Higher-Order Types and Lambda-Expressions

Orders of a logical system

- Predicates that speak about domain objects are of
- Predicates that speak about objects of at most i-th by themselves of $(i+1)$-th order.
- Functions that take and return domain objects are o
- Functions that take and return objects of at most i- are by themselves of $(i+1)$-th order.

Examples of constructs involving higher-order are:

induction: $\forall P\ P(0) \wedge \forall n{:}\mathrm{Nat}\ P(n) \rightarrow P(n+1) \rightarrow \forall n{:}\mathrm{N}$

Differentiation: $(x.y)' = x' \cdot y + x \cdot y'$.

Statements involving functions: Every function that is 1 differentiable in the complex plane is infinitely often diff

Abstract Mathematical Structures Interpretations, num lattices, groups.

λ-notation

In usual mathematical notation, there is no good way fo representing functions, as a consequence functions and f confused.

The expression $x \cdot y^2$ can represent infinitely many func type Real $\rightarrow$ Real.

Differentation is a function of type (Real $\rightarrow$ Real) $\rightarrow$ (Re

Mathematicians, however speak of differentiating a *blue* after a variable.

Differentiation of $x \cdot y^2$ after x results in y^2. Differentati after y results in $2xy$. Differentation of $x \cdot y^2$ after z res

Identifying functions and formulas causes problems. Do
example, the rule $(x \cdot y)' = x' \cdot y + x \cdot y'$ also holds for fu
are not represented by formulas? What if you change n

As a consequence, some notation for functions is needed
purpose, the λ-notation is used. $\lambda x{:}\, X\, t[x]$ is the functio
every $n \in X$, has the value $t[n]$. If $t[x]$ has type Y on th
assumption that x has type X, then $\lambda x{:}\, X\, t[x]$ has type

Using λ-notation, it can be seen that there is only one
differentiation operator of type $(\text{Real} \to \text{Real}) \to (\text{Real} \cdot$
The different derivatives are obtained by different λ-abs

$$(\lambda x{:}\,\text{Real}\ x \cdot y^2)' = \lambda x{:}\,\text{Real}\ y^2,$$

$$(\lambda y{:}\,\text{Real}\ x \cdot y^2)' = \lambda y{:}\,\text{Real}\ x,$$

$$(\lambda z{:}\,\text{Real}\ x \cdot y^2)' = \lambda x{:}\,\text{Real}\ 0.$$

The differentiation rule can now be formulated as

$$\forall x, y{:}\,\text{Real} \to \text{Real}\ (x \cdot y)' = x' \cdot y + x \cdot y'.$$

In this expression, the $+$ and the $\cdot$ are not the usual op
numbers of type $[\text{Real}, \text{Real}] \to \text{Real}$, instead they are o
$[\text{Real} \to \text{Real}, \text{Real} \to \text{Real}] \to \text{Real}$.

For example, $+$ is defined as

$$\lambda f_1, f_2{:}\,\text{Real} \to \text{Real}\ \lambda x{:}\,\text{Real}\ f_1(x) + f_2(x).$$

β-Conversion 1

By definition, $\lambda x{:}\, X\, t[x]$ is the function that, for every n
assumes the value $t[n]$. Therefore

$$(\lambda x{:}\, X\, t[x]) \cdot n = t[n].$$

This equivalence is called β-equivalence. Applying the e
from left to right is called β-reduction.

β-Conversion 2

Nothing goes without β-reduction. If one instantiates

$$\forall x, y{:}\,\mathrm{Real} \to \mathrm{Real}\ (x \cdot y)' = x' \cdot y + x \cdot y'$$

by

$$\lambda x{:}\,\mathrm{Real}\ x, \text{ and } \lambda x{:}\,\mathrm{Real}\ x^2,$$

the result equals

$$((\lambda x{:}\,\mathrm{Real}\ x) \cdot (\lambda x{:}\,\mathrm{Real}\ x^2))' =$$

$$(\lambda x{:}\,\mathrm{Real}\ x)' \cdot (\lambda x{:}\,\mathrm{Real}\ x^2) + (\lambda x{:}\,\mathrm{Real}\ x) \cdot (\lambda x{:}\,\mathrm{Rea}$$

