Type Theory, 0

Samstag, 15. Oktober 2022 18:52

Mario Weitzer

0. Computation and proof

·) Turing machines https://morphett.info/turing.html

- copy number: x-x-x → x-x-x--x-x-x
- add two numbers: x-x-x--x-x → x-x-x--x-x--x-x-x-x-x
- 3n+1 problem: $3 \xrightarrow{3n+1} 10 \xrightarrow{n:2} 5 \to 16 \to 8 \to 4 \to 2 \to 1$ ← final destination? ($3 \in \mathbb{N}$)

  TM halts ⇔ counterexample exists
- Goldbach conjecture, Riemann hypothesis, consistency of ZFC ⟷ halting of TM
- Turing completeness, Church-Turing thesis: inventor of λ-calculus, inventor of TMs

  Turing computability fundamental property of this universe?
- halting problem, busy beaver:

  Fundamental limits on computation and knowledge in this universe?

·) Physical computers YouTube: Ben Eater

transistor → logic gates → {binary adder, SR-latch (memory)} → Von Neumann architecture

·) Game of life

- glider → glider gun → logic gates → {computer, universal TM}
- GOL in GOL (cf. C++ quine)

·) Exotic computation and Turing completeness

Minecraft computer, Flappy Bird in Super Mario World, spectral gap of molecules

·) Computing ⟷ proving

- "compute" proofs with natural deduction (Modus Ponens, ∀-intro, ∀-elim, …) or ZFC set theory (axiom of extensionality, …) (proof: meta object)
- Curry-Howard-isomorphism: programs are proofs (very clear in typed λ-calculus) and vice versa!
- Gödel's incompleteness theorems follow from halting problem
- truth of law of excluded middle ⟷ movement of bishops in chess

  Logic is "just" a game which happens to model the validity of arguments

Type Theory, 1

Donnerstag, 13. Oktober 2022 09:47

1. Untyped Lambda Calculus λ

Inspiration: modelling of behavior of functions on most basic level

Objects of set theory: sets only, $\underbrace{1 \cup (\leq \setminus +) \cap \mathbb{R}}_{=?}$ is a valid expression

... depends on implementation of $1, \leq, +, \mathbb{R}$

Objects of λ-calculus: functions only

Basic operations in λ-calculus: application and abstraction

Definition 1.1: the set Λ of all λ-terms

Let V be an infinite set ... the set of "variables".

capital lambda → Let Λ be inductively defined by

(1) (variable) If $x \in V$, then $x \in \Lambda$

(2) (application) If $A, B \in \Lambda$, then $(A\,B) \in \Lambda$

(3) (abstraction) If $x \in V$, $A \in \Lambda$, then $(\lambda x. A) \in \Lambda$.

Short form: $\Lambda = V \mid (\Lambda\,\Lambda) \mid (\lambda V. \Lambda)$

("abstract syntax"/"grammar")

Example 1.2: $\overset{(1)}{x}, \overset{(1)}{y}, \overset{(1)}{z}, \overset{(2)}{(x\,x)}, \overset{(2)}{(x\,(x\,z))}, \overset{(3)}{(\lambda x.(x\,z))}, \overset{(2)}{(y\,(\lambda x.(x\,z)))} \in \Lambda$

Notation 1.3: elements of V: $a, b, c, x, x', x'', y_1, y_2, y_3, \dots$

elements of Λ: $A, B, C, X, X', X'', Y_1, Y_2, Y_3, \dots$

Definition 1.4: syntactical identity $\equiv$

inductive definition of $\equiv$:

(1) (var) $x \equiv x$, $x \not\equiv y$ (only property of a variable is its name)

(2) (app) $(A\,B) \equiv (C\,D)$ iff $A \equiv C$ and $B \equiv D$

(3) (abst) $(\lambda x.A) \equiv (\lambda x.B)$ iff $A \equiv B$

$(\lambda x.A) \not\equiv (\lambda y.A)$

Example 1.5: $(x\,y) \equiv (x\,y) \not\equiv (x\,z), (\lambda x.x) \not\equiv (\lambda x.y) \equiv (\lambda x.y) \not\equiv (\lambda y.y) \not\equiv (\lambda x.x)$

Type Theory, 2

Samstag, 15. Oktober 2022 16:48

identical elements may occur multiple times

Definition 1.6: the multiset Sub of subterms of a λ-term

(1) (var) $Sub(x) = \{x\}$

(2) (app) $Sub((AB)) = Sub(A) \cup Sub(B) \cup \{(AB)\}$

(3) (abst) $Sub((\lambda x.A)) = Sub(A) \cup \{(\lambda x.A)\}$

$A \in \Lambda$ is called a (proper) subterm of $B \in \Lambda$ if $A \in Sub(B)$ (and $A \not\equiv B$)

Lemma 1.7: (1) (reflexivity) $A \in Sub(A)$

(2) (transitivity) If $A \in Sub(B)$, $B \in Sub(C)$, then $A \in Sub(C)$

Proof: follows trivially by induction.

not really a tree from graph theory, embedding matters

Example and definition 1.8: "tree" of subterms

$Sub((y(\lambda x.(xz)))) = Sub(y) \cup Sub((\lambda x.(xz))) \cup \{(y(\lambda x.(xz)))\}$

