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

1 Introduction

2 Types
   - Motivation
   - Syntax
   - Polymorphism
   - Semantics

3 Terms
   - Syntax
   - Higher-Order Terms
   - Semantics

4 HOL Proof System
   - Formulas and Sequents
   - Axioms and Rules

5 Summary
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

1 Introduction

2 Types
   - Motivation
   - Syntax
   - Polymorphism
   - Semantics

3 Terms
   - Syntax
   - Higher-Order Terms
   - Semantics

4 HOL Proof System
   - Formulas and Sequents
   - Axioms and Rules

5 Summary
Problem: Russell’s Paradox

Russel’s Paradox

Having variables that range over predicates allows to write terms like

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

where $P$ is a variable. By $\beta$-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: $\alpha$
- compound type: $(\sigma_1, \ldots, \sigma_n) o p$
Type Examples

Example (Type Constant)
- \textit{bool}: Booleans
- \textit{num}: natural numbers
- \textit{weekday}: some appropriate user defined type

Example (Compound Types)
- \((\sigma_1, \sigma_2)\) \textit{fun}: functions from \(\sigma_1\) to \(\sigma_2\)
- \((\sigma_1, \sigma_2)\) \textit{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 \((\sigma_1, \sigma_2)\text{fun}\) is usually written \(\sigma_1 \to \sigma_2\) and 
  \(\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n = (\sigma_1 \to (\sigma_2 \to (\cdots \to \sigma_n)))\)

- The type \((\sigma_1, \sigma_2)\text{prod}\) is usually written \(\sigma_1 \times \sigma_2\) or \(\sigma_1 \ast \sigma_2\)
  and \(\sigma_1 \ast \sigma_2 \ast \cdots \ast \sigma_n = (\sigma_1 \ast (\sigma_2 \ast (\cdots \ast \sigma_n)))\)
Typing of Terms

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

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
  
  \[
  \frac{t_1 : \sigma_1 \rightarrow \sigma_2 \quad t_2 : \sigma_1}{(t_1 \ t_2) : \sigma_2}
  \]

- abstraction
  
  \[
  \frac{x : \sigma_1 \quad t : \sigma_2}{\lambda x. t : \sigma_1 \rightarrow \sigma_2}
  \]
Polymorphism

Example (Polymorphism)

Consider the constant $I$, defined by:

$$ I \overset{\text{def}}{=} \lambda x. x $$

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

- $I \, 7 = 7$ with $I : \text{num} \to \text{num}$
- $I \, T = T$ with $I : \text{bool} \to \text{bool}$

It seems that $I$ must have two different types.
Polymorphism

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

\[
I \overset{\text{def}}{=} (\lambda \, x. \, x) : \alpha \to \alpha
\]

where \( \alpha \) stands for ‘any type’. \( \alpha \to \alpha \) is the *generic* type of \( I \).

The constant \( I \) then has every type obtainable by substituting any type for the variable \( \alpha \) in its generic type:

- \( I : \text{bool} \to \text{bool} \)
- \( I : \text{num} \to \text{num} \)
- \( I : (\alpha \to \text{bool}) \to (\alpha \to \text{bool}) \)
- \( I : \alpha \to \alpha \)
Polymorphism Examples

Example (Function Composition)

\[ o \overset{\text{def}}{=} \lambda f.\lambda g.\lambda x.f(g(x)) \]

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

Example (Equality)

\[ = : \alpha \to \alpha \to \text{bool} \]

Example (Apply a Function and Add)

\[ \text{app}\_\text{add} \overset{\text{def}}{=} \lambda f.(\lambda x.f(x) + f(x)) \]

where \( \text{app}\_\text{add} : (\alpha \to \text{num}) \to (\alpha \to \text{num}) \)
Church’s Simple Theory of Types

Definition (Universe)