β-normalization (and applying the definitions of $+$ and

$$(\lambda x{:}\,\mathrm{Real}\ x^3)' = (\lambda x{:}\,\mathrm{Real}\ x)' \cdot (\lambda x{:}\,\mathrm{Real}\ x^2) + (\lambda x{:}\,\mathrm{Real}\ x) \cdot ($$

β-Conversion 3

Similarly, β-reduction is needed when applying inductio

If one instantiates in the induction axiom

$\forall P\ P(0) \wedge \forall n{:}\,\mathrm{Nat}\ P(n) \to P(n+1) \to \forall n{:}\,\mathrm{Nat}\ P(n)$,

the formula

$\lambda n{:}\,\mathrm{Nat}\ (\lambda x{:}\,\mathrm{Real}\ x^{n+1})' = (\lambda x{:}\,\mathrm{Real}\ (n+1).x^n)$,

the result equals

$(\lambda n{:}\,\mathrm{Nat}\ (\lambda x{:}\,\mathrm{Real}\ x^{n+1})' = (\lambda x{:}\,\mathrm{Real}\ (n+1).x^n))(0) \wedge$

$\forall n{:}\,\mathrm{Nat}\ \lambda n{:}\,\mathrm{Nat}\ (\lambda x{:}\,\mathrm{Real}\ x^{n+1})' = (\lambda x{:}\,\mathrm{Real}\ (n+1).x^n)($

$\lambda n{:}\,\mathrm{Nat}\ (\lambda x{:}\,\mathrm{Real}\ x^{n+1})' = (\lambda x{:}\,\mathrm{Real}\ (n+1).x^n)(n$

$\to$

$\forall n{:}\,\mathrm{Nat} \lambda n{:}\,\mathrm{Nat}\ (\lambda x{:}\,\mathrm{Real}\ x^{n+1})' = (\lambda x{:}\,\mathrm{Real}\ (n+1).x^n)($

After β-reduction, this monster equals

$$(\lambda x{:}\,\mathrm{Real}\ x)' = (\lambda x{:}\,\mathrm{Real}\ 1) \wedge$$

$$\forall n{:}\,\mathrm{Nat}\ (\lambda x{:}\,\mathrm{Real}\ x^{n+1})' = (\lambda x{:}\,\mathrm{Real} (n+1) \cdot x^n) \to$$
$$(\lambda x{:}\,\mathrm{Real}\ x^{n+2})' = (\lambda x{:}\,\mathrm{Real} (n+2) \cdot x^{n+2}) \to$$

$$\forall n{:}\,\mathrm{Nat}\ (\lambda x{:}\,\mathrm{Real}\ x^{n+1})' = (\lambda x{:}\,\mathrm{Real} (n+1) \cdot x$$

One can guess why the λ-notation never got popular. F computers are good at making such replacements.

Free and Bound Variables

In the λ-term

$$\lambda x{:}\,X\ t,$$

we call x the bound variable, X the type, and t the bod term.

The λ binds all the occurrences of x in its body. (Not i

A variable, occurring somewhere in a λ-term is bound i λ that binds it. Otherwise it is free.

α-Variants

Informally, two terms t_1 and t_2 are α-variants, if they h
same meaning. Two λ-terms t_1 and t_2 are α-variants if
following are true:

1. If somewhere in t_1 there is a subterm built by $\cdot$, the
   corresponding position in t_2, there is also a subterm
2. If somewhere in t_1 there is a subterm built by λ, th
   corresponding position in t_2, there is also a subterm
3. If a variable occurring somewhere in t_1 is free in t_1,
   corresponding position in t_2 occurs the same variab
   also free in t_2.
4. If a variable x occurring somewhere in t_1 is not free,
   corresponding position in t_2, there is also a variable
   also not free, and both are bound by λ's at the sam

Substitution

Substitution is the essentially the replacement of all occ
a free variable x of a term t by some term u.

BUT Free variable of u may get caught by λ's inside t.

THEREFORE First replace t by an α-variant, where
not happen. Such a variant always exists, when you hav
many variables.

The notation is

$$t[x := u].$$

Examples of Substitution

$$\lambda n{:}\mathrm{Nat}\ (+\ n\ m)[m := 1] \text{ equals } \lambda n{:}\mathrm{Nat}\ (+\ n\ 1$$

$$\lambda n{:}\mathrm{Nat}\ (+\ n\ m)[m := (+\ m\ 1)] \text{ equals } \lambda n{:}\mathrm{Nat}\ (+\ n\ ($$

$$\lambda n{:}\mathrm{Nat}\ (+\ n\ m)[m := (+\ n\ 1)] \text{ equals } \lambda z{:}\mathrm{Nat}\ (+\ z\ ($$

The term

$$\lambda n{:}\mathrm{Nat}\ (+\ n\ (+\ n\ 1)).$$

would be a totally different function.