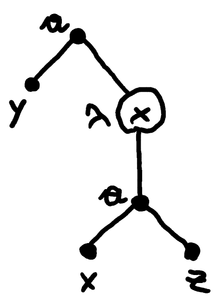
$= \{y\} \cup Sub((xz)) \cup \{(\lambda x.(xz))\} \cup \{(y(\lambda x.(xz)))\}$

$= \{y\} \cup Sub(x) \cup Sub(z) \cup \{(xz)\} \cup \{(\lambda x.(xz))\} \cup$

$\cup \{(y(\lambda x.(xz)))\}$

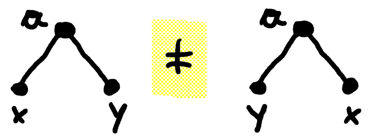
$= \{y, x, z, (xz), (\lambda x.(xz)), (y(\lambda x.(xz)))\}$

Tree of subterms of $(y(\lambda x.(xz)))$:



subterms $\longleftrightarrow$ nodes

embedding matters:



$Sub((xy)) = \{x, y, (xy)\}$ $\qquad$ $Sub((yx)) = \{y, x, (yx)\}$

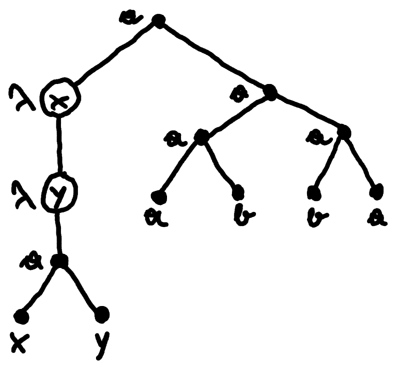
Type Theory, 3

Samstag, 15. Oktober 2022 17:42

Exercises 1

1.1: Find all subterms of $A \equiv ((\lambda x.(\lambda y.(xy)))((a\,b)(b\,a)))$ and draw the tree of subterms of A.

$$
\begin{aligned}
\mathrm{Sub}(A) &= \mathrm{Sub}((\lambda x.(\lambda y.(xy)))) \cup \mathrm{Sub}(((a\,b)(b\,a))) \cup \{A\} \\
&= \mathrm{Sub}((\lambda y.(xy))) \cup \{(\lambda x.(\lambda y.(xy)))\} \cup \mathrm{Sub}((a\,b)) \cup \\
&\quad \cup \mathrm{Sub}((b\,a)) \cup \{((a\,b)(b\,a))\} \cup \{A\} \\
&= \mathrm{Sub}((xy)) \cup \{(\lambda y.(xy))\} \cup \{(\lambda x.(\lambda y.(xy)))\} \cup \mathrm{Sub}(a) \cup \\
&\quad \cup \mathrm{Sub}(b) \cup \{(a\,b)\} \cup \mathrm{Sub}(b) \cup \mathrm{Sub}(a) \cup \{(b\,a)\} \\
&\quad \cup \{((a\,b)(b\,a))\} \cup \{A\} \\
&= \{x, y, (xy), (\lambda y.(xy)), (\lambda x.(\lambda y.(xy))), a, b, (a\,b), b, a, \\
&\quad (b\,a), ((a\,b)(b\,a)), ((\lambda x.(\lambda y.(xy)))((a\,b)(b\,a)))\}
\end{aligned}
$$



λ-calculus only considers functions in one variable.
For multiple variables: Currying
Curried version of $f : \mathbb{R}^2 \to \mathbb{R}$, $(x,y) \mapsto x+y$

$$
\begin{aligned}
f^{(c)} : \mathbb{R} &\to \mathrm{Func}(\mathbb{R}, \mathbb{R}) \\
x &\mapsto f_x : \mathbb{R} \to \mathbb{R} \\
&\qquad\quad y \mapsto x+y
\end{aligned}
$$

$f(x,y) = f^{(c)}(x)(y)$

Notation 1.9: associativity and precedence guided by idea of "Currying"

read: "stands for"

(1) drop outermost parenthesis: $AB \equiv (AB)$, $\lambda x.A \equiv (\lambda x.A)$

(2) application is left associative: $ABC \equiv ((AB)C)$

(3) abstraction is right associative, use only one λ: $\lambda xy.A \equiv \lambda x.(\lambda y.A)$

(4) application takes precedence over abstraction: $\lambda x.AB \equiv \lambda x.(AB)$

Example 1.10: $((((\lambda x.(\lambda y.(\lambda z.((y z) x)))) a) b) c) \equiv (\lambda xyz.yzx)\,a\,b\,c =_\beta b\,c\,a$

spoiler

Type Theory, 4

Mittwoch, 19. Oktober 2022 10:41

Definition 1.11: free, bound, binding variables

Let the sets FV, BV, and BiV be inductively defined by

(1)(var) $FV(x)=\{x\}$ $\quad BV(x)=\{\}$ $\quad BiV(x)=\{\}$

(2)(app) $FV(AB)=FV(A)\cup FV(B)$ $\quad BV(AB)=BV(A)\cup BV(B)$ $\quad BiV(AB)=BiV(A)\cup Bi(B)$

