Interactive Theorem Proving in HOL4

Course 02: Higher Order Logic

Dr Chun TIAN
chun.tian@anu.edu.au

Australian
National
University

8 August 2024

Acknowledgement of Country

We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present.

More information about Acknowledgement of Country can be found here and here

The LCF (Logic of Computable Functions)

- implement an abstract datatype `thm` to represent theorems
- semantics of ML ensure that values of type thm can only be created using its interface
- interface is very small
  - predefined theorems are axioms
  - function with result type theorem are inferences
- interface is carefully designed and checked
  - size of interface and implementation allow careful checking
  - one checks that the interface really implements only axioms and inferences that are valid in the used logic
- whenever you create a theorem, there is a proof for it
- proved theorems can be stored on disk, without having its proof (i.e. proof logging is optional)

(Modern) Interactive Theorem Proving

- combine strengths of manual and automated proofs
- typically the human user
  - provides insights into the problem
  - structures the proof
  - provides main arguments
- typically the computer
  - checks the proof
  - keeps track of all used assumptions
  - provides automation to grind through lengthy, but trivial proof steps
- automated provers (with or w/o proof logging) are locally used in interactive proofs

Higher Order Logic (HOL)

Higher Order Logic

- Higher Order Logic = classical higher order predicate calculus with terms from the typed λ-calculus (i.e. simple type theory)
- HOL **Logic** vs HOL **Theorem Prover** (HOL4, the software)
- HOL4 extends the Standard ML platform with more packages

Standard ML is used for

- Implementing the HOL theorem prover (kernel and utilities)
- User to writing formal proofs
- User to write custom proof tools (e.g. decision procedures)

Types in HOL

Type grammar

$$\sigma \quad ::= \quad \alpha \quad | \quad c \quad | \quad (\sigma_1, \ldots, \sigma_n)op \quad | \quad \sigma_1 \to \sigma_2$$

- **Type variables** ($\alpha, \beta, \ldots$): arbitrary (non-empty) sets in the universe $\mathcal{U}$
- **Atomic types** (c): fixed sets in the universe. Initial atomic types: `bool` and `ind`.
- **Compound types**. The type $(\sigma_1, \ldots, \sigma_n)op$ denotes the set resulting from applying the operation denoted by op to the sets denoted by $\sigma_1, \ldots, \sigma_n$.
- **Function types**. If σ_1 and σ_2 are types, then $\sigma_1 \to \sigma_2$ is the function type with domain σ_1 and codomain σ_2. It denotes the set of all (total) functions from the set denoted by its domain to the set denoted by its codomain.

Terms in HOL

Term grammar

$$t \quad ::= \quad x \quad | \quad c \quad | \quad t\,t' \quad | \quad \lambda x.\, t$$

Term grammar showing types

$$t_\sigma \quad :: \quad = x_\sigma \quad | \quad c_\sigma \quad | \quad (t_{\sigma_1 \to \sigma_2}\, t'_{\sigma_1})_{\sigma_2} \quad | \quad (\lambda x_{\sigma_1}.\, t_{\sigma_2})_{\sigma_1 \to \sigma_2}$$

- Variables (free and bound): $x, y, \ldots$
- Constants: $c, \ldots$
- Function applications: $t\,t'$, $f(x)$ or fx
- λ-Abstractions: $\lambda x.\, t$
- Terms must be well-typed in function applications.

The Theory `MIN` (Minimal)

Contents in this initial theory:

- The type constant `bool` of Booleans.
- The binary type operator `('a, 'b)fun` (or $\alpha \to \beta$) of functions.
- The type constant `ind` (rarely used directly) of individuals.
- Equality ($=$: $\alpha \to \alpha \to$ `bool`) is an infix operator.
- Implication ($\Rightarrow$: `bool` $\to$ `bool` $\to$ `bool`) is the material implication and is an infix operator that is right-associative.
- **Choice**: if `t` is a term having type $\sigma \to$ `bool`, then `@x.t(x)` denotes some member of the set whose characteristic function is `t`.

No theorems or axioms are placed in theory `min`. The primitive rules of inference of HOL depend on the presence of `min`.

