First Order Proof for Higher Order Logic Theorem Provers

Joe Hurd

Computing Laboratory
Oxford University

ESHOL Workshop
2 December 2005

Talk Plan

1. Proof Tools for Interactive Theorem Provers
   - Interactive Higher Order Logic Theorem Provers
   - First Order Proof Tools
2. Deploying First Order Provers in Higher Order Logic
   - Logical Interface
   - First Order Calculus

Interactive Theorem Provers

- Interactive theorem provers are used to construct mechanized versions of mathematical theories.
- Many applications, including program verification, formalization of mathematics, and analysis of language semantics.
- The expressivity of higher order logic makes it a popular choice to be implemented by interactive theorem provers.
  - Higher order: HOL, Isabelle, PVS, Coq.
  - First order: ACL2, Mizar.

LCF Design

- Theorem provers with an LCF design emphasize logical soundness.
  - Possibly at the cost of efficiency of execution.
- Bad News for Proving: Every theorem (and intermediate lemma) must be constructed by functions implementing the primitive rules of the logic.
- Good News for Proving: A full programming language is provided to automate common patterns of reasoning.
- In practice an LCF design rarely gets in the way of the user.
  - Some proof tools may take longer because of it,
  - but the resulting theorems are high assurance.

Interactive Proof: A How To

How to prove a statement S in an interactive theorem prover:

1. Set up S as an initial goal.
2. Select an automatic tactic that reduces the top goal to a set of simpler subgoals.
3. Go back to step 2 until all subgoals have been proved.

Tactics

- Automatic tactics are “little engines of proof” that reduce goals using primitive rules and simpler tactics.
- They can be low level for precise work, such as reducing the goal $A \wedge B$ to the set of subgoals $\{A, B\}$.
- Or they can be high level, such as a decision procedure that proves all Presburger arithmetic formulas.
- Why not embed a first order prover inside an automatic tactic?

First Order Provers

- Modern resolution provers are powerful tools.
  - Examples: Vampire, E, Spass, Gandalf.
- Their design emphasizes coverage and speed of execution.
  - Possibly at the cost of soundness.
  - Proofs found by a first order prover must be replayed by the LCF kernel to become theorems of higher order logic.
- Many first order provers are optimized for problems in the TPTP collection, from which the annual competition problems are drawn.
  - Larry Paulson has been contributing problems into TPTP derived from Isabelle subgoals.

First Order Logic Calculi

- Resolution was invented by Alan Robinson in the 1960s, and provers have been getting better ever since.
- Not just Moore's law! Many redundant inferences have been eliminated from the first order logic calculus.
- Ordered paramodulation has made a big improvement in the handling of equality.
  - Equality reasoning plays a part in most goals of higher order logic.

Previous Combinations

This is not a new idea!

- 1991 FAUST in HOL
- 1994 SEDUCT in LAMBDA
- 1996 MESON in HOL
- 1998 3TAP in KIV
- 1999 blast in Isabelle
- 1999 Gandalf in HOL
- 2000 Bliksem in Coq
- 2002 Metis in HOL

MESON In HOL

- Before Metis came along, `MESON_TAC` was the only first order proof tool in HOL.
  - Based on the model elimination calculus.
  - Added to HOL in 1996 by John Harrison.
- In 2002, building the core distribution of HOL used `MESON_TAC` to prove 1779 subgoals:
  - A further 2024 subgoals in the examples.
- Clearly the kind of tool that users want.
  - And this is despite the fact that `MESON_TAC` is weak on equality reasoning (equality is axiomatized).

Gandalf In HOL

- GANDALF_TAC is a HOL tactic that calls GANDALF.
  - Socket communications between HOL and GANDALF.
  - Added to HOL in 1999.
- The first-order calculus is powerful,
  and the C implementation is speedy.
- But there is a lot of infrastructure to maintain,
  and hard to tailor the first-order prover for HOL goals.
- GANDALF_TAC is obsolete today...
  ...but maybe it was ahead of its time?