Equivalences

Based on the intended meanings of the λ-terms, a coupl
equivalence relations can be defined. They are usually c
and η-equivalence. (Nobody seems to worry about the r
γ-equivalence)

One of them we defined already, namely α-equivalence:

$$t_1 \equiv_\alpha t_2$$

if t_1 can be obtained from t_2 by renaming bound variab

β and η-equivalence

Another type of equivalence is β-equivalence, which is d
follows:

$$t_1 \equiv_\beta t_2,$$

if t_2 can be obtained from t_1 by replacing a subterm of

$$(\lambda x{:}X\ f) \cdot t$$

by

$$f[x := t].$$

η-equivalence is defined as follows:
$t_1 \equiv_\eta t_2$ if t_2 is obtained from t_1 by replacing a subterm

$$(\lambda x{:}X\ (f\ x))$$

by f. It must be the case that x is not free in f.

δ-equivalence

The last equivalence is δ-equivalence. It is based on exp
definitions. Because of this, contrary to the other equiva
depends on a context C.

If context C contains a definition $x := y{:}Y$, then

$$t_1 \equiv_\delta t_1[x := y].$$

If x is defined as y, then it is allowed to replace x by y.

$\alpha, \beta, \delta, \eta$-equivalence

We write

$$C \vdash t_1 \equiv_{\alpha,\beta,\delta,\eta} t_2$$

if t_1 can be obtained from t_2 by finitely often applying t
-replacement rules, in either direction, and on every sub

Typing Rules

We assume that the following types are given:

Form : The type of formulae.
Type : The type of types.
Kind : The type of Form and Type, needed for technica

The objects Form, Type, Kind are called sorts. We write
set of sorts, that is $S = \{\text{Form}, \text{Type}, \text{Kind}\}$.

We now need the following:

1. Rules for determining the type of a λ-term if there
2. Rules for determining whether or not a context is w

Rules for well-formedness of Contexts

The empty context is well-formed.

DECL: If C is well-formed, x is a variable, not occurri
$C \vdash X{:}T$, where T is a sort, then

$$C, \quad x{:}X$$

is well-formed.

DEF: If C is well-formed,
$C \vdash y{:}Y$, and
x is a variable, not occurring in C, then

$$C, \quad x := y{:}Y$$

is well-formed.

Rules for determining the type of a λ-term

In the rules, we implicitly assume that all contexts are v

SORT: For every context C, we have
$C \vdash \text{Type}{:}\text{Kind}$ and
$C \vdash \text{Form}{:}\text{Kind}$.

AXIOM: $C,\ x{:}X \vdash x{:}X$.
$C,\ x := y{:}Y \vdash x{:}Y$.

WEAKENING: If $C \vdash x{:}X$, then $C, D \vdash x{:}X$, for eve
definition or declaration D.

APPL: If $C \vdash t{:}X$, and
$C \vdash f{:}\Pi x{:}X\ Y$, then $C \vdash (f \cdot t){:}(Y[x := t])$.

LAMBDA: If $C,\ x{:}X \vdash y{:}Y$, then
$C \vdash (\lambda x{:}X\ y){:}(\Pi x{:}X\ Y)$.
Note that $C, x{:}X$ has to be well-formed.

PI: If $C, x{:}X \vdash Y{:}T$, where T is a sort, then $C \vdash (\Pi x{:}X$
(Remember that $C, x{:}X$ must be well-formed)

EQUIV: If $C \vdash x{:}X_1$,
$C \vdash X_1 \equiv_{\alpha\beta\delta\eta} X_2$, and
$C \vdash X_2{:}T$, with T a sort, then $C \vdash x{:}X_2$.

Sequent Calculus for Higher-Order Logic

Sequents have form $\Gamma \vdash_C \Delta$, where C is a well-formed c that

- C contains declarations $\rightarrow\colon \mathrm{Form} \Rightarrow \mathrm{Form}$
  $\forall\colon \Pi X\colon \mathrm{Type}\ (X \Rightarrow \mathrm{Form}) \Rightarrow \mathrm{Form}$
- Each element of Γ, Δ has type Form in C.

