The HOL Logic and Tool

λ-Calculus plus Predicate Calculus.

- x, y, z — variables
- T, F, 0, 1 — constants
- $A = B$ — equality
- $\neg P,\ P \wedge Q,\ P \supset Q, \ldots$ — connectives
- $\lambda x. E$ — λ-abstraction
- $\forall x. P,\ \exists y. Q$ — logical quantifiers
- $f\,x$ — function application

What Does ‘Higher-Order’ Mean?

- There is no distinction between formulae and terms. Formulae are just boolean term.
- Predicates are, therefore, just boolean valued functions.
- Variables can range over functions (and hence predicates).
- Functions (predicates) can take functions (predicates) as arguments and return functions as results.

Example: Principle of Induction

$$\forall P. P\,0 \wedge (\forall n. P\,n \supset P\,(n+1)) \supset (\forall m. P\,m)$$

Example: Principle of Recursive Definition

$$\forall x\,f. \exists g.\, g\,0 = x \wedge (\forall n.\, g\,(n+1) = f\,(g\,n)\,n)$$

Terms of the λ-Calculus

```
TERM ::= CONSTANT
       | VARIABLE
       | TERM TERM
       | λVARIABLE. TERM
```

Constants: Written as identifiers: a, max, …, or symbols: T, +, =, ∧, ∀, …

Variables: Range over a type. Written as ordinary identifiers: x, *name*, …

Applications: $f\,x$ is the application of function f to x.
Application is left associative; e.g., $f\,x\,y$ is $(f\,x)\,y$.
Note: Some function constants are written as infixes;
e.g., 1 + 2 rather than + 1 2, but this is just a convenience.

Abstractions: $\lambda x. E$ is a function which when applied to v returns $E[v/x]$;
e.g., $(\lambda x. x+1)$ is the increment function.

λ-Abstraction and Binding

A occurrence of a variable v is *bound* if it appears in the body of a term of the form $\lambda v.$ *body*. Occurrences of a variable that are not bound are *free*.

$(\lambda x. f\,x)(\lambda y. x)$
↑ ↑
bound free

Bound variables can be renamed with the abstraction while keeping its meaning, so long as no free variables get bound (are *captured*).

$$(\lambda x. x+x+y) = (\lambda z. z+z+y) \neq (\lambda y. y+y+y)$$

Only λ binds variables, all other ‘binders’ are higher-order functions.
For example, $\forall x. P$ is just an abbreviation for $\forall(\lambda x. P)$,
where ∀ is a higher-order function defined as follows:

$$\forall f = (f = (\lambda x. \mathrm{T}))$$

Note: $\lambda x\,y. E$ can be used to abbreviate $\lambda x. (\lambda y. E)$.

Higher-Order Logic is Typed

The notation $E{:}\tau$ indicates that expression E has type τ.

The following types, and others, are available:

- $\mathbb{I}$
- $\mathbb{B}$
- $\mathbb{N}$
- $\mathbb{Z}$
- $\alpha \times \beta$
- $\alpha \rightarrow \beta$
- αlist
- αtree

A well-formed term must a have a unique type consistent with its subterms:

$$\frac{1{:}\mathbb{N} \quad x{:}\mathbb{N} \quad +{:}\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}}{(1+x){:}\mathbb{N}}$$

Why? Higher-order logic is too expressive without types!

Russel $\stackrel{\text{def}}{=} (\lambda p. \neg(p\,p))$

What is Russel Russel?

This definition is not well formed because it does not type-check.

