Higher-order logic

Bohua Zhan

Institute of Software, Chinese Academy of Sciences

December 2022

Reading

- Specification and Verification I (Mike Gordon), Chapter 6
- Concrete Semantics (Nipkow & Klein), Chapter 2

Higher-order logic

A generalization of first-order logic. Major features:

- Ability to quantify over predicates / functions. This allows (for example) directly stating induction rules:

$$\forall P.\ P(0) \longrightarrow (\forall n.\ P(n) \longrightarrow P(n+1)) \longrightarrow (\forall n.\ P(n))$$

- Lambda terms for representing predicates / functions. E.g.:

  Predicates on natural numbers:

$$\lambda x.\ x > 2, \quad \lambda x.\ \mathrm{prime}(x) \wedge \mathrm{prime}(x+2)$$

  Functions on natural numbers:

$$\lambda x.\ x + 2, \quad \lambda x.\ \lambda y.\ x^2 + y^2$$

Types

Each term in higher-order logic has a type.

- Fundamental types: nat, bool, ...
- Function type $\alpha \Rightarrow \beta$.
- Types depending on other types: α list, α set, $\alpha \times \beta$, ...
- Construct new type by induction (examples later).
- Construct new type by subtyping.

Terms

Terms are constructed from:

- Variables: x, y, v, w, a, b, ...
- Constants: zero, Suc, plus, times, nil, cons, append, ...
- Function application, written as $f\ x$ instead of $f(x)$.
- Lambda terms $\lambda x.\ e$ (where e is a term possibly containing x).

Typing rules (informally):

- Each variable and constant is associated a type.
- For function application $f\ x$, f and x must have types $\alpha \Rightarrow \beta$ and α for some α and β. Then $f\ x$ has type β.
- For lambda term $\lambda x.\ e$, if x has type α and e has type β, then $\lambda x.\ e$ has type $\alpha \Rightarrow \beta$.

Currying

- Function taking two arguments of type α and β, and returns a value of type γ, can be represented as a term of type

$$\alpha \Rightarrow (\beta \Rightarrow \gamma)$$

  or more simply $\alpha \Rightarrow \beta \Rightarrow \gamma$ (operation $\Rightarrow$ associates to the right).
- Application of such a function f to values a (of type α) and b (of type β) is written as $(f\ a)\ b$, or more simply $f\ a\ b$ (function application associates to the left).

Make sure you understand why this makes sense!

Notation

- plus and times are functions of type $nat \Rightarrow nat \Rightarrow nat$.
- We use infix notations:

$$a + b \text{ for plus } a\ b \quad \text{and} \quad a \times b \text{ for times } a\ b.$$

- Forall and Exists are functions of type

$$(\alpha \Rightarrow bool) \Rightarrow bool$$

- We use binder notations:

$$\forall x.\, P\,x \text{ for Forall}\,(\lambda x.\, P\,x) \quad \text{and} \quad \exists x.\, P\,x \text{ for Exists}\,(\lambda x.\, P\,x).$$

Deduction rules

Mostly generalizes that of first-order logic. Some new rules:

- Beta-conversion:
$$\frac{}{\vdash (\lambda x.\, t)\, s = t[s/x]}$$
.
- Abstraction:
$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash (\lambda x.\, t_1) = (\lambda x.\, t_2)}$$
where x does not occur in Γ.
- Forall introduction and elimination:
$$\frac{\Gamma \vdash t}{\Gamma \vdash \forall x.\, t} \quad \text{and} \quad \frac{\vdash \forall x.\, t}{\vdash t[s/x]}$$

Isabelle

- Interactive Theorem Prover: user proves theorems by interacting with the computer. The computer checks the proofs according to the rules of higher-order logic.
- Free (and open source) at

https://isabelle.in.tum.de/

Demo!