Primitive Inference Rules (1)

- Assumption introduction: `ASSUME : term -> thm`

$$\frac{}{t \vdash t : \texttt{bool}}$$

- Reflexivity: `REFL : term -> thm`

$$\frac{}{\vdash t = (t : \alpha)}$$

- β-conversion: `BETA_CONV : term -> thm`

$$\frac{}{\vdash (\lambda(x : \alpha).t_1 : \beta)(t_2 : \alpha) = t_1[x \mapsto t_2] : \beta}$$

- Substitution:
  `SUBST : (thm * term) list -> term -> thm -> thm`

$$\frac{\Gamma_1 \vdash t_1 = t_1' \quad \cdots \quad \Gamma_n \vdash t_n = t_n' \qquad \Gamma \vdash t[t_1, \ldots, t_n]}{\Gamma_1 \cup \cdots \cup \Gamma_n \cup \Gamma \vdash t[t_1', \ldots, t_n']}$$

Primitive Inference Rules (2)

- Abstraction: `ABS : term -> thm -> thm`

$$\frac{\Gamma \vdash t_1 = (t_2 : \beta)}{\Gamma \vdash (\lambda(x : \alpha).t_1) = (\lambda x.t_2) : \alpha \to \beta}$$

- Type instantiation: `INST_TYPE : ...`

$$\frac{\Gamma \vdash t}{\Gamma[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n] \vdash t[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n]}$$

- Discharging an assumption: `DISCH : term -> thm -> thm`

$$\frac{\Gamma \vdash t_2 : \texttt{bool}}{\Gamma - \{t_1\} \vdash (t_1 : \texttt{bool}) \Rightarrow t_2}$$

- Modus Ponens: `MP : thm -> thm -> thm`

$$\frac{\Gamma_1 \vdash (t_1 : \texttt{bool}) \Rightarrow (t_2 : \texttt{bool}) \qquad \Gamma_2 \vdash t_1}{\Gamma_1 \cup \Gamma_2 \vdash t_2}$$

The theory LOG (Logic)

Definitions of Boolean logic constants

Name	Symbol	Definition
True	T	$\mathtt{T} = ((\lambda x.x) = (\lambda x.x : \mathtt{bool}))$
Forall	!	$\forall = \lambda(P : \alpha \to \mathtt{bool}).P = (\lambda x.\mathtt{T})$
Exists	?	$\exists = \lambda(P : \alpha \to \mathtt{bool}).P(@x.Px)$
And	/\	$\wedge = \lambda t_1\, t_2.\, \forall t.\, (t_1 \Rightarrow t_2 \Rightarrow t) \Rightarrow t$
Or	\/	$\vee = \lambda t_1\, t_2.\, \forall t.\, (t_1 \Rightarrow t) \Rightarrow (t_2 \Rightarrow t) \Rightarrow t$
False	F	$\mathtt{F} = (\forall(t : \mathtt{bool}).\, t)$
Not	~	$\neg = (\lambda t.\, t \Rightarrow \mathrm{F})$
Exists unique	?!	$?! = (\lambda P.(\exists x.Px) \wedge (\forall xy.Px \wedge Py \Rightarrow (x = y)))$

Higher Order Logic = classical higher order predicate calculus with terms from the typed λ-calculus (i.e. simple type theory)

The theory INIT (Initial)

The first three axioms:

- BOOL_CASES_AX: $\vdash \forall t.(t = \mathtt{T}) \vee (t = \mathtt{F})$
- ETA_AX: $\vdash \forall t.(\lambda x.tx) = t$
- SELECT_AX: $\vdash \forall Px.Px \Rightarrow P(\$@P)$ (or $P(@x.Px)$)

The infinite axiom (and needed definitions):

- ONE_ONE $= (\lambda f.\forall x_1 x_2.f(x_1) = f(x_2) \Rightarrow x_1 = x_2)$
- ONTO $= (\lambda f.\forall y.\exists x.y = f(x))$
- INFINITY_AX: $\vdash \exists f.\mathtt{ONE\_ONE}\, f \wedge \neg \mathtt{ONTO}\, f$

$f(x) = 2x$ on the set of all natural numbers is both one-one and onto.

The above four axioms form HOL's Standard Theory.

