Overview of Higher Order Logic

Syntax

Higher Order Logic Notation

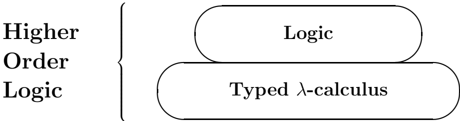
- **Conventional predicate calculus notation:**
  - x, y, z – variables
  - $\mathsf{T}, \mathsf{F}, 1$ – constants
  - $\mathrm{A} = \mathrm{B}$ – equality ('A equals B')
  - $\neg \mathrm{P}$ – negation ('not P')
  - $\mathrm{P} \wedge \mathrm{Q}$ – conjunction ('P and Q')
  - $\mathrm{P} \vee \mathrm{Q}$ – disjunction ('P or Q')
  - $\mathrm{P} \supset \mathrm{Q}$ – implication ('P implies Q')
  - $\forall x.\, \mathrm{P}$ – universal quantifier ('for all x, P')
  - $\exists x.\, \mathrm{P}$ – existential quantifier ('for some x, P')
- **Roughly speaking, 'higher order' means:**
  - variables can range over predicates and functions
  - functions can take functions as arguments
  - functions can return functions as results
- **Typical higher order statements:**

$$\forall P.\, (P(0) \wedge \forall m.\, P(m) \supset P(m{+}1)) \supset \forall n.\, P(n)$$

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

Church's Formulation

- **Church's formulation of higher order logic:**



- **Advantages of this approach:**
  - Very economical set of primitive notations and concepts (good for implementation).
  - The λ-calculus takes care of all *variable binding*.
  - The resulting logic incorporates the λ-calculus notation for functions.

Terms of the Typed λ-calculus

- **There are only four kinds of terms:**
  - constants (denote particular fixed values).
  - variables (range over sets of values).
  - applications (of functions to arguments).
  - abstractions (denote functions).
- **Syntax of raw terms:**

$$\mathrm{M} \;::=\; \underbrace{c}_{\text{constant}} \;\mid\; \underbrace{v}_{\text{variable}} \;\mid\; \underbrace{\mathrm{M}\;\;\mathrm{N}}_{\text{application}} \;\mid\; \underbrace{\lambda v.\, \mathrm{M}}_{\text{abstraction}}$$

Constants and Variables

- **Constants and variables are just identifiers:**

  x, y, foo, t', k_2, c_val, ...

  or certain special symbols:

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

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

Function Applications

- **An application looks like**

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

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

- **Parentheses can be used for grouping:**

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

- **By convention, application is left-associative:**

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

Abstractions

- **An abstraction looks like**

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

  and denotes the function: $x \mapsto \langle term \rangle\,[x/\langle var \rangle]$**.**

- **Examples:**

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

- **The rule of β-conversion:**

  $$\text{BETA_CONV:}\quad \frac{}{\vdash (\lambda x.\ \mathrm{M})\mathrm{N} = \mathrm{M}[\mathrm{N}/x]}$$

- **Notational convention:**

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

Free and Bound Variables

- **A variable x is *free* in a term if it does not occur inside the body of a subterm of the form $\lambda x.\langle body \rangle$.**

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

- **Example:** $(\lambda x.\ f\ x)(\lambda y.\ x)$ — first x in $f\ x$: bound; x in $\lambda y.\ x$: free

- **All variable binding in higher order logic is done using λ-abstractions.**

- **Examples of variable binding constructs:**
  - $\forall x.\, \mathrm{P}[x]$ ═ *represented by* ⟹ $\mathsf{All}(\lambda x.\, \mathrm{P}[x])$
  - $\exists x.\, \mathrm{P}[x]$ ═ *represented by* ⟹ $\mathsf{Some}(\lambda x.\, \mathrm{P}[x])$
  - $\sum_{i=1}^{n} f(i)$ ═ *represented by* ⟹ $\mathsf{Sum}\ 1\ n\ (\lambda i.\, f(i))$
  - $\int_0^\infty f(t)\,dt$ ═ *represented by* ⟹ $\mathsf{Int}(\lambda t.\, \mathrm{f}(t))$

Types

- **Every term has a *type* standing for the kind of value it denotes (number, function, etc).**
- **Motivation—types are used for**
  - **preventing logical inconsistency, e.g. by banning Russell's paradox:**

    $\exists P. \forall x. P(x) = \neg(x(x))$, hence $\exists P. P(P) = \neg(P(P))$!