(3)(abst) $FV(\lambda x.A)=FV(A)\setminus\{x\}$ $\quad BV(\lambda x.A)=BV(A)\cup(\{x\}\cap FV(A))$ $\quad BiV(\lambda x.A)=BiV(A)\cup\{x\}$

A variable x is called free / bound / binding in a λ-term A if $x\in FV(A)$ / $x\in BV(A)$ / $x\in BiV(A)$

Example 1.12: $FV(\lambda x.xy)=FV(xy)\setminus\{x\}=(FV(x)\cup FV(y))\setminus\{x\}=\{x,y\}\setminus\{x\}=\{y\}$

$FV(x(\lambda x.xy))=FV(x)\cup FV(\lambda x.xy)=\{x\}\cup\{y\}=\{x,y\}$

$BV(\lambda x.x)=BV(x)\cup(\{x\}\cap FV(x))=\{\}\cup\{x\}=x$, but $BV(\lambda x.y)=\{\}$

$BiV(\lambda x.y)=BiV(y)\cup\{x\}=\{\}\cup\{x\}=\{x\}$

Definition 1.13: closed λ-term, combinator, Λ°

A λ-term A is called closed or combinator if $FV(A)=\{\}$

$\Lambda^\circ:=\{A\in\Lambda \mid A \text{ closed}\}$

Definition 1.14: $A^{x\to y}$, renaming, alpha conversion, $=_\alpha$

For $A\in\Lambda$, $x,y\in V$ let $A^{x\to y}$ denote the λ-term obtained by replacing every free occurence of x in A by y: $x(\lambda x.xy)^{x\to y}\overset{\text{"stands for"}}{=}y(\lambda x.xy)$

For $A\in\Lambda$, $x,y\in V$ with $y\notin FV(A)\cup Bi(A)$ we define the relation "renaming"
i.e. y does not occur in A
by $\lambda x.A=_\alpha\lambda y.A^{x\to y}$ and say "$\lambda x.A$ has been renamed as $\lambda y.A^{x\to y}$".

Extend $=_\alpha$ to full alpha conversion:

(1)(renaming) If $y\notin FV(A)\cup Bi(A)$, then $\lambda x.A=_\alpha\lambda y.A^{x\to y}$

(2)(compatibility) If $A=_\alpha B$, then $AC=_\alpha BC$, $CA=_\alpha CB$, $\lambda z.A=_\alpha\lambda z.B$

(3a)(reflexivity) $A=_\alpha A$

(3b)(symmetry) If $A=_\alpha B$, then $B=_\alpha A$

(3c)(transitivity) If $A=_\alpha B$ and $B=_\alpha C$, then $A=_\alpha C$

If $A=_\alpha B$, then A and B are said to be "alpha convertibles" or "alpha equivalent".

Type Theory, 5

Mittwoch, 19. Oktober 2022 13:30

Remark 1.15: Conditions on renaming preserve status of free and bound variables.

If $y \in FV(A)$: $\lambda x.y$ (free) would become $\lambda y.y$ (bound)

$(\lambda x.y)z =_\beta y$ $\qquad$ $(\lambda y.y)z =_\beta z$

If $y \in BiV(A)$: $\lambda xy.x$ (bound to first λ) would become $\lambda yy.y$ (bound to second λ)

$(\lambda xy.x)z =_\beta \lambda y.z$ $\qquad$ $(\lambda yy.y)z =_\beta \lambda y.y$

Remark and notation 1.16: $\mathcal{L}$-equivalent terms have identical subterm trees apart from the names of binding variables and show exactly the same pattern of free, bound, and binding variables and consequently the same behavior under β-reduction (defined soon).

From now on we consider $\mathcal{L}$-equivalent terms to be syntactically identical: $=_\mathcal{L}$ " = " $\equiv$

Substitution: the basis of computation in λ-calculus

Definition 1.17: substitution

(1)(var) $x[x:=A] = A$, $y[x:=A] = y$

(2)(app) $(BC)[x:=A] = (B[x:=A])(C[x:=A])$

(3)(abst) $(\lambda x.B)[x:=A] = \lambda x.B$

to avoid capturing free variables in A

$(\lambda y.B)[x:=A] = \lambda z.(B^{y\to z}[x:=A])$ where $z \notin FV(A)$

depends on choice of new variable, problem solved by 1.16

Remark 1.18: $A^{x\to y}$ and $B[x:=A]$ are not λ-terms but meta terms, something which stands for a λ-term

no risk of capturing, nothing to substitute

If $y \notin FV(A)$ or $x \notin FV(B)$, then (3) simplifies to

$(\lambda y.B)[x:=A] = \lambda y.(B[x:=A])$

also $\equiv$ by 1.16

Renaming is a special case of substitution: $A^{x\to y} =_\mathcal{L} A[x:=y]$

Example 1.19: $(\lambda y.yx)[x:=xy] \equiv (\lambda z.zx)[x:=xy] \equiv \lambda z.((zx)[x:=xy])$
$\equiv \lambda z.((z[x:=xy])(x[x:=xy])) \equiv \lambda z.z(xy)$

