areas."

## 8 Related Work

In addition to MESON in HOL, there are many other examples of automatic first-order provers being used to prove problems in an interactive theorem prover: (in chronological order) FAUST in HOL [16]; SEDUCT in LAMBDA [5]; 3TAP in KIV [1]; Paulson's `blast` in Isabelle [19]; Gandalf in HOL [14]; and Bliksem in Coq [4]. Various mappings are used from the theorem prover subgoals into problems of first-order logic, defining the scope of what can be automatically proved. Using the architecture presented in this paper for translating first-order refutations would allow different first-order provers to be 'plugged-in' to the theorem prover. 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.

As part of the ILF Mathematical Library Project, Dahn and Wernhard [8] extracted 97 first-order problems from the article *Boolean Properties of Sets* in the MIZAR Mathematical Library. Later, Dahn [7] added the ability to represent MIZAR type information, and extracted 47 problems from the article *Relations Defined on Sets*. However, there has been no published study of the comparitive effectiveness of first-order provers on this problem set.

Several projects have aimed to create combination first-order provers that are better than the individual components. For example, the TECHS system [10] uses automatic referees to decide which clauses to exchange between provers, as opposed to our system that simply shares unit clauses. Further investigation is needed to decide the best way of combining proof procedures in our application.

Finally, we note that Robinson [22] proposed a version of higher-order logic in terms of combinators (though it is typeless and therefore unsound due to the 'Russell combinator' we defined in Section 2.3).[17] However, the motivation behind combinators instead of the $\lambda$-calculus is that proof automation can be simplified, and this also motivates our combinator mapping from HOL subgoals to first-order logic.

## Acknowledgements

This project greatly benefitted from discussions with Mike Gordon, Michael Norrish and Konrad Slind. Also, the CAML code in John Harrison's forthcoming book provided a useful starting point for my own ML implementation of first-order provers. Thanks are also due to Judita Preiss for technical support.

[17] Thanks to John Harrison for drawing my attention to this.

## References

1. Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, Wolfram Menzel, Wolfgang Reif, Gerhard Schellhorn, and Peter H. Schmitt. Integration of automated and interactive theorem proving. In W. Bibel and P. Schmitt, editors, *Automated Deduction: A Basis for Applications*, volume II, chapter 4, pages 97–116. Kluwer, 1998.
2. O. L. Astrachan and Donald W. Loveland. The use of lemmas in the model elimination procedure. *Journal of Automated Reasoning*, 19(1):117–141, August 1997.
3. Owen L. Astrachan and Mark E. Stickel. Caching and lemmaizing in model elimination theorem provers. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE-11)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 224–238, Saratoga Springs, NY, USA, June 1992. Springer.
4. 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.
5. H. Busch. First-order automation for higher-order-logic theorem proving. In Tom Melham and Juanito Camilleri, editors, *Higher Order Logic Theorem Proving and Its Applications, 7th International Workshop*, volume 859 of *Lecture Notes in Computer Science*, Valletta, Malta, September 1994. Springer.
6. Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.
7. Ingo Dahn. Interpretation of a Mizar-like logic in first-order logic. In Ricardo Caferra and Gernot Salzer, editors, *International Workshop on First-Order Theorem Proving (FTP '98)*, Technical Report E1852-GS-981, pages 116–126, Vienna, Austria, November 1998. Technische Universität Wien.
8. Ingo Dahn and Christoph Wernhard. First order proof problems extracted from an article in the MIZAR mathematical library. In Maria Paola Bonacina and Ulrich Furbach, editors, *International Workshop on First-Order Theorem Proving (FTP '97)*, number 97-50 in RISC-Linz Report Series, Schloss Hagenberg, Austria, October 1997. Johannes Kepler Universität Linz.
9. Morris DeGroot. *Probability and Statistics.* Addison-Wesley, 2nd edition, 1989.
10. Jörg Denzinger and Dirk Fuchs. Knowledge-based cooperation between theorem provers by TECHS. SEKI-Report SR-97-11, University of Kaiserslautern, 1997.
11. M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science.* Springer, 1979.
12. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL (A theorem-proving environment for higher order logic).* Cambridge University Press, 1993.
13. 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.
14. Joe Hurd. Integrating Gandalf and HOL. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs '99*, volume 1690 of *Lecture Notes in Computer Science*, pages 311–321, Nice, France, September 1999. Springer.
15. Joe Hurd. An LCF-style interface between HOL and first-order logic. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 134–138, Copenhagen, Denmark, July 2002. Springer.
16. 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.
17. Donald W. Loveland. Mechanical theorem proving by model elimination. *Journal of the ACM*, 15(2):236–251, April 1968.
18. Chris Okasaki. *Purely Functional Data Structures.* Cambridge University Press, Cambridge, UK, 1998.
19. L. C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3), March 1999.
20. J. A. Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.
21. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–49, January 1965.

22. J. A. Robinson. A note on mechanizing higher order logic. *Machine Intelligence*, 5:121–135, 1970.
23. Johann Ph. Schumann. DELTA — A bottom-up processor for top-down theorem provers (system abstract). In Alan Bundy, editor, *12th International Conference on Automated Deduction (CADE-12)*, volume 814 of *Lecture Notes in Artificial Intelligence*, Nancy, France, June 1994. Springer.
24. Christian B. Suttner and Geoff Sutcliffe. The TPTP problem library — v2.1.0. Technical Report JCU-CS-97/8, Department of Computer Science, James Cook University, December 1997.
25. D. A. Turner. A new implementation technique for applicative languages. *Software – Practice and Experience*, 9(1):31–49, January 1979.
26. Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning: Introduction and Applications.* McGraw-Hill, New York, 2nd edition, 1992.