An Overview of HOL-4

Konrad Slind

University of Utah

August 19, 2008

Introduction

The HOL-4 system is an implementation of Higher Order Logic.

It provides an environment for proving theorems in a formal logic.

One may interact with HOL-4 in order to build up theories and prove theorems using existing automated reasoning tools.

HOL-4 also provides provides an environment for building custom reasoning tools.

Overview of talk

We will discuss

- The HOL logic
- Its implementation in HOL-4
- Overview of reasoning tools
- System aspects

HOL is a big system, so I won't be able to cover very much.

Logic: syntax

The HOL Logic is essentially Church's Simple Type Theory.
It is based on simple types:

$$ty ::= tyvar \mid (ty_1, \ldots, ty_n)\ \mathbf{tyop}_n$$

which are used to build lambda-calculus terms:

$$tm ::= v : ty \mid \mathbf{c} : ty \mid tm_1\ tm_2 \mid \lambda v.\ tm$$

Type operators and (term) constants are held in the signature, which can be extended by the definition principles.

Logic: syntax

The HOL Logic is essentially Church's Simple Type Theory.
It is based on simple types:

$$\mathit{ty} ::= \mathit{tyvar} \mid (\mathit{ty}_1, \ldots, \mathit{ty}_n)\ \mathbf{tyop}_n$$

which are used to build lambda-calculus terms:

$$\mathit{tm} ::= v : \mathit{ty} \mid \mathbf{c} : \mathit{ty} \mid \mathit{tm}_1\ \mathit{tm}_2 \mid \lambda v.\ \mathit{tm}$$

Type operators and (term) constants are held in the signature, which can be extended by the definition principles.

Logic: initial signature

The initial signature has the following primitive type operators:

- Booleans (**bool**)
- Functions ($ty_1 \rightarrow ty_2$)
- Individuals (**ind**)

and constants:

Equality	$=: \alpha \rightarrow \alpha \rightarrow$ **bool**	$M = N$
Implication	$\Rightarrow$: **bool** $\rightarrow$ **bool** $\rightarrow$ **bool**	$P \Rightarrow Q$
Choice	$\varepsilon : (\alpha \rightarrow$ **bool**$) \rightarrow \alpha$	$\varepsilon x.\ P\ x$

Logic: initial signature

The initial signature has the following primitive type operators:

- Booleans (**bool**)
- Functions ($ty_1 \rightarrow ty_2$)
- Individuals (**ind**)

and constants:

Equality	$=: \alpha \rightarrow \alpha \rightarrow \mathbf{bool}$	$M = N$
Implication	$\Rightarrow: \mathbf{bool} \rightarrow \mathbf{bool} \rightarrow \mathbf{bool}$	$P \Rightarrow Q$
Choice	$\varepsilon : (\alpha \rightarrow \mathbf{bool}) \rightarrow \alpha$	$\varepsilon x.\ P\ x$

Logic: semantics

HOL has a classical set-theoretic semantics

- a type with no type variables in it represents a non-empty set.
- a type with *n* distinct type variables is represented semantically by a function that takes *n* n.e. sets to a n.e. set.

In particular a function of type $\tau_1 \rightarrow \tau_2$ represents a total function from τ_1 to τ_2.

Primitive Rules of Inference

ASSUME

$t \vdash t$

REFL

$\vdash t = t$

BETA_CONV

$\vdash (\lambda v.\ M)\ t = M[v \mapsto t]$

SUBST

$$\frac{\Gamma_1 \vdash t_1 = t_1', \ldots, \Gamma_n \vdash t_n = t_n' \quad \Gamma \vdash M[t_1, \ldots, t_n]}{\Gamma \cup \Gamma_1 \cup \ldots \Gamma_n \vdash M[t_1', \ldots, t_n']}$$

ABS

$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash (\lambda v.\ t_1) = (\lambda v.\ t_2)}$$

INST_TYPE

$$\frac{\Gamma \vdash t}{\theta(\Gamma) \vdash \theta(t)}$$

MP

$$\frac{\Gamma \vdash A \Rightarrow B \qquad \Delta \vdash A}{\Gamma \cup \Delta \vdash B}$$