The Five Axioms of HOL

Excluded Middle: $\forall b. (b = \mathrm{T}) \vee (b = \mathrm{F})$

Antisymmetric Implication: $\forall b\,c. (b \supset c) \supset (c \supset b) \supset (b = c)$

η-Conversion: $\forall f. (\lambda x. f\,x) = f$

Choice: $\forall P\,x. P\,x \supset P(\varepsilon P)$

Infinity: $\exists f{:}\mathbb{I} \rightarrow \mathbb{I}. (\forall x\,y. (f\,x = f\,y) \supset (x = y)) \wedge (\neg\forall x. \exists y. x = f\,y)$

The Seven Inference Rules of HOL

Assume: $\overline{P \vdash P}$

Reflect: $\overline{\vdash E = E}$

Modus Ponens: $\dfrac{\Gamma_1 \vdash P \supset Q \qquad \Gamma_2 \vdash P}{\Gamma_1, \Gamma_2 \vdash Q}$

Discharge: $\dfrac{\Gamma, P \vdash Q}{\Gamma \vdash P \supset Q}$

β-Convert: $\overline{\vdash (\lambda x. E)\,D = E[D/x]}$

Substitute: $\dfrac{\Gamma_1 \vdash D = E \qquad \Gamma_2 \vdash P[D]}{\Gamma_1, \Gamma_2 \vdash P[E]}$

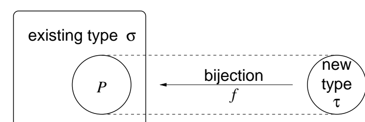
Instantiate Types: $\dfrac{\Gamma \vdash P}{\Gamma \vdash P[\tau_1, \ldots \tau_n/\alpha_1, \ldots \alpha_n]}$
Provided none of $\tau_1, \ldots \tau_n$ occur in Γ.

The Three Rules of HOL Definition

Constant Definition: $\vdash \mathrm{c}\,x_1 \ldots x_n = E$
Provided there are no free variables in $\lambda x_1 \ldots x_n. E$.

Constant Specification: $\vdash P[\mathrm{c}]$
Provided c is not already a constant and $\exists x. P[x]$.

Type Definition: $\exists f{:}\tau \rightarrow \sigma. (\forall x\,y. (f\,x = f\,y) \supset (x = y)) \wedge (\forall z. P\,z = (\exists w. z = f\,w))$



Providing τ is not already a type and $\exists x. P\,x$.

The HOL Tool and ML

HOL is a collection of definitions added to the programming language ML.

ML is functional:

```
- fun reverse [] = []
   | reverse (x::xs) = (reverse xs)⊃[x];
> val reverse = fn : α list → α list
- reverse [1, 2, 3];
> val it = [3, 2, 1] : int list
```

ML is higher order:

```
- fun map f [] = []
   | map f (x::xs) = (f x)::(map f xs);
> val map = fn : (α → β) → α list → β list
- map reverse [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
> val it = [[3, 2, 1], [6, 5, 4], [9, 8, 7]] :
  int list list
```


Inductive Data-types in ML

Users can define inductive data-types and recursive functions on them.

```
- datatype α tree = LEAF of α
                  | BRANCH of α tree × α tree;
> datatype α tree
  con LEAF = fn : α → α tree
  con BRANCH = fn : α tree × α tree → α tree

- fun fringe (LEAF x) = [x]
   | fringe (BRANCH (t1,t2)) = (fringe t1)⊃(fringe t2);
> val fringe = fn : α tree → α list
```


Using ML as a Meta-Language

Types and terms of the HOL logic are represented as ML inductive data-types.

```
- val sad_but_true = Term `∀x:𝕀. Man x ⊃ Mortal x`;
> val sad_but_true = `∀x. Man x ⊃ Mortal x` : Term.term

- free_vars sad_but_true;
> val it = [`Mortal`, `Man`] : Term.term list

- subst [Term `Mortal : 𝕀 → 𝔹` \ Term `Moron : 𝕀 → 𝔹`]
        sad_but_true;
> val it = `∀x. Man x ⊃ Moron x` : Term.term

- #conseq (dest_imp (body (rand sad_but_true)));
> val it = `Mortal x` : Term.term
- type_of it;
> val it = `:𝔹` : Type.hol_type
```


Abstract Data-Types in ML

```
structure stack :
   sig
      type α stack
      exception underflow
      val empty : α stack
      val push  : α × α stack → α stack
      val pop   : α stack → α stack
      val top   : α stack → α
   end
=
   struct
      type α stack = α list
      exception underflow
      val empty = []
      fun push (e,s) = e::s
      fun pop (e::s) = s
        | pop [] = raise underflow
      fun top (e::s) = e
        | top [] = raise underflow
   end;
```


