Higher-Order Logic

Specification and Verification with Higher-Order Logic

Arnd Poetzsch-Heffter
(Slides by Jens Brandt)

Software Technology Group
Fachbereich Informatik
Technische Universität Kaiserslautern

Sommersemester 2008

Outline

Overview

Higher-Order Logic

- quantification over predicates, functions and sets
- supports formalisation of arbitrary mathematics

Motivation

- reasoning about hardware and software can require very sophisticated mathematics
- floating point: real numbers and analysis
- correctness of randomised algorithms: probability

Outline

Problem: Russell's Paradox

Russel's Paradox

Having variables that range over predicates allows to write terms like

$$\Omega \overset{def}{=} \lambda P. \neg(P\ P)$$

where P is a variable. By β-reduction:

$$\Omega\,\Omega = (\lambda P. \neg(P\ P))\,\Omega = \neg(\Omega\,\Omega)$$

Conclusion

To avoid this kind of thing types are needed!

Types

Syntax of Types

- type constant: c
- type variable: α
- compound type: $(\sigma_1, \ldots, \sigma_n)op$

Type Examples

Example (Type Constant)

- *bool*: Booleans
- *num*: natural numbers
- *weekday*: some appropriate user defined type

Example (Compound Types)

- (σ_1, σ_2)*fun*: functions from σ_1 to σ_2
- (σ_1, σ_2)*prod*: pairs of values

Terminology and Notation

Definition (Type operator)

- '*op*' in $(\sigma_1, \ldots, \sigma_n)op$ is called a type constructor

Conventions

- The type $(\sigma_1, \sigma_2)fun$ is usually written $\sigma_1 \longrightarrow \sigma_2$ and $\sigma_1 \longrightarrow \sigma_2 \longrightarrow \cdots \longrightarrow \sigma_n = (\sigma_1 \longrightarrow (\sigma_2 \longrightarrow (\cdots \longrightarrow \sigma_n)))$
- The type $(\sigma_1, \sigma_2)prod$ is usually written $\sigma_1 \times \sigma_2$ or $\sigma_1 * \sigma_2$ and $\sigma_1 * \sigma_2 * \cdots * \sigma_n = (\sigma_1 * (\sigma_2 * (\cdots * \sigma_n)))$

Typing of Terms

Typing of Terms

- All terms must be well-typed.
- $t{:}\sigma$ means the term t is well-typed and has type σ.

Variables and Constants

- Variables may have any type: $v{:}\sigma$
- Constants have a fixed generic type: $c{:}\sigma$

Assigning Types to Terms

Rules for the Assignment

- function application

$$\frac{t_1 : \sigma_1 \longrightarrow \sigma_2 \qquad t_2 : \sigma_1}{(t_1\ t_2) : \sigma_2}$$

- abstraction

$$\frac{x : \sigma_1 \qquad t : \sigma_2}{\lambda x.\, t : \sigma_1 \longrightarrow \sigma_2}$$

Polymorphism

Example (Polymorphism)

Consider the constant I, defined by:

$$I \stackrel{def}{=} \lambda x.\, x$$

We may want to apply the function I to things of different types:

- $I\, 7 = 7$ with $I : num \rightarrow num$
- $I\, T = T$ with $I : bool \rightarrow bool$

It seems that I must have two different types.

Polymorphism and Generic Types

Polymorphism

The types of polymorphic functions such as I contain type variables:

$$I \stackrel{def}{=} (\lambda x.\, x){:}\alpha{\rightarrow}\alpha$$

where α stands for ‘any type’. $\alpha{\rightarrow}\alpha$ is the *generic* type of I.
The constant I then has every type obtainable by substituting any type for the variable α in its generic type:

- $I : bool \longrightarrow bool$
- $I : num \longrightarrow num$
- $I : (\alpha \longrightarrow bool) \longrightarrow (\alpha \longrightarrow bool)$
- $I : \alpha \longrightarrow \alpha$

Polymorphism Examples

Example (Function Composition)

$$o \overset{def}{=} \lambda f.\lambda g.\lambda x.f(g(x))$$

where $o : (\beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma)$

Example (Equality)

$$= : \ \alpha \to \alpha \to bool$$

Example (Apply a Function and Add)

$$app\_add \overset{def}{=} \lambda f.(\lambda x.f(x) + f(x))$$

where $app\_add : (\alpha \to num) \to (\alpha \to num)$

Church's Simple Theory of Types

Definition (Universe)

