

The HOL Theorem Prover and its Applications

T. F. Melham

Department of Computing Science
University of Glasgow
Lilybank Gardens
Glasgow, G12 8QQ
Scotland

Formal Methods

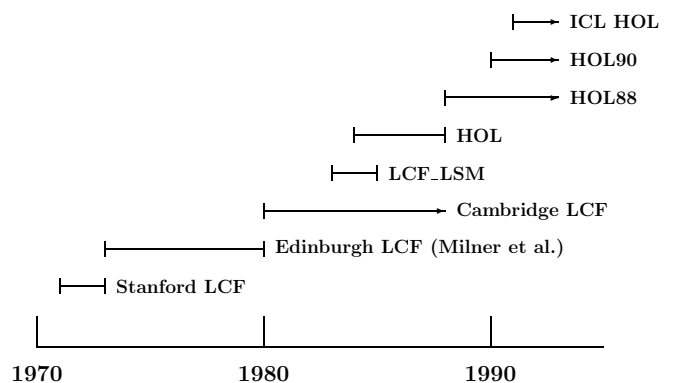
- Formal methods just means *mathematics*:
 - for precise specification of systems
 - for developing or designing systems
 - for verifying the correctness of systems
- Advantages – formal methods:
 - provide unambiguous requirements specification
 - help catch design errors
 - provide better understanding of designs
- But formal methods are *not* a panacea:
 - they can be inappropriately applied,
 - they can be expensive,
 - and they can be misunderstood.
- Effectiveness depends on:
 - thorough understanding of the technology
 - the support of good computer-based tools

What is HOL?

- HOL is an interactive program for mechanized formal reasoning using *higher order logic*.
- The HOL system provides:
 - an expressive and powerful *notation* for writing system specifications,
 - flexible and general facilities for creating formal *proofs* of properties of specifications.
- Features of the HOL system:
 - rigorous and well-understood theoretical basis
 - powerful command language (the general purpose programming language ML)
 - secure – can't prove false theorems
 - user-extendable, without compromising security
 - supports a variety of styles of reasoning
 - automates *some* low-level details of proofs

History of HOL

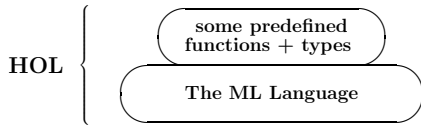
- A mature system; over 20 years research:



- Many contributors of ideas and code:
 - Stanford, Edinburgh, Cambridge, INRIA, ...
 - Substantial contributions from user community.

The Structure of HOL

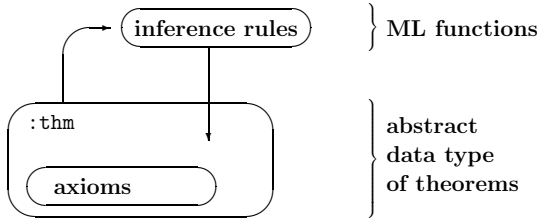
- HOL is built on top of ML:



- Arbitrary propositions are distinguished from proved theorems:

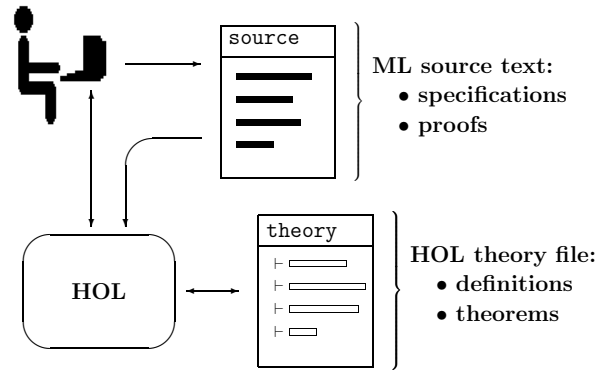
propositions	proved theorems
ML type :term	ML type :thm

- In the core of HOL we have:



How HOL is Used in Practice

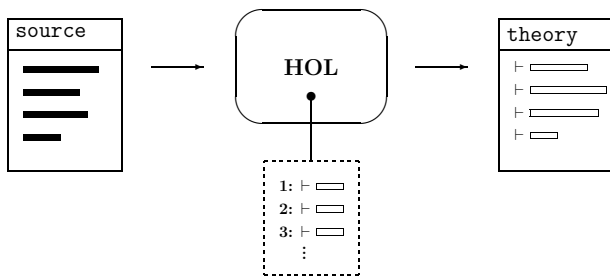
- HOL is a programming environment:
 - system commands = a programming language
 - proof = computation of theorems
- Theory-creation in the HOL system:



- Two main activities:
 - write and debug specifications in logic
 - develop (programs that generate) formal proofs

Formal Proof in HOL

- All theorems have complete formal proofs:



- The actual proof resides in the computation process itself and is ephemeral.
- Tools exist to record complete proof scripts:
 - these could be checked by independent tools,
 - but they can also be 1,000,000s of lines long!

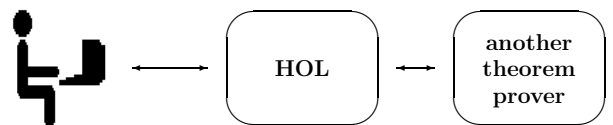
Other Modes of Use

- Hol as proof engine:



Example: interactive program refinement tool using Centaur [Thèry/Grundy].

- Hybrid theorem-proving:



Examples: links with Faust [Kumar], Voss [Joyce/Seeger], and Maple [Harrison/Thèry].

Embedding Other Formalisms

- Formalisms embedded in HOL include:
 - temporal logic (e.g. Interval Temporal Logic)
 - programming logics (e.g. Hoare logic)
 - process algebras (CCS, CSP, π -calculus)
- Example – temporal logic:

temporal logic	higher order logic
$\Box P$ (<i>always</i> P)	$\lambda t. \forall n. P(t+n)$
$\Diamond P$ (<i>sometimes</i> P)	$\lambda t. \exists n. P(t+n)$

This is an example of *shallow embedding*.

- Example – Hoare logic:

$$\frac{\vdash \{P \wedge B\} C \{P\}}{\vdash \{P\} \text{WHILE } B \text{ DO } C \{P \wedge \neg B\}}$$

$$\vdash \forall C:Com. \forall P B.$$

$$\text{Spec } (\lambda s. P s \wedge B s) C B \supset$$

$$\text{Spec } P (\text{WHILE } B C) (\lambda s. P s \wedge \neg B s)$$

This is an example of *deep embedding*.

Other Formalisms

- First-order logic:
 - some automation is possible (e.g. resolution)
 - but lacks expressive power
- Set theory:
 - very expressive formalism (usually untyped)
 - example – the Z specification language
 - but no well developed systems for *proofs* in Z
- Dependent type theory (e.g. Martin-Löf):
 - great expressive power in the type system
 - but type checking/type inference undecidable
 - also logic seems complex to non-experts

Other Systems (a selection)

- Isabelle:
 - A generic theorem prover (many logics).
 - Has a higher order logic instance.
- Nuprl:
 - Constructive type theory (Martin-Löf).
 - Program development, hardware verification.
- PVS:
 - Higher order logic with predicate subtypes.
 - Decision procedures built-in (not derived).
- IMPS:
 - Higher order logic with non-denoting terms.
 - Supports traditional mathematical reasoning.
- Boyer-Moore/Nqthm:
 - Quantifier-free, first order logic.
 - Automated proof, with some user guidance.

Advantages of HOL

- Higher order logic is:
 - general – not fixed to a single subject-matter
 - expressive – gives concise and clear specifications
 - powerful – includes most ‘ordinary’ mathematics
- The HOL system is:
 - secure – every theorem has a (correct!) proof
 - flexible – can always ‘program it’ in ML
 - extendable – can add new proof rules... safely
 - established – users, examples, manuals, support
- Some disadvantages:
 - limited automation (but this is being developed)
 - primitive interface (at least, by default)
 - development and support relies on researchers
 - fairly steep learning curve
- Course objectives:
 - familiarity with higher order logic

- thorough understanding of the ideas behind HOL
 - familiarity with the basic facilities of the system
 - understanding of certain applications techniques
 - ability to continue with self-learning
- **Course structure:**
 - mornings – lectures to introduce the ideas
 - afternoons – hands-on practical exercises
 - **General outline:**
 - Monday – higher order logic, the ML language
 - Tuesday – logic in the system, forward proof
 - Wednesday – goal-directed proof, tactics
 - Thursday – advanced features and techniques
 - Friday – applications of HOL