  - **representing data (traditional CS data types):**

    $\forall n{:}num.\ \exists tr{:}tree.\ \mathsf{height}(tr) = 2 \times n$

    $\forall s{:}stack.\ \mathsf{pop}(\mathsf{push}\ x\ s) = s$

- **Some types available in HOL:**
  - *bool* – booleans
  - *num* – natural numbers
  - *integer* – integers
  - *real* – reals
  - (σ)*list* – lists of σs
  - (σ)*btree* – binary trees with σ labels

Syntax of Types

- **Syntax of types:**

$$\sigma \ ::= \ \underbrace{c}_{\text{type constant}} \ | \ \underbrace{\alpha}_{\text{type variable}} \ | \ \underbrace{(\sigma_1, \ldots, \sigma_n)op}_{\text{compound type}}$$

- **Examples of type constants:**

bool	**booleans**
num	**natural numbers**
real	**real numbers**

- **Type variables:** α, β, γ, ...
- **Examples of compound types:**

$(\sigma_1, \sigma_2)fun$	**functions from σ_1 to σ_2**
$(\sigma_1, \sigma_2)prod$	**pairs of values of types σ_1 and σ_2**
$(\sigma)list$	**lists of values of type σ**

Terminology and Notation

- **Type operators:**
  - **The '*op*' in $(\sigma_1, \ldots, \sigma_n)op$ is called a *type operator*.**
- **Notation:**
  - **The type $(\sigma_1, \sigma_2)fun$ is usually written $\sigma_1 \rightarrow \sigma_2$.**
  - **The type $(\sigma_1, \sigma_2)prod$ is usually written $\sigma_1 \times \sigma_2$.**
- **Notational conventions:**

$$\sigma_1 \rightarrow \sigma_2 \rightarrow \ldots \rightarrow \sigma_n = (\sigma_1 \rightarrow (\sigma_2 \rightarrow (\ldots \rightarrow \sigma_n)))$$

$$\sigma_1 \times \sigma_2 \times \ldots \times \sigma_n = (\sigma_1 \times (\sigma_2 \times (\ldots \times \sigma_n)))$$

Typing of Terms

- **All well-formed terms must be well-typed.**
- **Notation: '$\mathrm{M} : \sigma$' means that the term M is well-typed and has type σ.**
- **Variables and constants:**
  - **A variable $v : \sigma$ can have any type σ.**
  - **Every constant c has a fixed *generic* type $\mathrm{Gen}(\mathsf{c})$.**
- **Rules for assigning types to terms:**

Var: $$\frac{}{v : \sigma}$$

Con: $$\frac{}{\mathsf{c} : \sigma} \quad \sigma \text{ an instance of } \mathrm{Gen}(\mathsf{c})$$

App: $$\frac{\mathrm{M} : \sigma_1 \rightarrow \sigma_2 \qquad \mathrm{N} : \sigma_1}{(\mathrm{M}\ \mathrm{N}) : \sigma_2}$$

Abs: $$\frac{x : \sigma_1 \qquad \mathrm{M} : \sigma_2}{(\lambda x.\mathrm{M}) : \sigma_1 \rightarrow \sigma_2}$$

Polymorphism and Generic Types

- **The types of *polymorphic* functions such as ID contain type variables:**

  $\mathsf{ID} \stackrel{def}{=} (\lambda x.\, x){:}\alpha{\to}\alpha$

  where α stands for 'any type'.

- **Here, $\alpha{\to}\alpha$ is the *generic* type of ID.**

- **The constant ID then has every type you can get by substituting a type for the variable α in its generic type:**

  $\mathsf{ID} : bool \to bool$
  $\mathsf{ID} : num \to num$
  $\mathsf{ID} : (\alpha \to bool) \to (\alpha \to bool)$
  $\mathsf{ID} : \alpha \to \alpha$

  and so '$\mathsf{ID}\ 7$' and '$\mathsf{ID}\ \mathsf{T}$' are both well-typed.

Examples of Polymorphism

- **Definition of function composition:**

  $\circ \stackrel{def}{=} \lambda f.\lambda g.\lambda x.f(g(x))$

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

- **Equality is also polymorphic:**

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

- **Operations on lists:**

  $\mathsf{HD} : (\alpha)list \to \alpha$
  $\mathsf{TL} : (\alpha)list \to (\alpha)list$
  $\mathsf{NIL} : (\alpha)list$
  $\mathsf{CONS} : \alpha \to (\alpha)list \to (\alpha)list$

