HOL Theorem Proving and Formal Probability (1)

Chun TIAN
chun.tian@anu.edu.au

13/03/2024

Aims of this Course

- Introduction to interactive theorem proving (ITP)
- Hands-on experience with HOL4
- Core theories of HOL4 (num, list, ...)
- Core Math theories of HOL4 (pred_set, real, ...)
- Formal probability theory (with measure theory and Lebesgue integration)
- Learn how to prove simple probability theroems
- ...

(NOTE: some slides used materials from the slides of Thomas Tuerk)

Formal Verification and Theorem Proving

- **Formal verification** (aka formal methods) is the act of proving or disproving the correctness of intended algorithms underlying a system w.r.t. certain formal specifications, using mathematical methods.
- Traditional formal verification techniques: **Model Checking**, **Testing** and **Theorem Proving**.
- Model Checking: fully automated (pros); state-explosion problem (cons, cf. SPIN vs. SMV)
- Testing: good for disproving; incapable for proving.
- Theorem Proving: good for proving; bad for disproving; expensive logics (non-decidable); tedious proofs (time consuming).

Mathematical (Informal) Proofs vs Formal Proofs

Mathematical (Informal, Pencil-and-paper) Proofs:

- informal, convinces other mathematicians
- checked by community of domain experts (“elders”)
- subtle errors are hard to find
- often short, but may require creativity and brilliant ideas; with gaps

Formal Proofs:

- encoded in a logical formalism
- checkable by stupid machines
- trustworthy
- often long and tedious

Automated vs. Manual Proofs

Automated Proofs (e.g. ACL2):

- amazing successes in certain domains (e.g., SAT solving)
- still, often infeasible for interesting problems
- hard to get insights in case a proof attempt fails
- even if it works, it is often not that automated (heuristics still needed)

Manual Proofs: (e.g. HOL88)

- very tedious; one has to grind through many trivial but detailed proofs
- easy to make mistakes (more code more mistakes);
- hard to maintain (if anything changed)

(Modern) Interactive Proofs

- 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

The LCF (Logic of Computable Functions) Approach

- 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)

Higher Order Logic and HOL4

- 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)

HOL Types

Type grammar:

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

- **Type variables** (α, β, ...): 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.

HOL Terms

Term grammar:

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

Term grammar (showing types):

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

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

The Theory `MIN` (part of `bool`)

What's contained 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 Rules of Inference of the HOL Logic

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

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

2. Reflexivity: `REFL : term -> thm`

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

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

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

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

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

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

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

6. Type instantiation: `INST_TYPE : ...`

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

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

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

8. 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 (part of bool)

The logical constants:

- **True**: $\mathtt{T} = ((\lambda(x:\mathtt{bool}).x) = (\lambda x.x))$
- **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**: $\mathtt{F} = (\forall(t:\mathtt{bool}).\, t)$
- **Not**: $\neg = (\lambda t.\, t \Rightarrow \mathtt{F})$
- **Exists unique**: $?! = (\lambda P.(\exists x.Px) \wedge (\forall xy.Px \wedge Py \Rightarrow (x = y)))$

The theory $\mathtt{INIT}$ (part of $\mathtt{bool}$)

The first three axioms (4 in total) in HOL's **Standard Theory**:

- $\mathtt{BOOL\_CASES\_AX}$: $\vdash \forall t.(t = \mathtt{T}) \vee (t = \mathtt{F})$
- $\mathtt{ETA\_AX}$: $\vdash \forall t.(\lambda x.tx) = t$
- $\mathtt{SELECT\_AX}$: $\vdash \forall Px.Px \Rightarrow P(\$@P)$ (or $P(@x.Px)$)

The infinite axiom (and needed definitions):

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

(Example: $f(x) = 2x$ on the set of all natural numbers.)