Proof With A First Order Tactic: A How To

Here's how to prove the higher order logic subgoal g:

1. Convert the negation of g to CNF; this results in a HOL theorem of the form

$$\vdash \neg g \iff \exists \vec{a}.\ (\forall \vec{v_1}.\ c_1) \land \cdots \land (\forall \vec{v_n}.\ c_n) \tag{1}$$

2. Skolemize and map each HOL term c_i to first-order logic:

$$C = \{C_1, \ldots, C_n\}$$

3. The first-order prover runs on C, and finds a refutation ρ.
4. The refutation ρ is translated to a HOL proof of the theorem

$$\{(\forall \vec{v_1}.\ c_1), \ldots, (\forall \vec{v_n}.\ c_n)\} \vdash \bot \tag{2}$$

5. Use theorems (1) and (2) to derive $\vdash g$.

Normalization: The Problem With CNF

- Resolution provers accept input problems in CNF
- But sometimes converting terms to CNF makes their size explode:

$$\mathrm{CNF}\left(\begin{array}{l}(a_0 \wedge a_1 \wedge a_2 \wedge a_3) \vee (b_0 \wedge b_1 \wedge b_2 \wedge b_3) \vee \\ (c_0 \wedge c_1 \wedge c_2 \wedge c_3) \vee (d_0 \wedge d_1 \wedge d_2 \wedge d_3)\end{array}\right)$$

$$=$$

$$\begin{array}{c}(a_3 \vee b_3 \vee c_3 \vee d_0) \wedge (a_2 \vee b_3 \vee c_3 \vee d_0) \wedge \\ (a_1 \vee b_3 \vee c_3 \vee d_0) \wedge (a_0 \vee b_3 \vee c_3 \vee d_0) \wedge \\ \ldots \text{992 more atoms} \ldots \\ (a_0 \vee b_3 \vee c_3 \vee d_3) \wedge (a_1 \vee b_3 \vee c_3 \vee d_3) \wedge \\ (a_2 \vee b_3 \vee c_3 \vee d_3) \wedge (a_3 \vee b_3 \vee c_3 \vee d_3)\end{array}$$

Definitional CNF

Definitional CNF guarantees the size of normalized terms will be linear in the size of original terms:

$$\mathrm{DEF\_CNF}\left(\begin{array}{l}(a_0 \land a_1 \land a_2 \land a_3) \lor (b_0 \land b_1 \land b_2 \land b_3) \lor \\ (c_0 \land c_1 \land c_2 \land c_3) \lor (d_0 \land d_1 \land d_2 \land d_3)\end{array}\right)$$

$$=$$

$$\begin{array}{l}\exists\, v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}. \\ \quad (v_{11} \lor \neg d_0 \lor \neg v_{10}) \land (v_{10} \lor \neg v_{11}) \land (d_0 \lor \neg v_{11}) \land \\ \quad (v_{10} \lor \neg d_1 \lor \neg v_9) \land (v_9 \lor \neg v_{10}) \land (d_1 \lor \neg v_{10}) \land \\ \quad \qquad \ldots \text{59 more atoms} \ldots \\ \quad (v_0 \lor \neg v_1) \land (a_1 \lor \neg v_1) \land (v_0 \lor \neg a_2 \lor \neg a_3) \land \\ \quad (a_3 \lor \neg v_0) \land (a_2 \lor \neg v_0) \land (v_2 \lor v_5 \lor v_8 \lor v_{11})\end{array}$$

Definitional CNF by Inference

- Given an input term t, it's easy to generate the definitional CNF normalized term t'.
- This allows a fast oracle implementation of normalization into definitional CNF:

$$\{ORACLE\_SAYS\} \vdash t \iff t'$$

- Require a HOL proof that t and t' are logically equivalent:

$$\vdash t \iff t'$$

- This requires additional implementation effort and a slower proof tool.
  - A rare case where the LCF design of HOL gets in the way.

Logical Interface

- Another source of incompleteness is the logical interface between higher and first order logic.
- Cannot hope to be complete, but it's annoying if the tactic fails on 'simple' goals like these:

$$\vdash \exists x.\, x$$
$$\vdash P\ (\lambda x.\, x) \land Q \implies Q \land P\ (\lambda y.\, y)$$

Logical Interface

- Can program versions of first-order calculi that work directly on HOL terms.
  - But types (and λ's) add complications;
  - and then the mapping from HOL terms to first-order logic is hard-coded.
- Would like to program versions of the calculi that work on standard first-order terms, and have someone else worry about the mapping to HOL terms.
  - Then coding is simpler and the mapping is flexible;
  - but how can we keep track of first-order proofs, and automatically translate them to HOL?

