Interactive Theorem Proving (ITP) Course
Part XVI, XVII

Thomas Tuerk (tuerk@kth.se)

Academic Year 2016/17, Period 4

version 260dbc7 of Mon Jun 12 09:47:04 2017

Part XVI

Overview

Overview of HOL 4

- in this course we discussed the basics of HOL 4
- you were encouraged to learn more on your own in exercises
- there is a lot more to learn even after the end of the course
  - many more libraries
  - proof tools
  - existing formalisations
  - ...
- to really use HOL well, you should continue learning
- to help getting started, a short overview is presented here

HOL Bare Source Directories

The following source directories are the very basis of HOL. They are required to build `hol.bare`.

- `src/portableML` – common stuff for PolyML and MoscowML
- `src/prekernel`
- `src/0` – Standard Kernel
- `src/logging-kernel` – Logging Kernel
- `src/experimental-kernel` – Experimental Kernel
- `src/postkernel`
- `src/opentheory`
- `src/parse`
- `src/bool`
- `src/1`
- `src/proofman`

HOL Basic Directories I

On top of hol.bare, there are many basic theories and tools. These are all required for building the main hol executable.

- src/compute – fast ground term rewriting
- src/HolSat – SAT solver interfaces
- src/taut – propositional proofs using HolSat
- src/marker – marking terms
- src/q – parsing support
- src/combin – combinators
- src/lite – some simple lib with various stuff
- src/refute – refutation prover, normal forms
- src/metis – first order resolution prover
- src/meson – first order model elimination prover

HOL Basic Directories II

- src/simp – simplifier
- src/holyhammer – tool for finding Metis proofs
- src/tactictoe – machine learning tool for finding proofs
- src/IndDef – (co)inductive relation definitions
- src/basicProof – library containing proof tools
- src/relation – relations and order theory
- src/one – unit type theory
- src/pair – tuples
- src/sum – sum types
- src/tfl – defining terminating functions
- src/option – option types

HOL Basic Directories III

- src/num – numbers and arithmetic
- src/pred_set – predicate sets
- src/datatype – Datatype package
- src/list – list theories
- src/monad – monads
- src/quantHeuristics – instantiating quantifiers
- src/unwind – lib for unwinding structural hardware definitions
- src/pattern_matches – pattern matches alternative
- src/bossLib – main HOL lib loaded at start

bossLib is one central library. It loads all basic theories and libraries and provides convenient wrappers for the most common tools.

HOL More Theories I

Besides the basic libraries and theories that are required and loaded by hol, there are many more developements in HOL's source directory.

- src/sort – sorting lists
- src/string – strings
- src/TeX – exporting LaTeX code
- src/res_quan – restricted quantifiers
- src/quotient – quotient type package
- src/finite_map – finite map theory
- src/bag – bags a. k. a. multisets
- src/n-bit – maschine words

HOL More Theories II

- src/ring – reasoning about rings
- src/integer – integers
- src/llists – lazy lists
- src/path – finite and infinite paths through a transition system
- src/patricia – efficient finite map implementations using trees
- src/emit – emitting SML and OCaml code
- src/search – traversal of graphs that may contain cycles

HOL More Theories III

- src/rational – rational numbers
- src/real – real numbers
- src/complex – comples numbers
- src/HolQbf – quantified boolean formulas
- src/HolSmt – support for external SMT solvers
- src/float – IEEE floating point numbers
- src/floating-point – new version of IEEE floating point numbers
- src/probability – some propability theory
- src/temporal – shallow embedding of temporal logic
- ...

HOL Selected Examples I

The directory examples hosts many theories and libraries as well. There is not always a clear distinction between an example and a development in src. However, in general examples are more specialised and often larger. They are not required to follow HOL's coding style as much as developments in src.

- examples/balanced_bst – finite maps via balanced trees
- examples/unification – (nominal) unification
- examples/Crypto – various block ciphers
- examples/elliptic – elliptic curve cryptography
- examples/formal-languages – regular and context free formal languages
- examples/computability – basic computability theory

HOL Selected Examples II

- examples/set-theory – axiomatic formalisation of set theory
- examples/lambda – lambda calculus
- examples/acl2 – connection to ACL2 prover
- examples/theorem-prover – soundness proof of Milawa prover
- examples/PSL – formalisation of PSL
- examples/HolBdd – Binary Decision Diagrams
- examples/HolCheck – basic model checker
- examples/temporal_deep – deep embedding of temporal logics and automata

HOL Selected Examples III

- `examples/pgcl` formalisation of pGCL (the Probabilistic Guarded Command Language)
- `examples/dev` – some hardware compilation
- `examples/STE` – symbolic trajectory evalutation
- `examples/separationLogic` – formalisation of separation logic
- `examples/ARM` – formalisation of ARM architecture
- `examples/l3-machine-code` – l3 language
- `examples/machine-code` – compilers and decompilers to machine-code
- ...

