HOL Foundations

by Arthur Grundner
HOL Foundations

- HOL is a family of proof assistants, using a variant of higher-order logic
- HOL4 is the primary descendent, still being actively developed on: https://hol-theorem-prover.org/
- HOL is the predecessor of Isabelle
- HOL has its roots in the LCF formalism
In 1969, the LCF ('Logic for computable functions') formalism was devised by Dana Scott. The intention was to improve reasoning about recursively defined functions in denotational semantics. Denotational semantics deals with finding mathematical objects ('domains') to explain the behavior of computer programs. Published in 1993.
The language of the LCF formalism

- **Terms**: Typed λ-terms; i.e. either variables, constants, λ-abstractions or λ-applications
- **Formulae**: Predicate calculus
- **Types**: Scott Domains
In 1972, Milner, Diffie, Weyhrauch and Newey developed the proof-checker LCF at Stanford University. It was based on the LCF formalism.
Features of Stanford LCF

- "The proof-checking program is designed to allow the user interactively to generate formal proofs about computable functions and functionals over a variety of domains, including those of interest to the computer scientist for example, integers, lists and computer programs and their semantics. The user’s task is alleviated by two features: a subgoaling facility and a powerful simplification mechanism." (Robin Milner)
Shortcomings of Stanford LCF

- Storage of proofs filled up memory quickly
- Repertoire of proof commands was immutable
Edinburgh LCF

- In Edinburgh, Milner tackled the problems of Stanford LCF
- Only result of proofs, not proofs themselves, should be stored
- For full customizability, Milner developed a strictly typed programming language ML ('Meta-Language')
Features of ML

- Exception handling mechanism
- Novel polymorphic type system (a term with type variables is a single polymorphic term)
- Own abstract data type for theorems

⇒ All theorems must have been correctly deduced simply because of their type
Tactics

- A tactic is a function with
  - Input: Goal, that needs to be proven
  - Output: List of sub-goals along with a justification function

- Notation:

- Example: (induction)
Tacticals

- A tactical is a function, that can compose tactics and returns a tactic.
- Example:
  - Let S and T be tactics and 'THEN' a tactical. Then 'S THEN T' applies S to some goal and then applies T to all sub-goals produced by S.
Cambridge LCF

- Gerard Huet ported Edinburgh LCF to the Lisp dialects Le Lisp and MacLisp
- Larry Paulson then improved Huet's code
- Many features and techniques were added
- The resulting system was called Cambridge LCF due Paulson's workplace and got ported to Standard ML
HOL

- Mike Gordon – inspired by a theorem proved by Robin Milner – invented a notation called LSM ('Logic of sequential machines')
- Gordon's main interest was the formal verification of hardware
- He then combined LSM with a version of Cambridge LCF, encoded terms in predicate calculus, which resulted in HOL
- Gordon used higher-order logic to be able to adequately model hardware
HOL's logic and novelties

- The language corresponds to that of the LCF formalism with the difference, that types were interpreted as sets instead of Scott Domains.
- Higher-order logic admits quantification over sets or predicates, that are nested arbitrarily deep.
- Example of a third-order term:
  \[ \forall Q \exists R \in Q \exists f \exists x \exists y : R(f(x)) \rightarrow R(y) \]
- Two theories form the basis of HOL (bool, ind)
The theory *bool*

- Contains:
  - Primitive type 'bool'
  - Four axioms for higher-order logic
  - Three primitive constants (Equality, Implication and Choice) and some more useful but less important constants

- With these three constants we can define $\top$ (truth), $\bot$ (falsity), $\neg$ (negation), $\land$ (conjunction), $\lor$ (disjunction), $\forall$ (universal quantification), $\exists$ (existential quantification) and $\exists!$ (unique existence quantification)
The Choice- or Hilbert's ε-operator