- each element $X \in \mathscr{U}$ is a non-empty set
- if $X \in \mathscr{U}$ and $Y \subseteq X$, then $Y \in \mathscr{U}$.
- if $X \in \mathscr{U}$ and $Y \in \mathscr{U}$, then $X \times Y \in \mathscr{U}$
- if $X \in \mathscr{U}$, then powerset $\wp(X) = \{Y : Y \subseteq X\} \in \mathscr{U}$
- $\mathscr{U}$ contains a distinguished infinite set I
- distinguished element $ch \in \Pi_{X \in \mathscr{U}} X$:
  $ch(X) \in X$ witnesses non-emptiness

Model

Definition (Model of Type Structure)

- given: type structure Ω as set of type constants (ν, n)
- model: $M(\nu) : \mathscr{U}^n \to \mathscr{U}$

Polymorphic Types

- types containing type variables: polymorphic
- meaning of polymorphic types not single set, but set-valued function

Summary of Types

Fact (Types)

Types are introduced to avoid inconsistency.

Types

- Type constants: *bool*, *num*, ...
- Type variables: α, β, γ, ...
- Compound Types: $(\sigma_1, \ldots, \sigma_n)op$ e.g. $\sigma_1 \to \sigma_2$, and $\sigma_1 \times \sigma_2$.

Polymorphism

- $twice \stackrel{def}{=} \lambda f.\lambda x.f(f(x))$
  where $twice : (\alpha \to \alpha) \to (\alpha \to \alpha)$

Outline

Syntax of Terms

Syntax of Terms

- constants: c
- variables : v
- function applications: $T_1 \ \ T_2$
- lambda abstractions $\lambda v.\, T$

Constants and Variables

Fact (Distinction between Constants and Variables)

The distinction between a constant and a variable always depends on the context.

Identifiers

x, y, foo, t', k_2, c_val, $\ldots$

Special Symbols

$\exists, \forall, \supset, \wedge, \vee, \neg, 1, 2, 3, \ldots, +, \times, =, \ldots$

Function Applications

Notation

$$\langle term_1 \rangle \ \langle term_2 \rangle$$

- denotes the result of applying the function $\langle term_1 \rangle$ to the value $\langle term_2 \rangle$.

Precedence

- parentheses can be used for grouping

$$f(x),\ f\ (g\ y),\ (f\ x)\ y, \ldots$$

- default precedence

$$f\ x_1\ x_2\ \cdots\ x_n = (((f\ x_1)\ x_2)\ \cdots\ x_n)$$

Abstractions

Notation

$$\lambda \langle var \rangle . \langle term \rangle$$

- denotes the function $x \mapsto term[x/var]$.

Convention

- $\lambda x_1\ x_2\ \cdots\ x_n\,.\,t\ =\ \lambda x_1\,.\,\lambda x_2\,.\ \cdots\ \lambda x_n\,.\,t$

Example (Abstraction)

- $\lambda x.\ x$: the identity function
- $\lambda x.\ f(f\ x)$: function that applies f twice
- $\lambda f.\lambda g.\lambda x.\ f(g\ x)$: function composition

Free and Bound Variables

Definition (Free Variable)

$$\lambda x.\langle body \rangle$$

- A variable x is called free in a term if it does not occur inside the body of an abstraction.

Definition (Bound Variables)

- If an instance of a variable is not free, it is bound.

Example (Free and Bound Variables)

- Consider variable x:

$$(\lambda x.\, f\ x)(\lambda y.\, x)$$

Syntactic Sugar

Infix Aplications

Certain constants are written in infix position:

- $t_1 + t_2$ abbreviates $+\ t_1\ t_2$
- $t_1 \times t_2$ abbreviates $\times\ t_1\ t_2$
- $t_1 \wedge t_2$ abbreviates $\wedge\ t_1\ t_2$

Summary of Terms

Terms

Terms may be

- Variables: x, y, a', a_var, $phi_1, \ldots$
- Constants: T, F, phi, $\exists$, $+, \ldots$
- Applications: $t_1\ t_2$, $t_1\ t_2\ t_3\ \ldots\ t_n$
- Abstractions: $\lambda x.\, t$, $\lambda\ x_1\ x_2\ \ldots\ x_n.\, t$

