Interactive Theorem Proving

Lecture 1.5

Cezary Kaliszyk (VO)
Vincent van Oostrom (PS)

October 11, 2016

Administration

Grading

- Homeworks + Performance (50%)
- Bigger Proof
- System Implementation
- Presentation

Proseminar content

- HOL Light introduction
- Kernel, rules, subgoal-package, tactics
- Type introduction, quotients, inductive
- Exercises for λP, $\lambda 2$
- Curry-Howard, BHK
- Logical Frameworks (LF, Pure)
- Proving properties modulo α
- Presentations

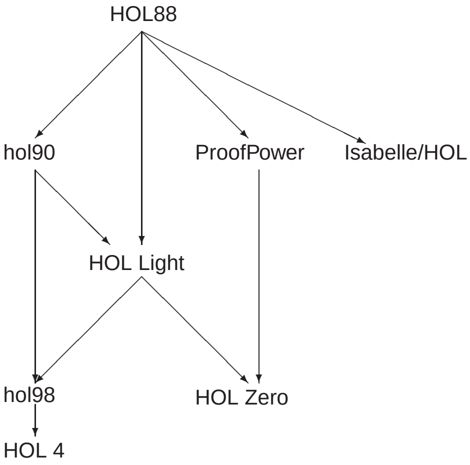
HOL Light

- Member of the HOL family of provers
  - Mike Gordon's original HOL system developed in the 80s
- LCF-style proof checker
  - Simply typed lambda calculus (polymorphic)
  - $+$ Classical higher-order logic
- Simple foundation
  - Minimal (uncluttered) implementation
- OCaml

LCF-style theorem proving

- Edinburgh LCF 1979
- Small set of simple inference rules
  - All proofs are reduced to this set
- Implemented as functions in a programming language
  - The power of the underlying programming language makes the approach practical
- HOL Light is one of the more radical LCF provers
  - Very few simple rules
  - Bigger proofs may expand to millions or billions of inferences

The HOL family DAG



2

(By John Harrison)

Simplicity of HOL Light

Close to the programming language top-level

- Easy to program
- Easy to extend
- Easy to experiment with new ideas
    - MMode [Harrison'96, Giero'04, Wiedijk'08]
    - Logical Foundations [Voelker'07, Fleuriot'12]
    - Architectures [Wiedijk'09]
    - Machine Learning Premise Selection [K., Urban]

However:

- Interface is primitive (spartan)
- Not user-friendly

HOL Light's use

- Analysis and Number Theory
  - Multivariate Analysis (for Flyspeck)
- Formal verification of hardware and software
  - Intel's floating point verification
  - HOL in HOL
- Algebra is less convenient
- Formalization of algorithms more limited
  - Only simple function definitions
  - No co-induction

Interesting Results

- Kepler conjecture
- Jordan curve theorem
- Prime number theorem
- Radon's theorem
- ...

HOL types

- Similar to OCaml types
  - (Simply typed lambda calculus with parametric polymorphism)
- A theorem can talk about (α)*list*
  - Inference rules allow instantiating the α to other types

```
type hol_type =
   Tyvar of string
 | Tyapp of string * hol_type list;;
```

Two primitive types:

```
let the_type_constants = ref ["bool",0; "fun",2];;
```

Then adding of axiomatic types and typedef.

HOL Terms

Terms of simply typed lambda calculus

```
type term =
   Var of string * hol_type
 | Const of string * hol_type
 | Comb of term * term
 | Abs of term * term;;
```

Type information only at variables and constants. (Exercise).

HOL Terms

Terms of simply typed lambda calculus

```
type term =
   Var of string * hol_type
 | Const of string * hol_type
 | Comb of term * term
 | Abs of term * term;;
```

Type information only at variables and constants. (Exercise).

- Abstract type and term interface allows only well typed terms

Primitive Constants

```
let the_term_constants =
  ref ["=", mk_fun_ty aty (mk_fun_ty aty bool_ty)];;
```

Again the abstract term interface makes sure that a constant is well typed.

- Constants can be introduced with definitions or axiomatically
  - (Axiom of choice)
- The type of theorems

```
type thm = Sequent (term list * term)
```

The basic inference rules (1/2)

$$\frac{}{\vdash t = t}\ \mathrm{REFL}$$

$$\frac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u}\ \mathrm{TRANS}$$

$$\frac{\Gamma \vdash s = t \quad \Delta \vdash u = v}{\Gamma \cup \Delta \vdash s(u) = t(v)}\ \mathrm{MK\_COMB}$$

$$\frac{\Gamma \vdash s = t}{\Gamma \vdash (\lambda x.s) = (\lambda x.t)}\ \mathrm{ABS}$$

$$\frac{}{\vdash (\lambda x.t)\ x = t}\ \mathrm{BETA}$$

$$\frac{}{\{p\} \vdash p}\ \mathrm{ASSUME}$$

$$\frac{\Gamma \vdash p \Leftrightarrow q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}\ \mathrm{EQ\_MP}$$

The basic inference rules (2/2)

$$\frac{\Gamma \vdash p \quad \Delta \vdash q}{(\Gamma - \{q\}) \cup (\Delta - \{p\}) \vdash p \Leftrightarrow q} \text{ DEDUCT\_ANTISYM\_RULE}$$

$$\frac{\Gamma[x_1, \ldots, x_n] \vdash p[x_1, \ldots, x_n]}{\Gamma[t_1, \ldots, t_n] \vdash p[t_1, \ldots, t_n]} \text{ INST}$$

$$\frac{\Gamma[\alpha_1, \ldots, \alpha_n] \vdash p[\alpha_1, \ldots, \alpha_n]}{\Gamma[\gamma_1, \ldots, \gamma_n] \vdash p[\gamma_1, \ldots, \gamma_n]} \text{ INST\_TYPE}$$

Guide to reading the source

- `hol.ml`: load order
- `lib.ml`: ML standard library for portability
- `fusion.ml`: the kernel
- `drule.ml`: simple derived rules
- `bool.ml`: basic boolean constants
- `tactic.ml`: subgoal package
- `simp.ml`: rewriting

Highlights of HOL Light

1. Open: Readable and higher-level. Close to abstract algorithm descriptions. Easy to investigate what happens “inside the box”.
2. Sound and Coherent: Thanks to LCF. Logically clean and comprehensible structure.
3. Extensible: Examples of decision procedures and tools.
4. Easy to connect to other systems. Clean interface. LCF ensures soundness.
5. Small and lightweight: Few MB of memory sufficient to run some challenging examples.
6. Different proof styles: Backwards and Mizar-style.
7. Special proof procedures: TAUT, Meson, Metis, ...

Summary

This Lecture

- LCF style
- HOL provers family
- HOL logic
- Proof Assistant Kernel

Next

- Typed λ-calculus
- HOL subgoal package and tactics