- Let $t[x]$ be a term of type $\sigma \to \text{bool}$ with a free variable $x$
- $\varepsilon x.t[x]$ returns some $a$ in $\sigma$, such that $t[a]$ is true. If $t[a]$ is false for all $a$ in $\sigma$, then $\varepsilon x.t[x]$ denotes some unspecified element in $\sigma$
- With the Hilbert-operator, we implicitly implement the Axiom of Choice
Examples

- $\exists n. n < 5$ denotes some unspecified number below 5
- $\exists n. (n^2 = 25) \land (n \geq 0)$ denotes 5
- $\exists n. \neg (n = n)$ is some unspecified number
Four axioms in \textit{bool}

\[\vdash \forall b. (b = \top) \lor (b = \bot)\]
\[\vdash \forall b_1 \ b_2. (b_1 \Rightarrow b_2) \Rightarrow (b_2 \Rightarrow b_1) \Rightarrow (b_1 = b_2)\]
\[\vdash \forall f. (\lambda x. f \ x) = f\]
\[\vdash \forall P \ x. P \ x \Rightarrow P (\$\varepsilon \ P)\]
The theory *ind*

- Contains:
  - Primitive type 'ind' (individuals)
  - Axiom of Infinity:

\[ \vdash \exists f : ind \rightarrow ind. (\text{One}_\text{One} f) \land \neg (\text{Onto} f) \]

- The Axiom of Infinity asserts that ind denotes an infinite set (would be an impossible construction in bool)

- Axioms of bool and ind sufficient for developing standard mathematics
Inference rules in HOL

- HOL uses eight inference rules:
  - ASSUME: Assumption Introduction
  - REFL: Reflexivity
  - BETA_CONV: Beta-conversion
  - SUBST: Substitution
  - ABS: Abstraction
  - INST_TYPE: Type Instantiation
  - DISCH: Discharging an assumption
  - MP: Modus Ponens
Two inference rules

• DISCH:

\[ \frac{\Gamma \vdash t_2}{\Gamma - \{t_1\} \vdash t_1 \Rightarrow t_2} \]

• BETA_CONV:

\[ \vdash (\lambda x.t_1)t_2 = t_1[t_2/x] \]
The LCF approach in ML

• Logical inference rules are implemented as functions

• Modus Ponens as an example:

\[
\frac{\Gamma \vdash p \Rightarrow q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}
\]

• In ML:

```
val MP : thm -> thm -> thm
MP (\Gamma \vdash p \Rightarrow q) (\Delta \vdash p) = (\Gamma \cup \Delta \vdash q)
```
HOL and Set theory - Comparison

- HOL fundamentally bases on typed higher-order logic, more generally on type theory
## HOL and Set theory - Comparison

<table>
<thead>
<tr>
<th><strong>Type Theory</strong></th>
<th><strong>Set Theory</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- No standard formulation for typed higher-order logic</td>
<td>- ZFC is the foundation for mathematics as recognized by most mathematicians.</td>
</tr>
<tr>
<td>- Functions as most basic operators, in simply typed lambda calculus even the only type operator</td>
<td>- Natural numbers defined as nested sets of the empty set:</td>
</tr>
<tr>
<td>- Natural numbers defined as inductive type with two constructors:</td>
<td>{\emptyset, {\emptyset}, \emptyset, {\emptyset}, \emptyset, {\emptyset}, \emptyset, {\emptyset}, \emptyset, {\emptyset}, \ldots }</td>
</tr>
<tr>
<td>- Easy access to tools for indexing terms, structuring data, checking types</td>
<td>- Known to most mathematicians</td>
</tr>
<tr>
<td>- Proofs/Theorems often shorter and simpler</td>
<td>- Elements can belong to different sets at the same time</td>
</tr>
<tr>
<td>- Not difficult to build set theory on top of type theory.</td>
<td></td>
</tr>
<tr>
<td>- Elements can usually belong to only one type</td>
<td></td>
</tr>
</tbody>
</table>
Sources

[8] Introduction to HOL (Book) (Mike Gordon)