- Each element $X \in \mathcal{U}$ is a non-empty set
- If $X \in \mathcal{U}$ and $Y \subseteq X$, then $Y \in \mathcal{U}$.
- If $X \in \mathcal{U}$ and $Y \in \mathcal{U}$, then $X \times Y \in \mathcal{U}$
- If $X \in \mathcal{U}$, then powerset $\mathcal{P}(X) = \{ Y : Y \subseteq X \} \in \mathcal{U}$
- $\mathcal{U}$ contains a distinguished infinite set $I$
- Distinguished element $ch \in \prod_{X \in \mathcal{U}} X: \quad ch(X) \in X$ witnesses non-emptiness
Definition (Model of Type Structure)

- given: type structure $\Omega$ as set of type constants $(\nu, n)$
- model: $M(\nu) : \mathcal{U}^n \rightarrow \mathcal{U}$

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: `(σ₁, …, σₙ)op` e.g. `σ₁ → σ₂`, and `σ₁ × σ₂`.

Polymorphism

- `twice ≡ \lambda f.\lambda x.f(f(x))` where `twice : (α → α) → (α → α)`. 
Outline

1. Introduction
2. Types
   - Motivation
   - Syntax
   - Polymorphism
   - Semantics
3. Terms
   - Syntax
   - Higher-Order Terms
   - Semantics
4. HOL Proof System
   - Formulas and Sequents
   - Axioms and Rules
5. Summary
Syntax of Terms

- constants: $c$
- variables: $v$
- function applications: $T_1 \ T_2$
- lambda abstractions $\lambda v. T$
The distinction between a constant and a variable always depends on the context.

Identifiers

\( x, y, \ foo, \ t', \ k_2, \ c_{\text{val}}, \ldots \)

Special Symbols

\( \exists, \ \forall, \ \exists, \ \wedge, \ \vee, \ \neg, \ 1, \ 2, \ 3, \ldots, +, \times, =, \ldots \)
Function Applications

Notation

\[ \langle \text{term}_1 \rangle \langle \text{term}_2 \rangle \]

denotes the result of applying the function \( \langle \text{term}_1 \rangle \) to the value \( \langle \text{term}_2 \rangle \).

Precedence

- parentheses can be used for grouping
  
  \[ f(x), f(gy), (fx)y, \ldots \]

- default precedence
  
  \[ f \ x_1 \ x_2 \ \cdots \ x_n = (((f \ x_1) \ x_2) \ \cdots \ x_n) \]
Abstractions

**Notation**

\[ \lambda \langle \text{var} \rangle . \langle \text{term} \rangle \]

- denotes the function \( x \mapsto \text{term}[x / \text{var}] \).

**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)

\[ \lambda x.\langle \text{body} \rangle \]

- 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 Applications

Certain constants are written in infix position:

- $t_1 + t_2$ abbreviates $+ t_1 t_2$
- $t_1 \times t_2$ abbreviates $\times t_1 t_2$
- $t_1 \land t_2$ abbreviates $\land t_1 t_2$
Summary of Terms

Terms

Terms may be

- Variables: $x, y, a', a_{\text{var}}, \phi_1, \ldots$
- 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 $\forall P. P(n) \rightarrow P(n+1)$, $P$ ranges over predicates
- Typical assertion

$$\forall x \; f. \, \exists g. (g \, 0 = x) \land \forall n. g \,(n+1) = (f \,(g \, n))$$
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 $P : (\alpha \to \text{bool})$:

- $\forall P$ means $P(x) = T$ for all $x$
- $\exists P$ means $P(x) = T$ for some $x$
Hilbert’s Choice Function

Definition ($\varepsilon$-Operator)

$\varepsilon x. t[x]$

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

- This term denotes the number 1: $\varepsilon x. 0 < x \land x < 2$
- 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 $\Omega$ 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 $\alpha$
- $\varepsilon$ 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** \( \vdash_{\text{def}} \neg = \lambda x.x \rightarrow \text{false} \)

**DISJUNCTION** \( \vdash_{\text{def}} \lor = \lambda (x, y). \neg x \rightarrow y \)

**CONJUNCTION** \( \vdash_{\text{def}} \land = \lambda (x, y). \neg (\neg x \lor \neg y) \)
Formulas

Definition (Formulas in HOL)