Formulae can be freely replaced by formulae that are $\alpha, \beta, \delta, \eta$-equivalent.

Axioms:

$$(\text{axiom})\ \frac{}{A \vdash_C A}$$

on the condition that $C \vdash A\colon \mathrm{Form}$.

Structural Rules:

$$(\text{weakening left})\ \frac{\Gamma \vdash_C \Delta}{\Gamma, A \vdash_C \Delta} \qquad (\text{weakening right})$$

$$(\text{contraction left})\ \frac{\Gamma, A, A \vdash_C \Delta}{\Gamma, A \vdash_C \Delta} \qquad (\text{contraction right})\ \frac{\Gamma$$

Rules for $\rightarrow$:

$$(\rightarrow\text{-left})\ \frac{\Gamma \vdash_C \Delta, A \qquad \Gamma, B \vdash_C \Delta}{\Gamma, A \rightarrow B \vdash_C \Delta}$$

$$(\rightarrow\text{-right})\ \frac{\Gamma, A \vdash_C \Delta, B}{\Gamma \vdash_C \Delta, A \rightarrow B}$$

Rules for $\forall$:

$$(\forall\text{-left})\ \frac{\Gamma,\ F[x := t] \vdash_C \Delta}{\Gamma,\ \forall x\colon X\ F \vdash_C \Delta}$$

It must be the case that $C \vdash t\colon X$.

$$(\forall\text{-right})\ \frac{\Gamma \vdash_{C'} \Delta,\ F[x := y]}{\Gamma \vdash_C \Delta,\ \forall x\colon X\ F}$$

The y must be a variable that is not free in $\Gamma, \Delta,\ \forall x\colon X$

C' has form $C,\ y\colon X$.

Natural Deduction for Higher-Order Logic

In natural deduction for HOL, the context and the assu mixed. As was the case with sequent calculus, formulae freely replaced by formulae that are $\alpha, \beta, \delta, \eta$-equivalent.

$\rightarrow$-introduction:

$$\dfrac{\dfrac{A}{\begin{array}{l}\cdots\\ B\end{array}}}{A \rightarrow B}$$

$\rightarrow$-elimination:

$$\begin{array}{l} A\\ \cdots\\ A \rightarrow B\\ \cdots\\ B \end{array}$$

$\forall$-introduction:

$$\dfrac{\dfrac{y{:}X}{\begin{array}{l}\cdots\\ F[x := y]\end{array}}}{\forall x{:}X\ F.}$$

$\forall$-elimination:

$$\begin{array}{l} \forall x{:}X\ F\\ \cdots\\ F[x := t] \end{array}$$

Term t must have type X at the point where $F[x := t]$ i introduced.

Intuitionistic vs. Classical

The situation is the same as with first order logic:

- Natural Deduction defines intuitionistic Higher Ord One can obtain classical Higher Order Logic from tl adding the law of excluded middle.
- Sequent Calculus defines classical Higher Order Log Intuitionistic Higher Order Logic can be obtained b sequents with more than one formula on the right.

Definition of Logical Operators in HOL

Both calculi contain only rules for $\forall$ and $\rightarrow$. Other logi operators can be defined or declared when they are neec

The following distinction is often made:

Meta Logic The logic used for checking the proofs. Ne HOL.

Object Logic The logic that the user is really interest doing his proofs, expressing his theorems.

So usually, the object logic is defined in HOL, after that is defined in the object logic.

Embedding of FOL into HOL

First order logic can be defined in two ways:

By declaration One declares the logical operators and axioms that describe their behaviour.

By definition One defines the logical operators by ter behave in the way the operators should.

