Automatic First-Order Proof in HOL

Joe Hurd
joe.hurd@cl.cam.ac.uk

University of Cambridge

Contents

- **Introduction**
- Logical Interface
- First-Order Calculi
- Comparison with `MESON_TAC`.
- Conclusion

Introduction

1. Why does HOL need automatic first-order proof?
2. What's wrong with `MESON_TAC`?
3. What's wrong with `GANDALF_TAC`?
4. What's new in this system?

Automatic First-Order Proof: Why?

Consider the following HOL subgoal:

```
...
1 subgoal:
> val it =
   (!P. (!n. (!m. m < n ==> P m) ==> P n) ==> !n. P n) ==>
    !P. P 0 /\ (!n. P n ==> P (SUC n)) ==> !n. P n
  : goalstack

-  ???
```

Automatic First-Order Proof: Why?

First, identify relevant lemmas:

```
...
1 subgoal:
> val it =
   (!P. (!n. (!m. m < n ==> P m) ==> P n) ==> !n. P n) ==>
    !P. P 0 /\ (!n. P n ==> P (SUC n)) ==> !n. P n
  : goalstack

- [LESS_SUC_REFL, num_CASES];
> val it =
   [|- !n. n < SUC n,
    |- !m. m = 0 \/ ?n. m = SUC n]
   : thm list

- ???
```

Automatic First-Order Proof: Why?

Proof 1: The HOL guru way.

```
...
1 subgoal:
> val it =
   (!P. (!n. (!m. m < n ==> P m) ==> P n) ==> !n. P n) ==>
    !P. P 0 /\ (!n. P n ==> P (SUC n)) ==> !n. P n
  : goalstack


- e (DISCH_THEN (fn t => NTAC 2 STRIP_TAC THEN MP_TAC (Q.ID_SPEC t))
     THEN DISCH_THEN MATCH_MP_TAC
     THEN (Cases THEN1 ASM_REWRITE_TAC [])
     THEN DISCH_THEN (MP_TAC o Q.SPEC ‘n’‘)
     THEN ASM_REWRITE_TAC [LESS_SUC_REFL]);

OK..

Goal proved.
|- (!P. (!n. (!m. m < n ==> P m) ==> P n) ==> !n. P n) ==>
   !P. P 0 /\ (!n. P n ==> P (SUC n)) ==> !n. P n
```

Automatic First-Order Proof: Why?

Proof 2: A simpler approach.

```
...
1 subgoal:
> val it =
   (!P. (!n. (!m. m < n ==> P m) ==> P n) ==> !n. P n) ==>
    !P. P 0 /\ (!n. P n ==> P (SUC n)) ==> !n. P n
  : goalstack


- e (METIS_TAC [LESS_SUC_REFL, num_CASES]);

OK..
metis: m-0-1-2-3-4-5-6r|*|0+7x0+0+0+0+0+0+0+0+0+1+3+1+0+0+
0+3+0+2+2+4+2+0+4+1x2+3+#

Goal proved.
|- (!P. (!n. (!m. m < n ==> P m) ==> P n) ==> !n. P n) ==>
   !P. P 0 /\ (!n. P n ==> P (SUC n)) ==> !n. P n
```

What's Wrong with `MESON_TAC`?

- `MESON_TAC` is the existing first-order prover in HOL.
  - Based on the model elimination calculus.
  - Added to HOL in 1996 by John Harrison.
- Today, building the core distribution of HOL uses `MESON_TAC` to prove 1779 subgoals:
  - Up from 1428 on 7 June (the Kananaskis-1 release).
  - A further 2024 subgoals in the examples.
- Clearly the kind of tool that users want.

What's Wrong with MESON_TAC?

- `MESON_TAC` doesn't support boolean variables;
  - ``` ``?x. x`` ```
- doesn't treat λ-terms properly;
  - ``` ``p (\x. x) /\ q ==> q /\ p (\y. y)`` ```
- isn't completely robust;
  - ```
    ``~q c /\ (!x. q x ==> ((x = c) \/ (p ((=) x)))) ==>
      !x. q x ==> (p ((=) x))``
    ```
- and implements a weak calculus for equality.
  - ```
    ``(!x y. x * y = y * x) /\ (!x y z. x * y * z = x * (y * z)) ==>
      a * b * c * d * e * f = f * e * d * c * b * a``
    ```
- What we'd like is a beefed up version of `MESON_TAC`.