The HOL Logic in Standard ML

Types

API for Types

```
mk_vartype : string -> hol_type
mk_thy_type : {Tyop:string, Thy:string, Args:hol_type list} -> hol_type
dest_vartype : hol_type -> string
dest_thy_type : hol_type -> {Tyop:string, Thy:string, Args:hol_type list}
```

Examples

```
> mk_vartype "'a";
val it = ``:'a``: hol_type
> mk_thy_type{Tyop="fun", Thy="min",
              Args=[mk_vartype "'a",
                    mk_thy_type{Tyop="bool", Thy="min", Args=[]}]};
val it = ``:'a -> bool``: hol_type
> ``:'a`` --> ``:bool``;
val it = ``:'a -> bool``: hol_type
```


Terms

API for Terms

```
mk_var          : (string * hol_type) -> term
mk_thy_const    : {Name:string, Thy:string, Ty:hol_type} -> term
mk_comb         : (term * term) -> term
mk_abs          : (term * term) -> term

dest_var        : term -> (string * hol_type)
dest_thy_const  : term -> {Name:string, Thy:string, Ty:hol_type}
dest_comb       : term -> (term * term)
dest_abs        : term -> (term * term)

type_of         : term -> hol_type
aconv           : term -> term -> bool
```

```
> aconv ``\x. x`` ``\y. y``;
val it = true: bool
```

Theorems

There's no API to make (valid) theorems directly.

API for Theorems

```
dest_thm : thm -> (term list * term)
hyp      : thm -> term list
concl    : thm -> term
```

```
> BOOL_CASES_AX;
val it = |- !(t :bool). (t <=> T) \/ (t <=> F): thm
> dest_thm it;
val it = ([], ``!(t :bool). (t <=> T) \/ (t <=> F)``): term list * term
> hyp BOOL_CASES_AX;
val it = []: term list
> concl BOOL_CASES_AX;
val it = ``!(t :bool). (t <=> T) \/ (t <=> F)``: term
```

Theories

API for Theories

```
current_theory : unit -> string
ancestry       : string -> string list
new_theory     : string -> unit
load           : string -> unit
new_type       : int -> string -> unit
new_constant   : (string * type) -> unit
new_axiom      : (string * term) -> thm
save_thm       : (string * thm) -> thm
store_thm      : (string * term * tactic) -> thm
export_theory  : unit -> unit
```

The `Holmake` command compiles `xxxScript.sml` to `xxxTheory.sml|sig`, where `xxx` is the theory name.

Appendix: The Axioms of Set Theory (1)

Axioms 1–6

- **Extensionality**. $\forall z(z \in x \leftrightarrow z \in y) \rightarrow x = y$
- **Foundation**. $\exists y(y \in x) \rightarrow \exists y(y \in x \wedge \neg\exists z(z \in x \wedge z \in y))$
- **Comprehension Scheme**. For each formula φ without y free,

$$\exists y \forall x(x \in y \leftrightarrow x \in v \wedge \varphi(x))$$

- **Pairing**. $\exists z(x \in z \wedge y \in z)$
- **Union**. $\exists A \forall Y \forall x(x \in Y \wedge Y \in \mathcal{F} \rightarrow x \in A)$
- **Replacement Scheme**. For each formula φ without B free,

$$\forall x \in A \exists! y\, \varphi(x, y) \rightarrow \exists B \forall x \in A \exists y \in B\, \varphi(x, y)$$

Appendix: The Axioms of Set Theory (2)

Definitions

$$x \subseteq y \iff \forall z(z \in x \to z \in y)$$
$$x = \emptyset \iff \forall z(z \notin x)$$
$$y = S(x) \iff \forall z(z \in y \leftrightarrow z \in x \vee z = x)$$
$$y = v \cap w \iff \forall x(x \in y \leftrightarrow x \in v \wedge x \in w)$$
$$\mathrm{SING}(x) \iff \exists y \in x \forall z \in x(z = y)$$

Axioms 7–9

- **Infinity**. $\exists x(\emptyset \in x \wedge \forall y \in x(S(y) \in x))$
- **Power Set**. $\exists y \forall z(z \subseteq x \to z \in y)$
- **Choice, or AC**

$$\emptyset \notin \mathcal{F} \wedge \forall x \in \mathcal{F} \forall y \in \mathcal{F}(x \neq y \to x \cap y = \emptyset) \to \exists \mathcal{C} \forall x \in \mathcal{F}\,(\mathrm{SING}(\mathcal{C} \cap x))$$