universität
innsbruck

Interactive Theorem Proving

Lecture & Exercises Week 2

Cezary Kaliszyk

Summary

Previous Lecture

- What is a Proof Assistant
- Uses, other tools
- Formal proof example, de Bruijn factor
- History, characteristics
- Coverage of Basic Mathematics

Today

- LCF Style
- Minimal Kernel
- HOL Light

How to make a trusted core?

LCF Style

- theorems as an abstract datatype
- basic logical inferences as functions
- no other way to create theorems (values of that type) due to strong typing

Example Rules

```
assume : term → thm
imp_elim : thm → thm → thm
```

Implement

$$\frac{}{A \vdash A}\ \text{assume} \qquad \frac{\Gamma \vdash A \to B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B}\ \text{imp\_elim}$$

LCF-style theorem proving

Starts with Edinburgh LCF 1977

Small set of simple inference rules

- All proofs are reduced to this set

Functions in a programming language

- The power of the underlying programming language makes the approach practical

HOL prover are more radical examples of LCF

- Very few simple rules
- Bigger proofs may expand to millions or billions of inferences

Timeline

1940 **Alonzo Church**
Simple Theory of Types

1972 **Robin Milner**
Stanford LCF

1985 **Larry Paulson**
Cambridge LCF

1993 **Tobias Nipkow**
Isabelle/HOL

1999 **Makarius Wenzel**
Isabelle/Isar

2011 **Isabelle/jEdit**
Prover IDE

1919 2019

1969 **Dana Scott**
Logic for Computable Functions

1973 **Standard ML**
Meta Language

1977 **Edinburgh LCF**

1986 **Isabelle**

1988 **Mike Gordon**
HOL4

1999 **John Harrison**
HOL Light

HOL Light

Member of the HOL family of provers

- Inspired by Mike Gordon’s original HOL system from the 80s

LCF-style proof checker

- Simply typed lambda calculus
- Polymorphic
- + Classical higher-order logic

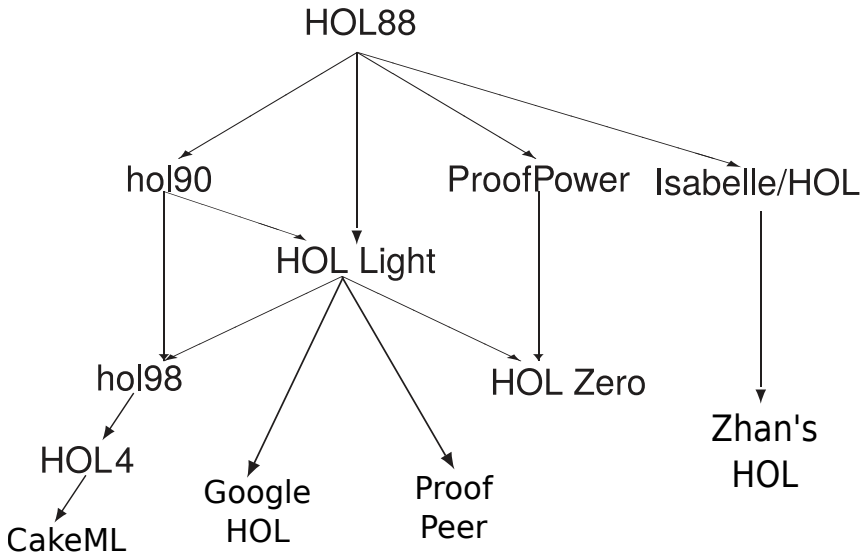
(Relatively) Simple foundation

- Minimal (uncluttered) implementation

OCaml

HOL Family Diagram

[*Harrison*]



Simplicity of HOL Light

Close to the programming language top-level

- Easy to program tactics etc

Easy to experiment with new ideas

- User Interfaces [*Harrison'96, Giero'04, Wiedijk'08*]
- Logical Foundations [*Voelker'07, Fleuriot'12*]
- Architectures [*Wiedijk'09*]
- Interaction Modes [*Tankink'14*]
- Machine Learning Premise Selection [*K., Urban*]

Disadvantages

- 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 and algorithms less convenient

- Only simple function definitions
- No co-induction

Interesting Results

- Kepler conjecture
- Jordan curve theorem
- Prime number theorem
- Radon’s theorem
- …

OCaml (in 1 slide, by example). Homework!

Syntax

```
let w = 1;;

let x =
  let y = w in
  let w = 2 in
  let z = w in
  y + z;;
```

Algebraic Datatypes (Variant Types)

- `type direction = North | East | South | West`
- `type nat = Zero | Succ of nat`
- `type 'a mylist = Nil | Cons of 'a * 'a mylist`

HOL types

Simply typed lambda calculus

- Parametric polymorphism: Similar to functional programming types
- 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];;
```

Later individuals 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

(Is this enough?) Abstract type and term interface allows only well typed terms

Primitive Constants

Only primitive constant is equality

```
let the_term_constants =
  ref ["=", `A -> A -> bool`];;
```

Abstract term interface makes sure that constants are well typed

Later choice

$$\epsilon x.P(x)$$

The type of theorems

Abstract sequents: $\Gamma \vdash t$

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

Exercises

- How do you get the type of booleans?
- How to make a variable?
- How to apply a function to a variable?

The basic inference rules (1/2)

$$\frac{}{\vdash t = t}\ \text{REFL} \qquad \frac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u}\ \text{TRANS}$$

$$\frac{\Gamma \vdash s = t \quad \Delta \vdash u = v}{\Gamma \cup \Delta \vdash s(u) = t(v)}\ \text{MK\_COMB} \qquad \frac{\Gamma \vdash s = t}{\Gamma \vdash (\lambda x.s) = (\lambda x.t)}\ \text{ABS}$$

$$\frac{}{\vdash (\lambda x.t)\, x = t}\ \text{BETA} \qquad \frac{}{\{p\} \vdash p}\ \text{ASSUME}$$

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

The basic inference rules (2/2)

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

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

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

Exercises

- Assume $a = b$. Show $f(a) = f(b)$.

Highlights of HOL Light

1. Higher-level. Close to abstract algorithm descriptions. Easy to investigate what happens.
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
- HOL Kernel

Next

- (typed) λ-calculus
- Curry-Howard isomorphism

Exercises

- Show symmetry of equality (using the HOL inference rules), namely show $A = B \vdash B = A$.
- How would you implement an LCF system that corresponds to some minimal basic propositional logic? What would be the types? Terms? Are there theorems and what would the rules to construct them be?
- Figure out how to run HOL Light
- Bonus: How would you show the S combinator $(A \to B \to C) \to (A \to B) \to A \to C$ with the basic HOL inference rules? (On paper or in a HOL system).