Notation 1.20: Barendregt convention: choose names of binding variables such that they are different from each other and all free variables: $(\lambda xy.xz)(\lambda xz.z) \to (\lambda xy.xz)(\lambda uv.v)$

Exercises 2

2.1: Remove brackets and combine λ-abstractions in the following λ-terms in accordance with Notation 1.9:

(a) $(\lambda x.(((xz)y)(xx)))$

(b) $((\lambda x.(\lambda y.(\lambda z.(z((xy)z)))))(\lambda u.u))$

$(\lambda x.(((xz)y)(xx))) = \lambda x.xzy(xx)$

$((\lambda x.(\lambda y.(\lambda z.(z((xy)z)))))(\lambda u.u)) = (\lambda xyz.z(xyz))(\lambda u.u)$

2.2: Mark all free, bound, and binding variables in

$ab(\lambda u.uv)x(\lambda xyz.(xy)(\lambda x.bx(zb))y)c$

$ab(\lambda u.uv)x(\lambda xyz.(xy)(\lambda x.bx(zb))y)c$ ... free, bound, binding

2.3: Which of the following λ-terms are α-equivalent to $(\lambda x.x(\lambda z.xy))z$:

(a) $(\lambda z.z(\lambda x.zy))z$ (b) $(\lambda y.y(\lambda z.yy))z$

(c) $(\lambda z.z(\lambda z.zy))z$ (d) $(\lambda u.u(\lambda z.uy))v$

(a) ✓ (b) × (c) × (d) ×

2.4: Which of the following λ-terms are α-equivalent to $(\lambda z.zxz)((\lambda y.xy)x)$:

(a) $(\lambda y.yxy)((\lambda z.xz)x)$ (b) $(\lambda x.xyx)((\lambda z.yz)y)$

(c) $(\lambda y.yxy)((\lambda y.xy)x)$ (d) $(\lambda u.(ux)u)((\lambda v.vu)x)$

(a) ✓ (b) × (c) ✓ (d) ×

Type Theory, 7

Freitag, 21. Oktober 2022 21:12

2.5: Perform the following substitutions:

(a) $((\lambda y.yyx)[x:=yz]$

(b) $(\lambda x.y(\lambda y.xy))[y:=\lambda z.zx]$

(c) $((xyz)[x:=y])[y:=z]$

$((xyz)[y:=z])[x:=y]$

$((xyz)[y:=z])[x:=y[y:=z]]$

$$
\begin{aligned}
((\lambda y.yyx)[x:=yz] &= ((\lambda u.uux)[x:=yz] \\
&= \lambda u.((uux)[x:=yz]) \\
&= \lambda u.((uu)[x:=yz])(x[x:=yz]) \\
&= \lambda u.((u[x:=yz])(u[x:=yz]))(yz) \\
&= \lambda u.uu(yz)
\end{aligned}
$$

$$
\begin{aligned}
&(\lambda x.y(\lambda y.xy))[y:=\lambda z.zx] = \\
&= (\lambda u.y(\lambda y.uy))[y:=\lambda z.zx] \\
&= \lambda u.((y(\lambda y.uy))[y:=\lambda z.zx]) \\
&= \lambda u.(y[y:=\lambda z.zx])((\lambda y.uy)[y:=\lambda z.zx]) \\
&= \lambda u.(\lambda z.zx)(\lambda y.uy)
\end{aligned}
$$

$$
\begin{aligned}
((xyz)[x:=y])[y:=z] &= (yyz)[y:=z] = zzz \\
((xyz)[y:=z])[x:=y] &= (xzz)[x:=y] = yzz \\
((xyz)[y:=z])[x:=y[y:=z]] &= (xzz)[x:=z] = zzz
\end{aligned}
$$

$\neq$ $=$

Lemma 1.21 (sequential substitution):

substitution is left associative

If $x \notin FV(C)$, then $A[x:=B][y:=C] \equiv A[y:=C][x:=B[y:=C]]$

Type Theory, 8

Mittwoch, 19. Oktober 2022 17:30

Beta reduction: the actual computation

Definition 1.22: one step β-reduction, $\to_\beta$, redex, contractum

(1) (basis) $(\lambda x.A)B \to_\beta A[x:=B]$

(2) (compatibility) If $A \to_\beta B$, then $AC \to_\beta BC$, $CA \to_\beta CB$, $\lambda x.A \to_\beta \lambda x.B$.

A subterm of the form $(\lambda x.A)B$ is called redex ("reducible expression") and $A[x:=B]$ its contractum.

Example 1.23: (1) $(\lambda x.(\lambda y.yx)z)u \to_\beta (\lambda y.yu)z \to_\beta zu$

$(\lambda x.(\lambda y.yx)z)u \to_\beta (\lambda x.zx)u \to_\beta zu$

$(\lambda y.yu)z \neq_\alpha (\lambda x.zx)u$, $zu =_\alpha zu$

temporarily ignore Barendregt convention

(2) $(\lambda x.xx)(\lambda x.xx) \to_\beta (\lambda x.xx)(\lambda x.xx)$

Definition 1.24: β-reduction, $\twoheadrightarrow_\beta$, β-conversion, $=_\beta$

$A \twoheadrightarrow_\beta B$ if $\exists n \in \mathbb{N}_0 : \exists A_0, \dots, A_n \in \Lambda : A_0 \equiv A,\ A_n \equiv B,\ \forall i \in \{0, \dots, n-1\} : A_i \to_\beta A_{i+1}$.