DISCH

$$\frac{\Gamma \vdash A}{\Gamma - \{t\} \vdash t \Rightarrow A}$$

Axioms

BOOL_CASES
$\vdash \forall t.\ t \vee \neg t$

ETA
$\vdash (\lambda x.\ M\ x) = M$

SELECT
$\vdash \forall P\ x.\ P\ x \Rightarrow P(\varepsilon y. P\ y)$

INFINITY
$\vdash \exists f : \mathbf{ind} \rightarrow \mathbf{ind}.\ \mathbf{ONE\_ONE}\ f \wedge \neg \mathbf{ONTO}\ f$

- Might be able to get away with three axioms since

$$\text{SELECT} \Rightarrow \text{BOOL\_CASES}$$

- INFINITY only used once, to build $\mathbb{N}$.

Primitive Definition Principles

Allow extending the syntax of the logic in a consistency-preserving way.

- Type definition
- Constant specification
- Constant definition

Won't go into details, but are intentionally very simple.

That's it for the primitive logic. Proved sound wrt its semantics (Gordon/Pitts).

Primitive Definition Principles

Allow extending the syntax of the logic in a consistency-preserving way.

- Type definition
- Constant specification
- Constant definition

Won't go into details, but are intentionally very simple.

That's it for the primitive logic. Proved sound wrt its semantics (Gordon/Pitts).

Basic Logic

The usual connectives and quantifiers can then be defined.

Truth	$\vdash T = ((\lambda x.\ x) = (\lambda x.\ x))$	$\mathbf{T}$
Universal	$\vdash (\forall)\ P = (\lambda x.\ \mathbf{T})$	$\forall x.\ P\ x$
Falsity	$\vdash F = \forall b.\ b$	$\mathbf{F}$
Negation	$\vdash (\neg)\ P = (P \Rightarrow \mathbf{F})$	$\neg P$
Existential	$\vdash (\exists)\ P = P(\varepsilon x.\ P\ x)$	$\exists x.\ P\ x$
Conjunction	$\vdash (\wedge)\ P\ Q = \forall t.\ (P \Rightarrow Q \Rightarrow t) \Rightarrow t$	$P \wedge Q$
Disjunction	$\vdash (\vee)\ P\ Q = \forall t.\ (P \Rightarrow t) \Rightarrow (Q \Rightarrow t) \Rightarrow t$	$P \vee Q$

Plus a few others.

We can now prove intro/elim rules and go forth to reason boldly!

Implementation

Implementors:

> *The bulk of* **HOL** *is based on code written by—in alphabetical order—Hasan Amjad, Bruno Barras, Richard Boulton, Anthony Fox, Mike Gordon, John Harrison, Peter Homeier, Joe Hurd, Ken Larsen, Tom Melham, Robin Milner, Malcolm Newey, Michael Norrish, Larry Paulson, Konrad Slind, and Don Syme.*

Many others have supplied parts of the system, bug fixes, etc.

History

- Late 70's to early 80's: LCF and ML
- Mid 80's to 1988: internal Cambridge HOL versions
- HOL88
- ICL HOL (now Proof Power)
- HOL90
- HOL98
- HOL Light
- HOL-4

Multiple implementations; but HOL logic essentially unchanged.

Kernel

HOL follows in the LCF tradition in having a small kernel implementation which encapsulates the primitive rules, axioms, and definition principles of the logic.

Arbitrary programming on top of the kernel is used to build tools.

Set of possible inference rules is the closure of the primitives under ML programming.

If the kernel is sound then programming on top can not result in soundness bugs, *e.g.*, derivation of $\vdash$ **F**.

Note similarity with micro-kernel idea from OS.

Kernel

HOL follows in the LCF tradition in having a small kernel implementation which encapsulates the primitive rules, axioms, and definition principles of the logic.

Arbitrary programming on top of the kernel is used to build tools.

Set of possible inference rules is the closure of the primitives under ML programming.

If the kernel is sound then programming on top can not result in soundness bugs, *e.g.*, derivation of $\vdash$ **F**.

Note similarity with micro-kernel idea from OS.

Kernel

HOL follows in the LCF tradition in having a small kernel implementation which encapsulates the primitive rules, axioms, and definition principles of the logic.