What's Wrong with GANDALF_TAC?

- GANDALF_TAC is a HOL tactic that calls GANDALF.
  - GANDALF won the CADE ATP competition in 1998.
- Socket communications between HOL and GANDALF.
  - Michael Norrish's Prosper Plug-in Interface made this easy.
- 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?

What's New in This System?

- Not much! (Mainly an engineering challenge.)
  - Robustly mapping formula between higher-order and first-order logic.
  - Implementing efficient first-order calculi in ML.
- Main novelty: a clean logical interface between HOL and first-order logic.

Contents

- Introduction
- **Logical Interface**
- First-Order Calculi
- Comparison with `MESON_TAC`.
- Conclusion

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.
    - We'll see examples of this later.

Four Mappings

We have implemented 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 $\times$

$$\vdash \quad \forall f, s, a, b.\ (\forall x.\ f\ x = a)\ \wedge\ b \in \mathsf{image}\ f\ s\ \Rightarrow\ (a = b)$$

(f has different arities)

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

(x is a predicate variable)

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

(f is a function variable)

typed $\surd$ untyped $\times$

$$\vdash \quad \mathsf{length}\ ([\,] : \mathbb{N}^*) = 0\ \wedge\ \mathsf{length}\ ([\,] : \mathbb{R}^*) = 0\ \Rightarrow\ \mathsf{length}\ ([\,] : \mathbb{R}^*) = 0$$

(indistinguishable terms)

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

(extensionality applied too many times)

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

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

Contents

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 the TPTP problem set for most of the tuning.
  - Verified correlation between performance on TPTP and performance on HOL subgoals.

Model Elimination

- Similar search strategy (but not identical!) to `MESON_TAC`.
- 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 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:

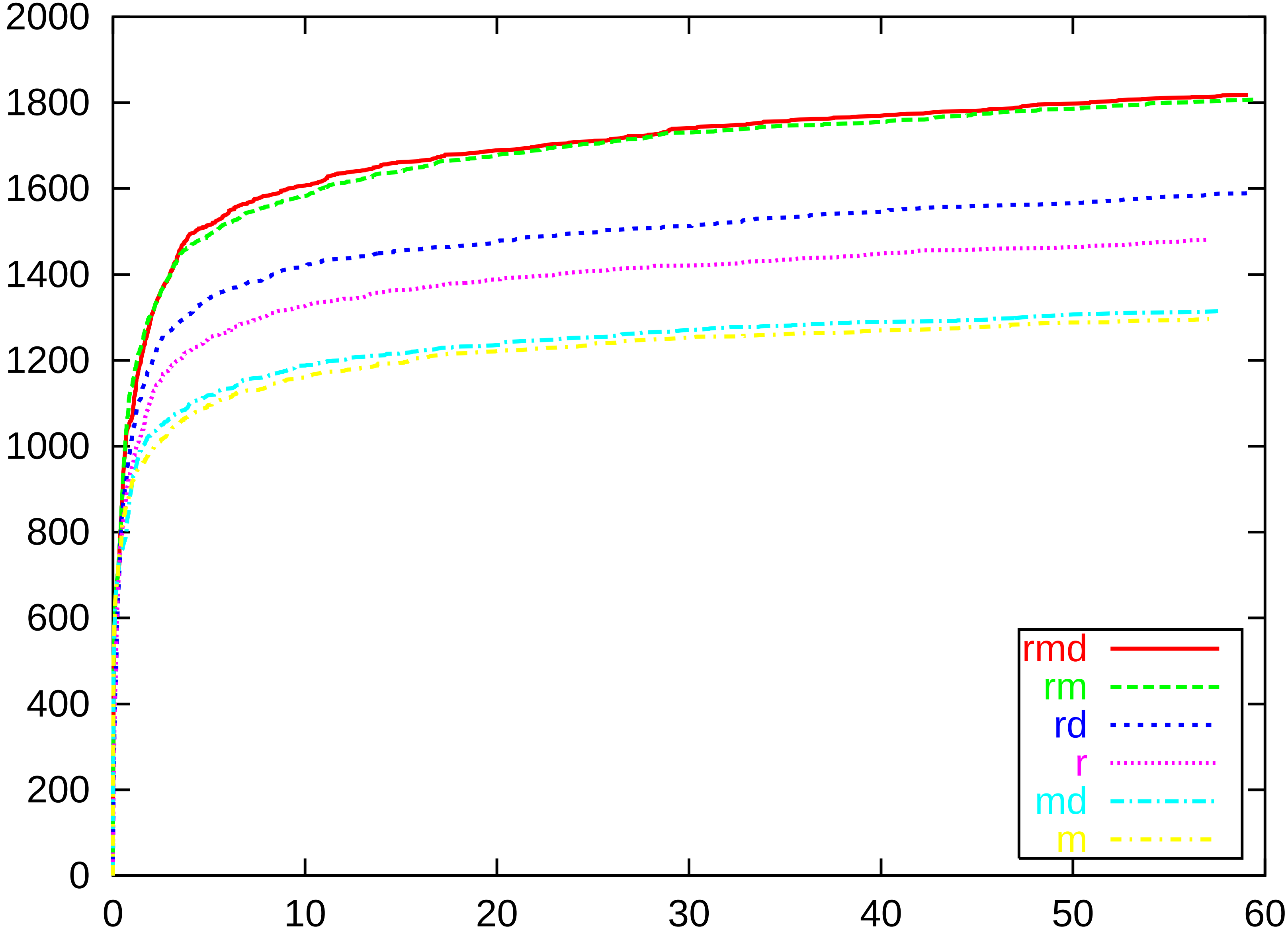
```
``(!x y. x*y = y*x) /\
  (!x y z. x*y*z = x*(y*z)) ==>
  a*b*c*d*e*f*g*h*i = i*h*g*f*e*d*c*b*a``
```