$A =_\beta B$ if $\exists n \in \mathbb{N}_0 : \exists A_0, \dots, A_n \in \Lambda : A_0 \equiv A,\ A_n \equiv B,\ \forall i \in \{0, \dots, n-1\} : A_i \to_\beta A_{i+1} \vee A_{i+1} \to_\beta A_i$.

If $A =_\beta B$, then A and B are said to be "beta convertibles" or "beta equivalent".

Example 1.25: $(\lambda x.(\lambda y.yx)z)u =_\beta (\lambda y.yu)z =_\beta (\lambda x.zx)u =_\beta zu$

but $(\lambda y.yu)z \underset{\beta \not\twoheadleftarrow}{\not\twoheadrightarrow_\beta} (\lambda x.zx)u$

Lemma 1.26: (1) $\twoheadrightarrow_\beta$ extends $\to_\beta$: $A \to_\beta B \Rightarrow A \twoheadrightarrow_\beta B$

$=_\beta$ extends $\twoheadrightarrow_\beta$ in both directions: $A \twoheadrightarrow_\beta B \vee B \twoheadrightarrow_\beta A \Rightarrow A =_\beta B$

(2) $\twoheadrightarrow_\beta$ is reflexive and transitive

$=_\beta$ is an equivalence relation

Remark 1.27: compare: $(3+7)\cdot(8-2) \to 10\cdot(8-2) \to 10\cdot 6 \to 60$

$(3+7)\cdot(8-2) \to (3+7)\cdot 6 \to 10\cdot 6 \to 60$

motivation for inverse β-reduction

$ax^2+bx+c \leftarrow a(x^2+\frac{b}{a}\cdot x)+c \leftarrow a(x^2+2\frac{b}{2a}\cdot x)+c$

$\leftarrow a\cdot(x+\frac{b}{2a})^2 - \frac{b^2}{4a} + c$

$\Rightarrow c - \frac{b^2}{4a}$ is extreme value for $x = -\frac{b}{2a}$

Normal forms and confluence

Definition 1.28: β-normal form, β-normalizing

A is in β-normal form (short: β-nf) if A contains no redexes.
A has a β-normal form / is β-normalizing if $\exists B$ in β-nf: $A =_\beta B$.
In that case B is called a β-normal form of A.

Lemma 1.29: if A is in β-nf and $A \twoheadrightarrow_\beta B$, then $A \equiv B$.

Proof: follows directly from the definitions.

Example 1.30: (1) $(\lambda x.(\lambda y.yx)z)u \twoheadrightarrow_\beta zu$, so
zu is a β-nf of $(\lambda x.(\lambda y.yx)z)u$.

(2) If $\Omega := (\lambda x.xx)(\lambda x.xx)$, then $\Omega \rightarrow_\beta \Omega$ and
Ω has no β-nf.

(3) If $\Delta := \lambda x.xxx$, then $\Delta\Delta \rightarrow_\beta \Delta\Delta\Delta \rightarrow_\beta \Delta\Delta\Delta\Delta \rightarrow_\beta \ldots$ and
$\Delta\Delta$ has no β-nf.

(4) $\left.\begin{array}{l}(\lambda u.v)\Omega \rightarrow_\beta (\lambda u.v)\Omega \\ (\lambda u.v)\Omega \rightarrow_\beta v\end{array}\right\}$, so
$(\lambda u.v)\Omega$ has a β-nf, but it may not be reached
if the wrong redex is chosen repeatedly.

Remark 1.31: By (2), the converse of Lemma 1.28 is not true,
since $\Omega \twoheadrightarrow_\beta \Omega$ and $\Omega \equiv \Omega$ but Ω is not in β-nf.

Definition 1.32: reduction path

A finite reduction path from A is a sequence $A_0, \ldots, A_n$ such that
$A \equiv A_0$ and $\forall i \in \{0, \ldots, n-1\}: A_i \rightarrow_\beta A_{i+1}$
An infinite reduction path from A is a sequence $A_0, A_1, \ldots$ such that
$A \equiv A_0$ and $\forall i \in \mathbb{N}_0: A_i \rightarrow_\beta A_{i+1}$

Definition 1.33: weak normalization, strong normalization

A is weakly normalizing if $\exists B$ in β-nf: $A \twoheadrightarrow_\beta B$.

A is strongly normalizing if $\nexists$ infinite reduction paths from A.

Example 1.34: (1) $(\lambda u.v)\Omega$ is weakly normalizing

(2) $(\lambda x.(\lambda y.yx)z)u$ is strongly normalizing

(3) Ω and $\Delta\Delta$ are not weakly normalizing

Theorem 1.35 (Church-Rosser, confluence, diamond property):

DP

Let $A, B_1, B_2 \in \Lambda$ with $A \twoheadrightarrow_\beta B_1$, $A \twoheadrightarrow_\beta B_2$.