Higher-Order Terms

Fact (Higher-Order Terms)

- *Variables can range over functions or predicates (i. e. boolean-valued functions)*

Example (Higher-Order Term)

- in $\lambda f. f\ 0$, the variable f ranges over functions
- in $\forall P.\ P(n) \longrightarrow P(n{+}1)$, P ranges over predicates
- typical assertion

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

Syntactic Sugar

Binders

The quantifiers $\forall$ and $\exists$ are in fact polymorphic constants with types:

- $\forall : (\alpha \to \mathbb{B}) \to \mathbb{B}$
- $\exists : (\alpha \to \mathbb{B}) \to \mathbb{B}$

They are defined such that for $P : (\alpha \to bool)$:

- $\forall P$ means $P(x) = T$ for all x
- $\exists P$ means $P(x) = T$ for some x

Hilbert's Choice Function

Definition (ε-Operator)

$$\varepsilon x.\, t[x]$$

- with $x : \sigma$ and $t[x]$ a term involving x
- binder of type $(\sigma \to \mathbb{B}) \to \sigma$
- denotes a value of type σ
  - some value of type σ, $v{:}\sigma$ such that $t[v]$ is true
  - no such value exists: arbitrary but fixed value of type σ

Examples of ε-Terms

Example (ε-Terms)

- This term denotes the number 1: $\varepsilon x.\, 0 < x \wedge x < 2$
- This term denotes an even number: $\varepsilon x.\, \exists y.\, x = 2 \cdot y$
- An unspecified natural number: $\varepsilon x.\, x + 1 = x$
- The following proposition is true: $(\varepsilon x.\, x + 3 = 9) = 6$

Standard Signatures

Standard Signature and Intended Interpretation

- standard type structure Ω contains the atomic types $\mathbb{B}$ of Boolean values and I of individuals
- $\rightarrow$ of type $(\mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B})$
  Intended interpretation: implication
- $=$ of type $(\alpha \rightarrow \alpha \rightarrow \mathbb{B})$
  Intended interpretation: equality on the set α
- ε of type $((\alpha \rightarrow \mathbb{B}) \rightarrow \alpha)$
  Intended interpretation: Hilbert's choice function.

Standard Logical Constants

Definition of Standard Logical Constants

EXISTS $\vdash_{\text{def}} \exists = \lambda P.P(\varepsilon P)$
TRUTH $\vdash_{\text{def}} \text{true} = ((\lambda x.x) = (\lambda x.x))$
FORALL $\vdash_{\text{def}} \forall = \lambda P.(P = (\lambda x.\text{true}))$
FALSITY $\vdash_{\text{def}} \text{false} = \forall x.x$
NEGATION $\vdash_{\text{def}} \neg = \lambda x.x \longrightarrow \text{false}$
DISJUNCTION $\vdash_{\text{def}} \vee = \lambda(x, y).\neg x \longrightarrow y$
CONJUNCTION $\vdash_{\text{def}} \wedge = \lambda(x, y).\neg(\neg x \vee \neg y)$

Outline

Formulas

Definition (Formulas in HOL)

- Formulas in HOL are terms of type $\mathbb{B}$

Example (Formulas in HOL)

- $\forall x.\, x = 0 \vee \neg(x = 0)$
- true
- $(\lambda x.\, \neg x)(\forall y.\, y = y)$
- $\forall x.\, x = \text{true}$

Sequents

Definition (Sequents in HOL)

A sequent is a pair (Γ, t) where

- Γ is a set of formulas (assumptions)
- t is a formula (conclusion)

A sequent (Γ, t) essentially means

- From the formulas in Γ, t can be derived.

Example (Sequents in HOL)

The sequent $(\{x = 3, \forall n.\, n = n\}, x = 99)$ means

$$\{\, x = 3,\ y = 7,\ \forall n.\, n = n \,\} \vdash x + y = 10$$

Theorems

Definition (Theorems in HOL)

A theorem is a sequent that is either

- an axiom, or
- can be derived from other theorems

Notation

- $\Gamma \vdash t$ or just $\vdash t$ if Γ is empty

Example (HOL Theorems)

- $\vdash \forall x.\, x = 0 \vee \neg(x = 0)$?
- $\vdash$ true ?
- $\vdash (\lambda x.\, \neg x)(\forall y.\, y = y)$?
- $\vdash \forall x.\, x = \text{true}$?