First-order Logical Kernel

Use the ML type system to create an LCF-style logical kernel for clausal first-order logic:

```
signature Kernel = sig
  (* An ABSTRACT type for theorems *)
  eqtype thm

  (* Destruction of theorems is fine *)
  val dest_thm : thm → formula list × proof

  (* But creation is only allowed by these primitive rules *)
  val AXIOM    : formula list → thm
  val REFL     : term → thm
  val ASSUME   : formula → thm
  val INST     : subst → thm → thm
  val FACTOR   : thm → thm
  val RESOLVE  : formula → thm → thm → thm
  val EQUALITY : formula → int list → term → bool → thm → thm
end
```

Making Mappings Modular

The logical kernel keeps track of proofs, and allows the HOL mapping to first-order logic to be modular:

```
signature Mapping =
sig
  (* Mapping HOL goals to first-order logic *)
  val map_goal : HOL.term → FOL.formula list

  (* Translating first-order logic proofs to HOL *)
  type Axiom_map = FOL.formula list → HOL.thm
  val translate_proof : Axiom_map → Kernel.thm → HOL.thm
end
```

Implementations of `Mapping` simply provide HOL versions of the primitive inference steps in the logical kernel, and then *all* first-order theorems can be translated to HOL.

Type Information?

- It is not necessary to include type information in the mapping from HOL terms to first-order terms/formulas.
- Principal types can be inferred when translating first-order terms back to HOL.
  - This wouldn't be the case if the type system was undecidable (e.g., the PVS type system).
- But for various reasons the untyped mapping occasionally fails.
  - Examples coming up.

Four Mappings

Metis includes four mappings from HOL to first-order logic.

Their effect is illustrated on the HOL goal $n < n + 1$:

Mapping	**First-order formula**
first-order, untyped	$n < n + 1$
first-order, typed	$(n : \mathbb{N}) < ((n : \mathbb{N}) + (1 : \mathbb{N}) : \mathbb{N})$
higher-order, untyped	$\uparrow ((< \,.\, n) \,.\, ((+ \,.\, n) \,.\, 1))$
higher-order, typed	

$$\uparrow (((< : \mathbb{N} \to \mathbb{N} \to \mathbb{B}) \,.\, (n : \mathbb{N}) : \mathbb{N} \to \mathbb{B}) \,.\, (((+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}) \,.\, (n : \mathbb{N}) : \mathbb{N} \to \mathbb{N}) \,.\, (1 : \mathbb{N}) : \mathbb{N}) : \mathbb{B})$$

Mapping Efficiency

- Effect of the mapping on the time taken by model elimination calculus to prove a HOL version of Łoś's 'nonobvious' problem:

Mapping	untyped	typed
first-order	1.70s	2.49s
higher-order	2.87s	7.89s

- These timing are typical, although 2% of the time higher-order, typed does beat first-order, untyped.
- We run in untyped mode, and if an error occurs during proof translation then restart search in typed mode.
    - Restarts 17+3 times over all 1779+2024 subgoals.

Mapping Coverage

higher-order $\surd$ first-order $\times$

$$\vdash \forall f, s, a, b.\ (\forall x.\ f\ x = a) \land b \in \mathrm{image}\ f\ s \implies (a = b)$$

(f has different arities)

$$\vdash \exists x.\ x$$

(x is a predicate variable)

$$\vdash \exists f.\ \forall x.\ f\ x = x$$

(f is a function variable)

typed $\surd$ untyped $\times$

$$\vdash \mathrm{length}\ ([\,] : \mathbb{N}^*) = 0 \land \mathrm{length}\ ([\,] : \mathbb{R}^*) = 0 \implies \mathrm{length}\ ([\,] : \mathbb{R}^*) = 0$$

(indistinguishable terms)

$$\vdash \forall x.\ \mathrm{S}\ \mathrm{K}\ x = \mathrm{I}$$

(extensionality applied too many times)

$$\vdash (\forall x.\ x = c) \implies a = b$$

(bad proof via $\top = \bot$)

Equality And Completeness

- Suppose the higher order, typed mapping is used.
- Any λ-terms remaining after normalization are translated into combinators:

$$\begin{array}{l} P\ (\lambda x.\, x) \wedge Q \implies Q \wedge P\ (\lambda y.\, y) \\ \leadsto P\ I \wedge Q \implies Q \wedge P\ I \end{array}$$

- The definitions for the combinators are added as axioms.
- The following boolean equality theorems are also added:

$$\begin{array}{l} \vdash \top \qquad \vdash \neg\bot \\ \vdash \forall x, y.\ \neg x \vee (x \neq y) \vee y \\ \vdash \forall x, y.\ x \vee (x = y) \vee y \\ \vdash \forall x, y.\ \neg x \vee (x = y) \vee \neg y \end{array}$$

- Question: what is the exact coverage of this tactic?