An Abstract Data-Type of Theorems

The axioms are theorem valued constants.

```
- BOOL_CASES_AX;
> val it = ⊢ ∀t. (t = T) ∨ (t = F) : Thm.thm
- IMP_ANTISYM_AX;
> val it = ⊢ ∀t1 t2. (t1 ⊃ t2) ⊃ (t2 ⊃ t1) ⊃
                     (t1 = t2) : Thm.thm
```

The inferences rule (and definition rules) are functions that return theorems.

```
- ASSUME;
> val it = fn : Term.term → Thm.thm
- sad_but_true;
> val it = `∀x. Man x ⊃ Mortal x` : Term.term
- ASSUME sad_but_true;
> val it =
       [∀x. Man x ⊃ Mortal x] ⊢ ∀x. Man x ⊃ Mortal x :
       Thm.thm
```


A Proof in HOL

HOL proofs are ML programs that create values of the theorem type.

$$\dfrac{\dfrac{\dfrac{}{\forall x.\,Man\,x \supset Mortal\,x \vdash \forall x.\,Man\,x \supset Mortal\,x}[\text{ASSUME}]}{\forall x.\,Man\,x \supset Mortal\,x \vdash Man\,socrates \supset Mortal\,socrates}[\text{SPECIALISE}] \qquad \dfrac{}{Man\,socrates \vdash Man\,socrates}[\text{ASSUME}]}{\forall x.\,Man\,x \supset Mortal\,x, Man\,socrates \vdash Mortal\,socrates}[\text{MODUSPONENS}]$$

This proof is translated into HOL as follows:

```
- val th1 = ASSUME (Term `∀x:𝕀. Man x ⊃ Mortal x`);
> val th1 = [∀x. Man x ⊃ Mortal x] ⊢ ∀x. Man x ⊃ Mortal x :
      Thm.thm
- val th2 = ASSUME (Term `(Man (socrates:𝕀)):𝔹`);
> val th2 = [Man socrates] ⊢ Man socrates : Thm.thm
- val th3 = SPEC (Term `socrates:𝕀`) th1;
> val th3 =
      [∀x. Man x ⊃ Mortal x] ⊢ Man socrates ⊃ Mortal socrates :
      Thm.thm
- MP th3 th2;
> val it = [∀x. Man x ⊃ Mortal x, Man socrates] ⊢ Mortal socrates :
         Thm.thm
```


Goal Directed Proof in HOL

```
- g `(∀x:𝕀. Man x ⊃ Mortal x) ∧ Man socrates ⊃ Mortal socrates`;
> val it = Proof manager status: 1 proofs.
    1. Incomplete:
         Initial goal:
         (∀x. Man x ⊃ Mortal x) ∧ Man socrates ⊃ Mortal socrates
    : GoalstackPure.proofs
- e STRIP_TAC;
OK.. 1 subgoal:
> val it =
    Mortal socrates
    ------------------------------------
      0.  ∀x. Man x ⊃ Mortal x
      1.  Man socrates
     : GoalstackPure.goalstack
- e RES_TAC;
OK.. Goal proved.
 [..] ⊢ Mortal socrates
> val it =
    Initial goal proved.
    ⊢ (∀x. Man x ⊃ Mortal x) ∧ Man socrates ⊃ Mortal socrates
    : GoalstackPure.goalstack
```


Strengths and Weaknesses of HOL

- ✔ Expressive logic — good for capturing specifications and models
- ✘ Expressive logic — difficult to decide
- ✔ Typed logic — well matched to many computing problem domains
- ✘ Typed logic — a hindrance in some noncomputing domains
- ✔ Simple logic syntax — easy to mechanise
- ✔ Simple deductive system — easy to mechanise
- ✔ Secure architecture — users can extend the system safely
- ✘ User hostile — though not a fundamental limitation