Axioms of the HOL Logic

Five Axioms

- $\vdash \forall\, b.\ (b = true) \vee (b = false)$
- $\vdash \forall\, b_1\ b_2.\ (b_1 \longrightarrow b_2) \longrightarrow (b_2 \longrightarrow b_1) \longrightarrow (b_1 = b_2)$
- $\vdash \forall\, f.\ (\lambda x.\ fx) = f$
- $\vdash \forall\, P\ x.\ P\ x \longrightarrow P(\varepsilon\ P)$
- $\vdash \exists f.(\forall x\ y.\ fx = fy \longrightarrow x = y) \wedge (\neg \forall x.\ \exists y.\ x = f\ y)$

Inference Rules

Primitive Inference Rules

- **ASSUME** $\dfrac{}{\{t\} \vdash t}$
- **REFL** $\dfrac{}{\vdash t = t}$
- **MP** $\dfrac{\Gamma_1 \vdash t_1 \longrightarrow t_2 \qquad \Gamma_2 \vdash t_1}{\Gamma_1 \cup \Gamma_2 \vdash t_2}$
- **DISCH** $\dfrac{\Gamma \vdash t_2}{\Gamma - \{t_1\} \vdash t_1 \longrightarrow t_2}$
- **ABS** $\dfrac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash (\lambda x.\ t_1) = (\lambda x.\ t_2)}$ (with x not free in Γ)

Inference Rules

Primitive Inference Rules (continued)

- **BETA_CONV** $\dfrac{}{\vdash (\lambda x.\ t_1)t_2 = t_1[t_2/x]}$
- **SUBST** $\dfrac{\Gamma_1 \vdash t_1 = t_2 \qquad \Gamma_2 \vdash t[t_1]}{\Gamma_1 \cup \Gamma_2 \vdash t[t_2]}$
- **INST_TYPE** $\dfrac{\Gamma \vdash t}{\Gamma \vdash t[\sigma_1 \ldots \sigma_n/\alpha_1 \ldots \alpha_n]}$

Beta Conversion

Rule for Beta-Conversion

$$\textbf{BETA\_CONV} \quad \frac{}{\vdash (\lambda x.\, t_1)t_2 = t_1[t_2/x]}$$

- $t_1[t_2/x]$ denotes the result of substituting t_2 for all free occurrences of x in t_1
- bound variables renamed if necessary so that no free variable in t_2 becomes bound

Example (Beta Conversion)

- $\vdash (\lambda x.\, x+3)\, 7 = 7+3$
- $\vdash (\lambda x.\, (\forall x.\, x = \mathrm{true}) \rightarrow x)\, \mathit{false} = (\forall x.\, x = \mathrm{true}) \rightarrow \mathrm{false})$
- $\vdash (\lambda y.\, \forall x.\, x = y)\, x = (\forall x'.\, x' = x)$

Substitution

Rule for Substitution

$$\textbf{SUBST} \quad \frac{\Gamma_1 \vdash t_1 = t_2 \qquad \Gamma_2 \vdash t[t_1]}{\Gamma_1 \cup \Gamma_2 \vdash t[t_2]}$$

- where $t[t_1]$ is a term with selected free occurences of t_1 ‘singled out’ for
- $t[t_2]$ is the result of replacing those chosen t_1 by t_2
- bound variables are renamed so that variables free in t_2
- do not become bound in $t[t_2]$

Type Instantiation

Rule for Type Instantiation

$$\textbf{INST\_TYPE} \quad \frac{\Gamma \vdash t}{\Gamma \vdash t[\sigma_1 \ldots \sigma_n / \alpha_1 \ldots \alpha_n]}$$

which effects the parallel substitution of types $\sigma_1 \ldots \sigma_n$ for type variables $\alpha_1 \ldots \alpha_n$ in t.
Restriction: none of $\alpha_1 \ldots \alpha_n$ occur in Γ.

Example (Type Instantiation)

$$\frac{\vdash I(x : \alpha) = x}{\vdash I(x : num) = x}$$

Outline

Summary

Higher-Order Logic

- types and terms
- quantification over predicates, functions and sets

HOL Proof System

- five axioms and eight primitive inference rules