Declaration of Logical Operators

$\bot$: Form
$\top$: Form
$\neg$: Form $\Rightarrow$ Form
$\vee$: Form $\Rightarrow$ Form $\Rightarrow$ Form
$\wedge$: Form $\Rightarrow$ Form $\Rightarrow$ Form
$\leftrightarrow$: Form $\Rightarrow$ Form $\Rightarrow$ Form

$\exists$: ΠD: Set P: $D \Rightarrow$ Form $(P \Rightarrow$ Form$) \Rightarrow$ Form
$\approx$: ΠD: Set $D \Rightarrow D \Rightarrow$ Form

Axioms for Logical Operators (1)

$\forall P$: Form $\bot \rightarrow P$

$\top$

$\forall P$: Form $(P \rightarrow \bot) \rightarrow \neg P$
$\forall P$: Form $P \rightarrow (\neg P) \rightarrow \bot$

Axioms for Logical Operators (2)

$\forall P, Q{:}\,\text{Form}\ P \to Q \to P \wedge Q$
$\forall P, Q{:}\,\text{Form}\ P \wedge Q \to P$
$\forall P, Q{:}\,\text{Form}\ P \wedge Q \to Q$

$\forall P, Q{:}\,\text{Form}\ P \to P \vee Q$
$\forall P, Q{:}\,\text{Form}\ Q \to P \vee Q$
$\forall P, Q{:}\,\text{Form}\ P \vee Q \to \forall C{:}\,\text{Form}\ (P \to C) \to (Q \to C) \to$

$\forall P, Q{:}\,\text{Form}\ (P \to Q) \to (Q \to P) \to (P \leftrightarrow Q)$
$\forall P, Q{:}\,\text{Form}\ (P \leftrightarrow Q) \to (P \to Q)$
$\forall P, Q{:}\,\text{Form}\ (P \leftrightarrow Q) \to (Q \to P)$

Axioms for Logical Operators (3)

$\forall D{:}\,\text{Type}\ \forall P{:}\,D \Rightarrow \text{Form}\ \forall d{:}\,D\ (P\ d) \to (\exists\ D\ P)$
$\forall D{:}\,\text{Type}\ \forall P{:}\,D \Rightarrow \text{Form}$
$(\exists\ D\ P) \to \forall C{:}\,\text{Form}\ (\forall d{:}\,D\ (P\ d) \to C) \to C$

$\forall D{:}\,\text{Type}\ (\approx\ D\ d\ d)$
$\forall D{:}\,\text{Type}\ \forall d_1, d_2{:}\,D \Rightarrow \text{Form}$
$(\approx\ D\ d_1\ d_2) \to \forall P{:}\,D \Rightarrow \text{Form}\ (P\ d_1) \to (P$

Definitions of Logical Operators

Defining the Logical Operators has the advantage that t
operators are invisible, because definitions can be expan
without additional proof steps. This makes the proofs u
shorter. Unfortunately proofs also become harder to un

$\bot := \Pi P{:}\,\text{Form}{:}\,\text{Form}$
$\neg := \lambda P{:}\ \text{Form}\ (P \to \bot){:}\ \text{Form} \Rightarrow \text{Form}$
$\vee := \lambda P, Q{:}\,\text{Form}$
$\forall R{:}\,\text{Form}\ (P \to R) \to (Q \to R) \to R{:}\,\text{Form} \Rightarrow$

$\wedge := \lambda P, Q{:}\,\text{Form}$
$\forall R{:}\,\text{Form}\ (P \to Q \to R) \to R{:}\,\text{Form} \Rightarrow \text{Form} =$

$\leftrightarrow := \lambda P, Q{:}\,\text{Form}\ (P \to Q) \wedge (Q \to P){:}\,\text{Form} \Rightarrow \text{Form} \Rightarrow$

$\exists := \lambda D{:}\,\text{Type}\ \lambda P{:}\,D \to \text{Form}$
$\forall Q{:}\,\text{Form}\ (\forall d{:}\,D\ (P\ d) \to Q) \to Q$
$:\Pi D{:}\,\text{Type}\ \Pi P{:}\,D \Rightarrow \text{Form}\ (P \Rightarrow \text{Form}) \Rightarrow$

$\approx := \lambda D{:}\,\text{Type}\ \lambda d_1, d_2{:}\,D$
$\forall P{:}\,D \Rightarrow \text{Form}\ (P\ d_1) \to (P\ d_2){:}\,\Pi D{:}\,\text{Type}\ D$

An alternative definition is the following:
$\approx := \lambda D{:}\,\text{Type}\ \lambda d_1, d_2{:}\,D$
$\forall P{:}\,D \Rightarrow D \Rightarrow \text{Form}\ \forall d{:}\,D\ (P\ d\ d) \to (P\ d_1\ d$
$\Pi D{:}\,\text{Type} D \Rightarrow D \Rightarrow \text{Form}$