Arbitrary programming on top of the kernel is used to build tools.

Set of possible inference rules is the closure of the primitives under ML programming.

If the kernel is sound then programming on top can not result in soundness bugs, *e.g.*, derivation of $\vdash$ **F**.

Note similarity with micro-kernel idea from OS.

Kernel

HOL follows in the LCF tradition in having a small kernel implementation which encapsulates the primitive rules, axioms, and definition principles of the logic.

Arbitrary programming on top of the kernel is used to build tools.

Set of possible inference rules is the closure of the primitives under ML programming.

If the kernel is sound then programming on top can not result in soundness bugs, *e.g.*, derivation of $\vdash$ **F**.

Note similarity with micro-kernel idea from OS.

Kernel

HOL follows in the LCF tradition in having a small kernel implementation which encapsulates the primitive rules, axioms, and definition principles of the logic.

Arbitrary programming on top of the kernel is used to build tools.

Set of possible inference rules is the closure of the primitives under ML programming.

If the kernel is sound then programming on top can not result in soundness bugs, *e.g.*, derivation of $\vdash \mathbf{F}$.

Note similarity with micro-kernel idea from OS.

Two Kernels

- LCF/HOL88 : name-carrying terms
- HOL90 : deBruijn-like (early example of *locally nameless*)
- HOL98 : explicit substitutions (thanks to Bruno Barras)
- HOL4 : two kernels

Why?

Two Kernels

- LCF/HOL88 : name-carrying terms
- HOL90 : deBruijn-like (early example of *locally nameless*)
- HOL98 : explicit substitutions (thanks to Bruno Barras)
- HOL4 : two kernels

Why?

Comparison

Terms implemented as an abstract type, with constructors and destructors.

$$\textbf{mk\_abs} : \textbf{term} * \textbf{term} \to \textbf{term}$$
$$\textbf{dest\_abs} : \textbf{term} \to \textbf{term} * \textbf{term}$$

In a dB-style implemention $\textbf{mk\_abs}(v, M)$ must (eventually) traverse all of M replacing suitable dB indices by free variable v.

Naively, in order to build an iterated λ-abstraction $\lambda v_1 \ldots v_n.\ M$ or a quantifier iteration, $n \times \textbf{size}(M)$ work needs to be done. Partially addressed by explicit substitutions, but not completely.

Upshot: significant slowdown on terms with deeply nested quantifiers.

Comparison

In contrast, name-carrying terms have constant-time **mk_abs** and **dest_abs**.

BUT matching, α-conversion, computing free vars, *etc.* are slower.

Question: Is there an optimal implementation of the HOL prelogic?

Our "solution": have two separate kernels. Pick which one you want at system build time. The use of abstract datatypes has made this almost completely painless: no duplication of code above the kernel.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, n-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, *n*-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, n-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, *n*-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, *n*-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, *n*-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, *n*-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, *n*-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Theories

The system provides a collection of already-developed theories. (All built up definitionally.)

- Basics: booleans, pairs, sums, options, relations
- Numbers: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, fixed point, floating point, *n*-bit words.
- Sequences: lists, lazy lists, character strings
- Collections: predicate sets, multisets
- Misc. math: partial orders, monad instances, finite maps, polynomials, probability, abstract algebra, elliptic curves
- Temporal logics: (ω-automata, CTL, μ-calculus, PSL)
- Lambda calculus: chapters from Barendregt
- Program logics: Hoare logic, separation logic
- Machine models: ARM, PPC, and IA32

Represents hundreds of person-years of effort.

Reasoning Support

Advanced Definition Principles

These reduce complex definitions into simple ones, and derive (by proof) the desired formulas.

- Datatypes (Melham, Gunter, Harrison)
- Inductive relations (Melham, Harrison, Homeier)
- Recursive functions (Gordon, Melham, Slind)
- Quotients (Homeier)

Datatypes

Similar to subset of ML datatypes.

- Recursive
- Mutually Recursive
- Nested Recursive
- Mutual and Nested
- Records

Scales pretty well. Used for defining ASTs for PL in

- Owen's formalization of Ocaml
- Norrish's formalization of C++

Inductive Relations

- Recursive
- Mutually Recursive
- Infinitary premises

Scales pretty well. Used to define

- Evaluation relation for OCaml and C++
- Transition relation for TCP

Recursive Functions

- ML-style pattern-matching
- Mutual Recursion
- Nested Recursion
- Higher order recursion
- Automatic derivation of custom induction theorem
- Naive but useful termination prover

With datatype package, offers a way of doing functional programming in logic.