Then, there is a $C \in \Lambda$ such that $B_1 \twoheadrightarrow_\beta C$, $B_2 \twoheadrightarrow_\beta C$.

$$\begin{tikzcd}
 & A \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
B_1 \arrow[dr, "\beta"'] & & B_2 \arrow[dl, "\beta"] \\
 & C &
\end{tikzcd}$$

Proof:

We have 3 binary relations on Λ:

$\rightarrow_\beta$: (1) $(\lambda x.A)B \rightarrow_\beta A[x:=B]$

reflexive closure of $\rightarrow_\beta$

(2) If $A \rightarrow_\beta B$, then $AC \rightarrow_\beta BC$, $CA \rightarrow_\beta CB$, $\lambda x.A \rightarrow_\beta \lambda x.B$

$\overrightarrow{\equiv}_\beta$: (1) If $A \rightarrow_\beta B$, then $A \overrightarrow{\equiv}_\beta B$

transitive closure of $\overrightarrow{\equiv}_\beta$

(2) $A \overrightarrow{\equiv}_\beta A$

$\twoheadrightarrow_\beta$: (1) If $A \overrightarrow{\equiv}_\beta B$, then $A \twoheadrightarrow_\beta B$

(2) If $A \twoheadrightarrow_\beta B$, $B \twoheadrightarrow_\beta C$, then $A \twoheadrightarrow_\beta C$

Claim 1: if a binary relation satisfies DP, so does its transitive closure.

Let $\rightarrow$ be any binary relation satisfying DP, $\twoheadrightarrow$ its transitive closure, and A, B_1, B_2 such that $A \twoheadrightarrow B_1$, $A \twoheadrightarrow B_2$.

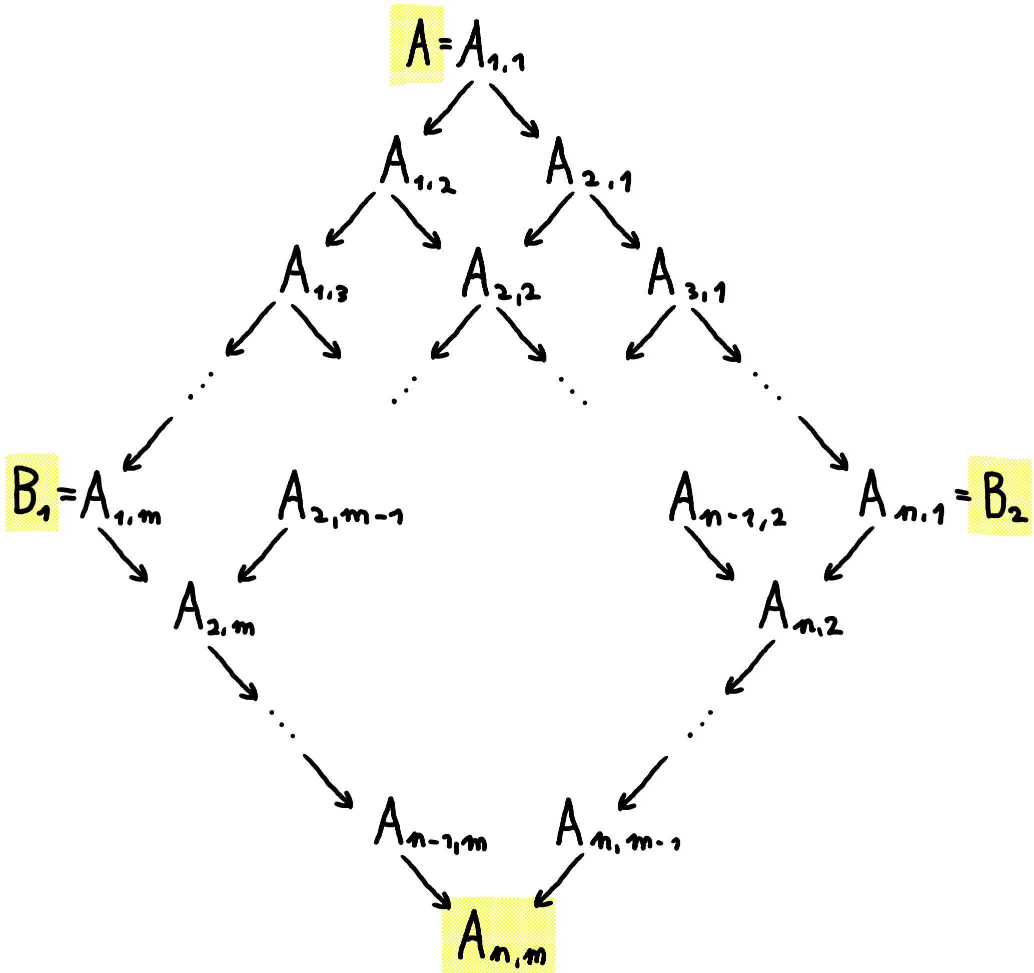
Then $A = A_{1,1} \rightarrow A_{1,2} \rightarrow \ldots \rightarrow A_{1,m} = B_1$.