First-Order Calculi

- Implemented ML versions of several first-order calculi.
  - Model elimination; resolution; the delta preprocessor.
  - Trivial reduction to our first-order primitive inferences.
- Can run them simultaneously using time slicing.
  - They cooperate by contributing to a central pool of unit clauses.
- Used HOL subgoals to guide the overall design.
  - For example, the focus on equality reasoning and fairly small clause sets.
- Used the TPTP problem collection to tune the parameters.
  - As a standalone prover, it comes mid-table when run on the problems drawn for two previous CASCs.

Model Elimination

- Similar search strategy (but not identical!) to `MESON_TAC`.
  - Equality is axiomatized.
- Incorporated three major optimizations:
  - Ancestor pruning (Loveland).
  - Unit lemmaizing (Astrachan and Stickel).
  - Divide & conquer searching (Harrison).
- Unit lemmaizing gave a big win.
  - The logical kernel made it easy to spot unit clauses.
  - Surprise: divide & conquer searching can occasionally prevent useful unit clauses being found!

Resolution

- Implements ordered resolution and ordered paramodulation.
- Powerful equality calculus allows proofs way out of `MESON_TAC`'s range:

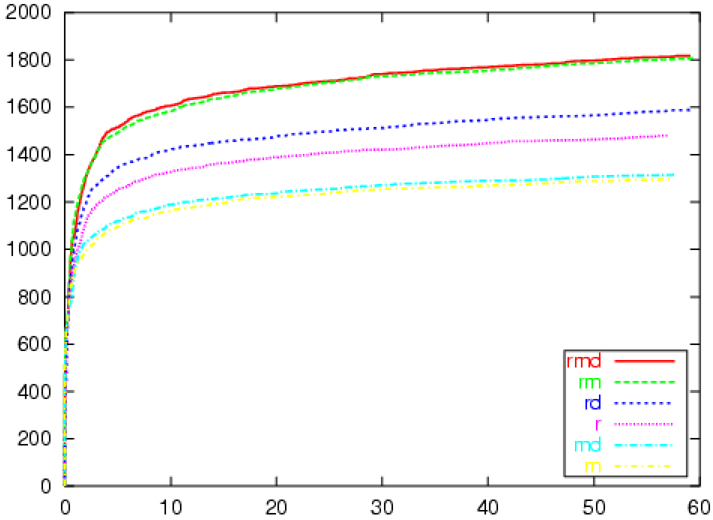
```
``(!x y. x*y = y*x) /\
  (!x y z. x*y*z = x*(y*z)) ==>
  a*b*c*d*e*f*g*h*i = i*h*g*f*e*d*c*b*a``
```

- Had to tweak it for HOL in two important ways:
  - Avoid paramodulation into a typed variable.
  - Sizes of clauses shouldn't include types.

Delta Preprocessor

- Schumann's idea: perform *shallow resolutions* on clauses before passing them to model elimination prover.
- Our version: for each predicate P/n in the goal, use model elimination to search for unit clauses of the form $P(X_1, \ldots, X_n)$ and $\neg P(Y_1, \ldots, Y_n)$.
- Doesn't directly solve the goal, but provides help in the form of unit clauses.

Evaluation on TPTP v2.4.1



Current Work: Finite Models

- Slaney proposed using unsatifiability in a finite model as a clause weighting strategy.
- Slaney used finite models found with a constraint solver, but a positive effect can be observed just using random models.
- For a first order prover being used as a higher order logic tactic, it is possible to tailor make finite models that satisfy important theorems.
    - For example, the natural numbers modulo n satisfy most of Peano's axioms.
- Preliminary experiments have shown this to be an effective strategy, and it costs very little to randomly test clauses for satisfiability.

Summary

- Have given a tour of combining first order provers and interactive higher order logic theorem provers.
    - Focused on the problems that can occur at each step, and techniques for solving them.
- Moral: there are many interesting design choices to be made at the interface between the logics.
- The time is ripe for a successful combination of higher order logic theorem provers and first order provers.