Summary — Typed λ-calculus

- **Terms may be**
  - **Variables:** $x,\ y,\ a',\ a_var,\ phi_1, \ldots$
  - **Constants:** $\mathsf{T},\ \mathsf{F},\ \mathsf{phi},\ \exists,\ +, \ldots$
  - **Applications:** $t_1\, t_2,\ \ t_1\, t_2\, t_3\, \ldots\, t_n$
  - **Abstractions:** $\lambda x.\, t,\ \ \lambda\, x_1\, x_2\, \ldots\, x_n.\, t$
- **Types may be:**
  - **Type constants:** $bool$, num, ...
  - **Type variables:** α, β, γ, ...
  - **Compound types:** $(\sigma_1, \ldots, \sigma_n)op$
- **Well-typed terms:**
  - $(\lambda x.t) : \sigma_1 \to \sigma_2$ **when** $x : \sigma_1$ **and** $t : \sigma_2$
  - $(t_1\ t_2) : \sigma$ **when** $t_1 : \sigma_1 \to \sigma$ **and** $t_2 : \sigma_1$
- **Polymorphism:**

  $\mathsf{twice} \stackrel{def}{=} \lambda f.\lambda x.f(f(x))$

  $\mathsf{twice} : (\alpha \to \alpha) \to (\alpha \to \alpha)$

Higher Order Logic

- **How logic is built on top of the λ-calculus:**
  - **Introduce a type constant** $bool$ **for propositions.**
  - **Introduce various constants for logic:**

    $\mathsf{T} : bool \quad \mathsf{F} ; bool \qquad = : \alpha \to \alpha \to bool$
    $\neg : bool \to bool \qquad \wedge : bool \to bool \to bool$
    $\vee : bool \to bool \to bool \qquad \supset : bool \to bool \to bool$
    $\forall : (\alpha \to bool) \to bool \qquad \exists : (\alpha \to bool) \to bool$

  - **Introduce any other types and constants required (e.g.** num, $+$, $\times$, $<$, ...**).**
  - **Introduce the other logical apparatus needed:**
    - axioms
    - rules of inference
    - principles of definition

- **Example of a proposition:**

  $\forall(\lambda n{:}num.\ \supset\ (>\ n\ 1)\ (>\ (\mathsf{Suc}\ n)\ 2))\ \ :\ \ bool$

  That is, $\forall n.\,(n > 1) \supset (\mathsf{Suc}\ n > 2)$.

Syntactic Sugar — Infix Applications

- **Certain (designated) constants with generic types of the form**

  $\sigma_1 \to (\sigma_2 \to \sigma_3)$

  can be written in infix position.

- **Some examples:**

$$\begin{array}{lll} \mathrm{A} = \mathrm{B} & =\!\textit{abbreviates}\!\Longrightarrow & (=\ \mathrm{A})\ \mathrm{B} \\ \mathrm{P} \wedge \mathrm{Q} & =\!\textit{abbreviates}\!\Longrightarrow & (\wedge\ \mathrm{P})\ \mathrm{Q} \\ \mathrm{N} + \mathrm{M} & =\!\textit{abbreviates}\!\Longrightarrow & (+\ \mathrm{N})\ \mathrm{M} \end{array}$$

- **Convention—infixes associate to the right:**

  $\mathrm{P}_1 \wedge \mathrm{P}_2 \wedge \ldots \wedge \mathrm{P}_n = \mathrm{P}_1 \wedge (\mathrm{P}_2 \wedge (\ldots \wedge \mathrm{P}_n))$

Syntactic Sugar — Binders

- **The quantifiers $\forall$ and $\exists$ are in fact polymorphic constants with generic types**

  $\forall : (\alpha \to \mathit{bool}) \to \mathit{bool}$

  $\exists : (\alpha \to \mathit{bool}) \to \mathit{bool}$

- **These constants are defined so that for any predicate $\mathrm{P} : (\alpha \to \mathit{bool})$**

  '$\forall$ P' means $\mathrm{P}(x) = \mathsf{T}$ for all x

  '$\exists$ P' means $\mathrm{P}(x) = \mathsf{T}$ for some x

Binder Notation

- **We could just use $\forall$ and $\exists$ as follows**

  $\forall(\lambda x.\ 0 \leq x)$ **and** $\exists(\lambda x.\ x = 2 + 3)$

  but it is more conventional to write

  $\forall x.\ 0 \leq x$ **and** $\exists x.\ x = 2 + 3$

- **So we simply follow the convention that:**

$$\begin{array}{lll} \forall x.\,\mathrm{P} & =\!\textit{abbreviates}\!\Longrightarrow & \forall\ (\lambda x.\,\mathrm{P}) \\ \exists x.\,\mathrm{P} & =\!\textit{abbreviates}\!\Longrightarrow & \exists\ (\lambda x.\,\mathrm{P}) \end{array}$$

- **Constants like $\forall$ and $\exists$ abbreviated in this way are called *binders*.**

- **Any (designated) constant with a generic type $(\sigma_1 \to \sigma_2) \to \sigma_3$ can be treated as a binder.**