Concluding Remarks

- some useful tools are a bit hidden in the HOL sources
- moreover there are developments outside the main HOL 4 sources
  - CakeML `https://cakeml.org`
- keep in touch with community to continue learning about HOL 4
  - mailing-list `hol-info`
  - GitHub `https://github.com/HOL-Theorem-Prover/HOL`
  - `https://hol-theorem-prover.org`
- if you continue using HOL, please consider sharing your work with the community

Part XVII

Other Interactive Theorem Provers

Other Interactive Theorem Provers

- at the beginning we very briefly discussed other theorem provers
- now, with more knowledge about HOL 4 we can discuss other provers and their differences to HOL 4 in more detail
- HOL 4 is a good system
- it is very well suited for the tasks required by the PROSPER project
- however, as always **choose the right tool for your task**
- you might find a different prover more suitable for your needs
- hopefully this course has enabled you to learn to use other provers on your own without much trouble

HOL 4

- based on classical higher order logic
- logic is sweet spot between expressivity and automation
- \+ very trustworthy thanks to LCF approach
- \+ simple enough to understand easily
- \+ very easy to write custom prove tools, i. e. own automation
- \+ reasonably fast and efficient
- decent automation
- – no user-interface
- – no special proof language
- – no IDE, very little editor support

HOL Omega

- mainly developed by Peter Homeier
  `http://www.trustworthytools.com/`
- extension of HOL 4
  - \+ logic extended by kinds
  - \+ allows type operator variables
  - \+ allows quantification over type variables
- \+ sometimes handy to e. g. model category theory
- – not very actively developed
- – HOL 4 usually sufficient and better supported

HOL Light

- mainly developed by John Harrison
- `https://github.com/jrh13/hol-light`
- cleanup and reimplementation of HOL in OCaml
- little legacy code
- however, still very similar to HOL 4
- \+ much better automation for real analysis
- – OCaml introduces some minor issues with trustworthiness
- – some other libs and tools of HOL 4 are missing
- – HOL 4 has bigger community

Isabelle

- Isabelle is also a descendant of LCF
- originally developed by Larry Paulson in Cambridge
  `https://www.cl.cam.ac.uk/research/hvg/Isabelle/`
- meanwhile also developed at TU Munich by Tobias Nipkow
  `http://www21.in.tum.de`
- huge contributions by Markarius Wenzel
  `http://sketis.net`
- Isabelle is a generic theorem prover
- most used instantiation is Isabelle/HOL
- other important one is Isabelle/ZF

Isabelle / HOL - Logic

- logic of Isabelle / HOL very similar to HOL's logic
  - meta logic leads to meta level quantification and object level quantification
  - + type classes
  - + powerful module system
  - + existential variables
  - ...
- Isabelle is implemented using the LCF approach
- it uses SML (Poly/ML)
- many original tools (e. g. simplifier) very similar to HOL
- focused as HOL on equational reasoning
- many tools are exchanged between HOL 4 and Isabelle / HOL
  - Metis
  - Sledgehammer
  - ...

Isabelle / HOL - Engineering

+ a lot of engineering went into Isabelle/HOL
+ it has a very nice GUI
  - IDE based on JEdit
  - special language for proofs (Isar)
  - good error messages
  - ...
+ very good automation
+ efficient implementations
+ many libraries (Archive of Formal Proof)
+ excellent code extraction
+ good documentation
+ easy for new users

Isabelle / HOL - Isar

- special proof language Isar used
- this allows to write **declarative proofs**
  - very high level
  - easy to read by humans
  - very robust
  - very good tool support
  - ...

– however, tactical proofs are not easily accessible any more
  - many intermediate goals need to be stated (declared) explicitly
  - this can be very tedious
  - tools like verification condition generators are hard to use

Isabelle / HOL - Drawbacks

+ Isabelle/HOL provides excellent out of the box automation
+ it provides a very nice user interface
+ it is very nice for new users

– however, this comes at a price
  - multiple layers added between kernel and user
  - hard to understand all these layers
  - a lot of knowledge is needed to write your own automation

– Isabelle/HOL due to focus on declarative proofs not well suited for e. g. PROSPER

Coq

- Coq is a proof assistant using the Calculus of Inductive Constructions
- inspired by HOL 88
- backward proofs as in HOL 4 used
- however, very big differences
  - much more powerful logic
  - dependent types
  - constructive logic
  - not exactly following LCF approach
- \+ good user interface
- \+ very good community support

Coq - Logic

- \+ Coq's logic is very powerful
- \+ it is very natural for mathematicians
- \+ very natural for language theory
- \+ allows reasoning about proofs
- allows to add axioms as needed
- as a result, Coq is used often to
  - formalise mathematics
  - formalise programming language semantics
  - reason about proof theory

Coq - Drawbacks

- Coq's power comes at a price
- – there is not much automation
- – proofs tend to be very long
  - they are very simple though
  - \+ comparably easy to maintain
- – Coq's proof checking can be very slow
- – when verifying programs or hardware you notice that HOL was designed for this purpose
  - need for **obvious** termination is tedious
  - missing automation
  - very slow

Summary

- there are many good theorem provers out there
- **pick the right tool for your purpose**
- the HOL theorem prover is a good system for many purposes
- for PROSPER it is a good choice
- I encourage you to continue learning about HOL and ITP
- if you have any questions feel free to contact me (**tuerk@kth.se**)