Automated Reasoners

- Simplification
- Evaluation
- First order proof search
- Decision procedures

Simplification

- Conditional and contextual rewriter
- Higher order matching (Miller/Nipkow style)
- Permutative rewriting (normalizing AC terms)
- User-extensible with arbitrary reasoners that deliver equality theorems (conversions and dec. procs)

Uses: interactive proof, prototyping reasoning tools

Evaluation

- Call-by-value by deductive steps
- Ground (and symbolic) evaluation using database of logic functions
- Author: Bruno Barras

Uses: custom ground evaluators; symbolic simulation

First order proof search

- METIS
- Ordered resolution
- Reduction of some higher order stuff to first order (via combinator translation)
- Author: Joe Hurd

Uses: interactive theory development

Decision procedures

- For linear fragment of $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{W}$
- For CS logics: μ-calculus, fragments of Separation logic, ...
- Coq-style partial reflection (for rings)

Uses: interactive proof, automating proof in user-defined domains

Programming Automated Reasoners

How are these things built?

- Use ML as a proof composition language for forward proof
- Tactics (subgoal decomposition, Milner)
- Conversions (equational reasoning, Paulson)

HOL4 offers APIs supporting such activities.

Importing results

Importing results from other proof systems

- Without proof (via oracle mechanism).
  - trusted BDD operations
  - interaction with ACL2
- Translating given proof object into HOL proof
  - minisat interface for tautology checking

Generating code and TeX

- Given a collection of datatypes and function definitions, the HOL user can generate SML and/or OCaml code
- Given an arbitrary theory, a TeX version can be automatically generated.

Interactive Proof

Nothing fancy.

- A simple goalstack interface for tactic-based interactive proof
- Set of emacs macros for building proof script while interacting with ML top-level
- Declarative elements integrated into tactic proof, e.g.

  $$P \ \texttt{by} \ \mathit{tactic}$$

  uses tactic to prove P in the current proof context and then adds P as a new assumption.

Documentation

HOL-4 provides online and offline documentation, although the distinction is somewhat blurred (a good thing!)

- Offline: HOL Logic, Description, Reference, Tutorial
- Online: help system for Moscow ML, HOL API, HOL theorems, *etc*.
- Online: Webpages for the same

Offline documentation

High quality typeset documentation

- Logic: formal description of syntax and semantics of HOL logic, due to Mike Gordon and Andy Pitts.
- Tutorial: range of detailed examples in using HOL
- Description: detailed explanation of embedding, theories, libraries, and reasoning tools. Not read by enough people!

Platforms

OS:

- Windows (auto-installer)
- Mac OS X
- Linux
- other Unixes (AIX, Solaris, etc)

ML:

- Moscow ML
- Poly/ML (thanks to Scott Owens)

Interesting Verification Projects

A sampling of projects that use HOL-4:

- Network semantics
- Semantics of OCaml (Owens, ESOP'08)
- Semantics of C++ (Norrish, TTVSI)
- Semantics of MP x86 (Sewell, in progress)
- Semantics of ARM (Fox)
- Compilation of functional programs to hardware (Gordon,Iyoda, Slind)
- Compilation of functional programs to assembly (Li,Myreen,Slind)
- Others

Extending HOL?

The HOL logic has proved to be quite good at modelling much mathematics and CS.
But occasionally its type system is not strong enough to capture common and useful notions.

- Monads
- Formal languages

We have resisted making ad-hoc extensions, since there are a bewildering variety of choices.

HOL extensions

Recently, some nice extensions of HOL have been proposed:

- Norbert Völcker's **HOL2P** system adds type operator variables and universal types. Implementation by revising HOL-Light in a backwards compatible way.
- Peter Homeier has been working on an extension of the HOL logic, called **HOL** $-\ \omega$. This is being implemented by revising HOL-4, also in a backwards compatible way.

Webpage

http://hol.sourceforge.net