- **Convention:**

$$\begin{array}{l} \forall x_1\ x_2\ \cdots\ x_n\,.\,\mathrm{P}\ =\ \forall x_1\,.\,\forall x_2\,.\ \cdots\ \forall x_n\,.\,\mathrm{P} \\ \exists x_1\ x_2\ \cdots\ x_n\,.\,\mathrm{P}\ =\ \exists x_1\,.\,\exists x_2\,.\ \cdots\ \exists x_n\,.\,\mathrm{P} \end{array}$$

Pairs and Tuples

- **The symbol ',' is an infix constant of type**

  $\alpha \to \beta \to (\alpha \times \beta)$

  where $\alpha \times \beta$ is the type of pairs (a, b) for which $a{:}\alpha$ and $b{:}\beta$.

- **Constants for selecting components:**

  $\mathsf{FST}(a, b) = a$
  $\mathsf{SND}(a, b) = b$

  where

  $\mathsf{FST} : (\alpha \times \beta) \to \alpha$
  $\mathsf{SND} : (\alpha \times \beta) \to \beta$

- **Convention:**

  $(x_1, x_2, \ldots, x_n)\ =\ (x_1, (x_2, (\ldots, x_n)))$

Summary of Syntactic Sugar

- **Infixes, for example:**

  $P \wedge Q \quad =\!=\textit{abbreviates}\Longrightarrow \quad (\wedge\, P)\, Q$

- **Binders, for example:**

  $\forall x.\, P \quad =\!=\textit{abbreviates}\Longrightarrow \quad \forall(\lambda x.\, P)$

- **Pairs and tuples:**

  $(t_1, t_2) : \sigma_1 \times \sigma_2,$ **where** $t_1{:}\sigma_1$ **and** $t_2{:}\sigma_2$

  $(t_1, t_2, t_3, \ldots) \;=\; (t_1, (t_2, (t_3, \ldots)))$

Formulas and Sequents

- **In Church's approach, a *formula* ('statement', 'proposition') is just a term of type *bool*.**
- **In HOL, the logic is formulated using *sequents*.**
- **A sequent is a pair of the form (Γ, P) where**
  - Γ is a set of formulas (assumptions) and
  - P is a formula (conclusion)
- **A sequent '(Γ, P)' means:**

  'if every formula in Γ is true, then so is P'

- **For example**

  $(\ \{\ n < m,\ m < p\ \},\ n < p\ )$

  is a sequent which means the same as

  $(n < m \ \wedge\ m < p) \ \supset\ n < p$

Axioms and Theorems

- **An *axiom* is just a sequent that is postulated to be true.**
- **A *theorem* is a sequent that is either**
  - an axiom, or
  - follows from other theorems by an *inference rule*.
- **For theorems we write**

  $\Gamma \vdash P$

  or just $\vdash P$ **if** Γ **is empty.**

Theories

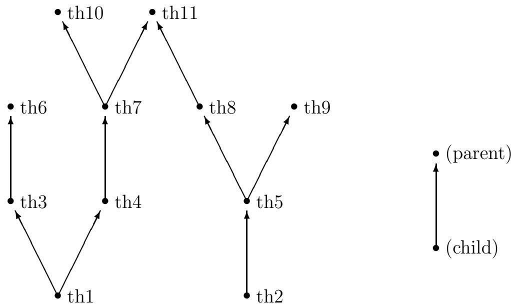
- **A *theory* is a collection of**
  - type constants
  - type operators
  - constants

    } the 'vocabulary' of the theory
  - axioms
  - theorems
- **A theory formally describes some domain of discourse (e.g. sets, groups, data structures, real arithmetic).**
- **Higher order logic itself is just a theory:**
  - type constants: *bool*, *ind*.
  - type operators: *fun* (i.e. '$\rightarrow$').
  - constants: $=$, $\supset$, ε, T, F, $\neg$, $\wedge$, $\vee$, $\forall$, $\exists$, $\exists!$.
  - axioms: ...to be described later.
  - theorems: ...facts derivable from the axioms.

  together with certain inference rules.

Theory Hierarchies

- Theories can have other theories as parents:



- Everything in the parent theory is also in the child and other descendent theories.
- Useful for structuring your work.
- Loops in the parent graph are not allowed.

Summary

- Formulas: these are just boolean terms.
- Sequents: a pair of the form (Γ, P).
- Theorems: written $\Gamma \vdash \mathrm{P}$ and $\vdash \mathrm{P}$.
- Theories: these are collections of
  - type constants
  - type operators
  - constants
  - axioms
  - theorems
- Higher order logic is:
  - a certain theory + some rules of inference