- 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.

TPTP Evaluation



TPTP Evaluation

Total "unsatisfiable" problems in TPTP = 3297

	rmd	rm	rd	r	md	m	total
rmd	$*$	$+20$ 95.0%	$+238$ 99.5%	$+351$ 99.5%	$+575$ 99.5%	$+591$ 99.5%	1819
rm	$\underline{+11}$	$*$	$+231$ 99.5%	$+338$ 99.5%	$+575$ 99.5%	$+591$ 99.5%	1811
rd	$\underline{+10}$	$\underline{+12}$	$*$	$+114$ 99.5%	$+558$ 99.5%	$+571$ 99.5%	1592
r	$\underline{+14}$	$\underline{+10}$	$\underline{+5}$	$*$	$+549$ 99.5%	$+562$ 99.5%	1483
md	$\underline{+72}$	$\underline{+81}$	$\underline{+283}$	$\underline{+383}$	$*$	$+21$ 99.5%	1316
m	$\underline{+69}$	$\underline{+78}$	$\underline{+277}$	$\underline{+377}$	$\underline{+2}$	$*$	1297

Contents

- Introduction
- Logical Interface
- First-Order Calculi
- **Comparison with `MESON_TAC`.**
- Conclusion

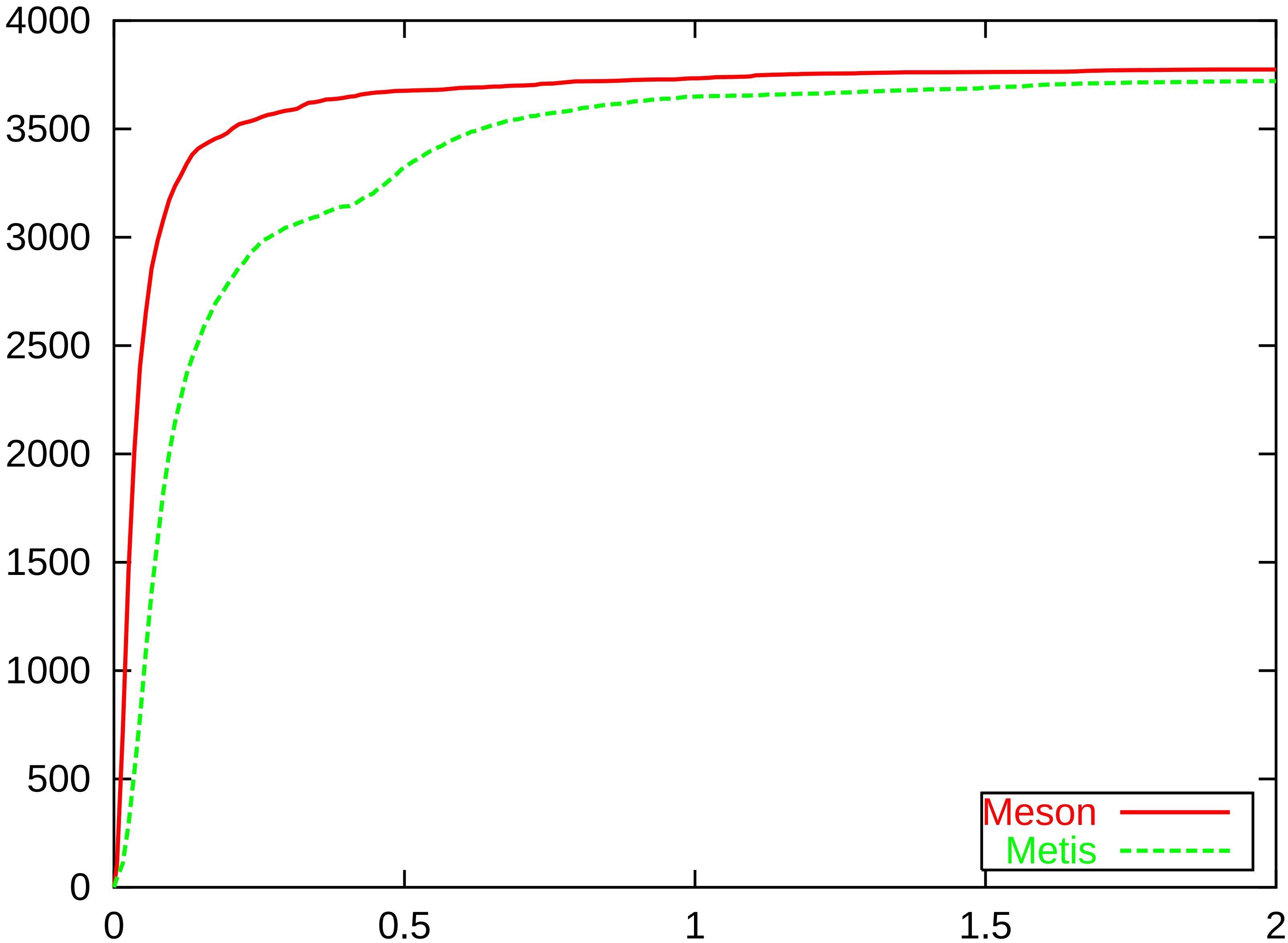
Comparison with MESON_TAC

```
Total subgoals:       1779 + 2024 = 3803
Proved by MESON_TAC: 1779 + 2017 = 3796
Proved by METIS_TAC: 1774 + 2007 = 3781

prob_53(0.02) prob_44(0.02) int_arith_139(0.09)
DeepSyntax_47(0.11) Omega_13(0.11) euclid_8(0.2)
measure_138(0.23) MachineTransition_0(0.29) nc_6(0.38
prob_169(0.39) prob_170(0.42) fol_1(0.8)
measure_86(0.93) Omega_71(1.78) fol_2(7.63)

TIME DIFFERENCE
Arithmetic mean: 0.30s
Geometric mean:  318%
```

HOL Evaluation



Contents

Conclusions

- Use `METIS_TAC` in your HOL proofs today!
  - Just do `load "metisLib"; open metisLib;` to make `METIS_TAC` and `METIS_PROVE` available.
- However, it's not the right time to retire `MESON_TAC`.
  - Given the fragile nature of first-order provers, it's quite useful to have two complementary tactics.
- Relied on the logical interface to abstract away (almost) all the details of higher-order logic.
  - Proof logging is simple in any first-order calculus.
  - Refutations are automatically lifted to HOL.
- More re-implementation than research up to this point, but now there is plenty of scope for original work that can be done in HOL.

Future Work

- Specialize first-order prover to create *point tools*:
  - Simple arithmetic reasoning.
  - Support predicate subtyping via always-fire rules.
  - Decision procedure for theories such as `finite_map`.
- Would really like to store theorems, so the user doesn't have to find the right lemmas each time.
- Improved treatment of combinators at first-order level (pattern unification?).
- Use the interface to create a new link to 'industrial strength' first-order provers.
- More powerful first-order calculus: Knuth-Bendix completion, semantic constraints, etc, etc, . . .