$A = A_{1,1} \rightarrow A_{2,1} \rightarrow \ldots \rightarrow A_{n,1} = B_2$

Type Theory, 11

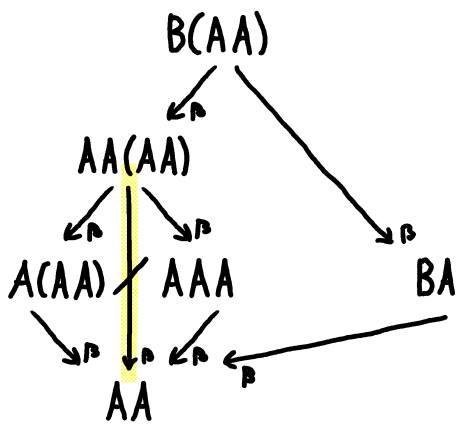
Mittwoch, 26. Oktober 2022 15:54

For $i \in \{2, \dots, n\}$, $j \in \{2, \dots, m\}$ let $A_{i,j}$ such that $A_{i-1,j} \to A_{i,j}$, $A_{i,j-1} \to A_{i,j}$ (by DP of $\to$)



Then, $B_1 \twoheadrightarrow A_{n,m}$, $B_2 \twoheadrightarrow A_{n,m}$. ✓

Unfortunately $\to_\beta$ doesn't satisfy DP: if $A \equiv \lambda x.x$, $B \equiv \lambda x.xx$, then



Type Theory, 12

Mittwoch, 26. Oktober 2022 16:49

"s" for simultaneous

as relations in set theoretic sense (i.e. subsets of Λ^2)

Solution: find new binary relation $\to_s$ on Λ with $\to_\beta \subset \twoheadrightarrow_{\equiv\beta} \subset \to_s \subset \twoheadrightarrow_\beta$

such that $\to_s$ satisfies DP (then so does $\twoheadrightarrow_\beta$).

by Claim 1

$\to_s$: (1) $A \to_s A$

(2) If $A \to_s A'$, $B \to_s B'$, then $AB \to_s A'B'$

(3) If $A \to_s A'$, then $\lambda x.A \to_s \lambda x.A'$

simultaneous reduction possible, cf. problem with $\twoheadrightarrow_{\equiv\beta}$

(4) If $A \to_s A'$, $B \to_s B'$, then $(\lambda x.A)B \to_s A'[x:=B']$

Claim 2: if $A \to_s A'$, $B \to_s B'$, then $A[x:=B] \to_s A'[x:=B']$.

Proof by induction on $A \to_s A'$:

Case 1: $A \to_s A'$ because $A \equiv A'$

Show $A[x:=B] \to_s A[x:=B']$ by induction on A

Case 1.1: $A \equiv x$ or $A \equiv y$

$$x[x:=B] = B \to_s B' = x[x:=B']$$

$$y[x:=B] = y \to_s y = y[x:=B'] \checkmark$$

Case 1.2: $A \equiv PQ$

$$\begin{aligned} PQ[x:=B] &= (P[x:=B])(Q[x:=B]) \\ \overset{(IH)}{\to_s} &\; (P[x:=B'])(Q[x:=B']) \\ &= PQ[x:=B'] \checkmark \end{aligned}$$

Case 1.3: $A \equiv \lambda x.P$ or $A \equiv \lambda y.P$

$$(\lambda x.P)[x:=B] = \lambda x.P \to_s \lambda x.P = (\lambda x.P)[x:=B']$$

$$\begin{aligned} (\lambda y.P)[x:=B] &= \lambda z.(P^{y\mapsto z}[x:=B]) \quad (z \notin FV(B)) \\ \overset{(IH)}{\to_s} &\; \lambda z.(P^{y\mapsto z}[x:=B']) \quad (z \notin FV(B')) \\ &= (\lambda y.P)[x:=B'] \checkmark \end{aligned}$$

Case 2: $A \to_s A'$ because $A \equiv PQ$, $A' \equiv P'Q'$ with $P \to_s P'$, $Q \to_s Q'$

$$(PQ)[x:=B] = (P[x:=B])(Q[x:=B])$$

$$\overset{(IH)}{\to_s} (P'[x:=B'])(Q'[x:=B']) = (P'Q')[x:=B'] \checkmark$$

Case 3: $A \to_s A'$ because $A \equiv \lambda x.P$, $A' \equiv \lambda x.P'$ or $A \equiv \lambda y.P$, $A' \equiv \lambda y.P'$ with $P \to_s P'$

$$(\lambda x.P)[x:=B] = \lambda x.P \to_s \lambda x.P' = (\lambda x.P')[x:=B']$$

$$(\lambda y.P)[x:=B] = \lambda z.(P^{y\to z}[x:=B]) \quad (z \notin FV(B))$$

$$\overset{(IH)}{\to_s} \lambda z.(P'^{y\to z}[x:=B']) \quad (z \notin FV(B'))$$

$$= (\lambda y.P')[x:=B'] \checkmark$$

Case 4: $A \to_s A'$ because $A \equiv (\lambda x.P)Q$, $A' \equiv P'[x:=Q']$ or
$A \equiv (\lambda y.P)Q$, $A' \equiv P'[y:=Q']$ with $P \to_s P'$, $Q \to_s Q'$