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

Example (Formulas in HOL)

- $\forall x. x = 0 \lor \neg (x = 0)$
- true
- $(\lambda x. \neg x)(\forall y. y = y)$
- $\forall x. x = true$
Sequents

Definition (Sequents in HOL)

A sequent is a pair \((\Gamma, t)\) where
- \(\Gamma\) is a set of formulas (assumptions)
- \(t\) is a formula (conclusion)

A sequent \((\Gamma, t)\) essentially means
- From the formulas in \(\Gamma\), \(t\) 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 $\Gamma$ is empty

Example (HOL Theorems)

- $\vdash \forall x. x = 0 \lor \neg(x = 0)$ ?
- $\vdash true$ ?
- $\vdash (\lambda x. \neg x)(\forall y. y = y)$ ?
- $\vdash \forall x. x = true$ ?
Axioms of the HOL Logic

Five Axioms

1. \( \forall b. (b = true) \lor (b = false) \)
2. \( \forall b_1 b_2. (b_1 \rightarrow b_2) \rightarrow (b_2 \rightarrow b_1) \rightarrow (b_1 = b_2) \)
3. \( \forall f. (\lambda x. fx) = f \)
4. \( \forall P x. P x \rightarrow P(\varepsilon P) \)
5. \( \exists f. (\forall x y. fx = fy \rightarrow x = y) \land (\neg \forall x. \exists y. x = f y) \)
Inference Rules

### Primitive Inference Rules

**ASSUME**

\[ \{ t \} \vdash t \]

**REFL**

\[ \vdash t = t \]

**MP**

\[ \Gamma_1 \vdash t_1 \to t_2 \quad \Gamma_2 \vdash t_1 \]

\[ \Gamma_1 \cup \Gamma_2 \vdash t_2 \]

**DISCH**

\[ \Gamma \vdash t_2 \]

\[ \Gamma - \{ t_1 \} \vdash t_1 \to t_2 \]

**ABS**

\[ \Gamma \vdash t_1 = t_2 \]

\[ \Gamma \vdash (\lambda x. t_1) = (\lambda x. t_2) \] (with \( x \) not free in \( \Gamma \))
Inference Rules

Primitive Inference Rules (continued)

- **BETA_CONV**
  \[ \Gamma \vdash (\lambda x. t_1) t_2 = t_1[t_2/x] \]

- **SUBST**
  \[ \Gamma_1 \vdash t_1 = t_2 \quad \Gamma_2 \vdash t[t_1] \]
  \[ \Gamma_1 \cup \Gamma_2 \vdash t[t_2] \]

- **INST_TYPE**
  \[ \Gamma \vdash t \]
  \[ \Gamma \vdash t[\sigma_1 \ldots \sigma_n/\alpha_1 \ldots \alpha_n] \]
Beta Conversion

Rule for Beta-Conversion

\[
\text{BETA_CONV} \quad \frac{}{\vdash (\lambda x. t_1)t_2 = t_1[t_2/x]}
\]

- \( t_1[t_2/x] \) 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)

- \( \vdash (\lambda x. x + 3)7 = 7 + 3 \)
- \( \vdash (\lambda x. (\forall x. x = \text{true}) \rightarrow x) \text{false} = (\forall x. x = \text{true}) \rightarrow \text{false} \)
- \( \vdash (\lambda y. \forall x. x = y) \, x = (\forall x'. x' = x) \)
Rule for Substitution

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

- where \( t[t_1] \) is a term with selected free occurrences 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

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

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 \(\Gamma\).

Example (Type Instantiation)

\[
\begin{align*}
\vdash I(x : \alpha) &= x \\
\vdash I(x : \text{num}) &= x
\end{align*}
\]
Summary

**Higher-Order Logic**
- types and terms
- quantification over predicates, functions and sets

**HOL Proof System**
- five axioms and eight primitive inference rules