$$\begin{aligned}
((\lambda x.P)Q)[x:=B] &= ((\lambda x.P)[x:=B])(Q[x:=B]) \\
&= (\lambda x.P)(Q[x:=B]) \\
\overset{(IH)}{\to_s}\ & P'[x:=Q'[x:=B']] \\
&= (P'^{x\to z}[x:=B'])[z:=Q'[x:=B']] \\
\text{Lemma 1.21} \searrow\ &= (P'^{x\to z}[z:=Q'])[x:=B'] \quad (z \notin FV(B')) \\
&= (P'[x:=Q'])[x:=B']
\end{aligned}$$

$$\begin{aligned}
((\lambda y.P)Q)[x:=B] &= ((\lambda y.P)[x:=B])(Q[x:=B]) \\
&= (\lambda z.(P^{y\to z}[x:=B]))(Q[x:=B]) \quad (z \notin FV(B)) \\
\overset{(IH)}{\to_s}\ & (P'^{y\to z}[x:=B'])[z:=Q'[x:=B']] \\
\text{Lemma 1.21} \nearrow\ &= (P'^{y\to z}[z:=Q'])[x:=B'] \quad (z \notin FV(B')) \\
&= (P'[y:=Q'])[x:=B'] \checkmark
\end{aligned}$$

Claim 3: (1) If $\lambda x.A \to_s B$, then $B \equiv \lambda x.A'$ with $A \to_s A'$.

(2) If $AB \to_s C$, then either $C \equiv A'B'$ with $A \to_s A'$, $B \to_s B'$
or $A \equiv \lambda x.P$, $C \equiv P'[x:=B']$ with $P \to_s P'$, $B \to_s B'$.

Proof by easy induction on definition of $\to_s$. $\checkmark$

Type Theory, 14

Mittwoch, 26. Oktober 2022 20:14

Claim 4: $\to_s$ satisfies DP.

Show $\forall B_2: A \to_s B_2 \Rightarrow \exists C: B_1 \to_s C, B_2 \to_s C$ by induction on $A \to_s B_1$.

Case 1: $A \to_s B_1$ because $A \equiv B_1$

$C \equiv B_2 \Rightarrow B_1 \equiv A \to_s B_2 \equiv C$, $B_2 \equiv C \to_s C$ ✓

Case 2: $A \to_s B_1$ because $A \equiv PQ$, $B_1 \equiv P'Q'$ with $P \to_s P'$, $Q \to_s Q'$

Case 2.1: $B_2 \equiv P''Q''$ with $P \to_s P''$, $Q \to_s Q''$

according to Claim 3 (2)

$C \equiv P'''Q'''$ with $P''', Q''' \in \Lambda$ such that

$P' \to_s P'''$, $P'' \to_s P'''$, $Q' \to_s Q'''$, $Q'' \to_s Q'''$ (IH)

$\Rightarrow B_1 \equiv P'Q' \to_s P'''Q''' \equiv C$, $B_2 \equiv P''Q'' \to_s P'''Q''' \equiv C$ ✓

Case 2.2: $P \equiv \lambda x.R$, $B_2 \equiv R''[x:=Q'']$ with $R \to_s R''$, $Q \to_s Q''$

Claim 3 (1) $\to$ $\lambda x.R \to_s P' \Rightarrow P' \equiv \lambda x.R'$ with $R \to_s R'$

$C \equiv R'''[x:=Q''']$ with $R''', Q''' \in \Lambda$ such that

$R' \to_s R'''$, $R'' \to_s R'''$, $Q' \to_s Q'''$, $Q'' \to_s Q'''$ (IH)

$\Rightarrow B_1 \equiv P'Q' \equiv (\lambda x.R')Q' \to_s R'''[x:=Q'''] \equiv C$,

Claim 2 $\to$ $B_2 \equiv R''[x:=Q''] \to_s R'''[x:=Q'''] \equiv C$ ✓

Case 3: $A \to_s B_1$ because $A \equiv \lambda x.P$, $B_1 \equiv \lambda x.P'$ with $P \to_s P'$

Claim 3 (1) $\to$ $\lambda x.P \to_s B_2 \Rightarrow B_2 \equiv \lambda x.P''$ with $P \to_s P''$

$C \equiv \lambda x.P'''$ with $P''' \in \Lambda$ such that $P' \to_s P'''$, $P'' \to_s P'''$ (IH)

$\Rightarrow B_1 \equiv \lambda x.P' \to_s \lambda x.P''' \equiv C$, $B_2 \equiv \lambda x.P'' \to_s \lambda x.P''' \equiv C$ ✓

Case 4: $A \to_s B_1$ because $A \equiv (\lambda x.P)Q$, $B_1 \equiv P'[x:=Q']$ with $P \to_s P'$, $Q \to_s Q'$

Case 4.1: $B_2 \equiv (\lambda x.P'')Q''$ with $P \to_s P''$, $Q \to_s Q''$

according to Claim 3 (2)

$C \equiv P'''[x:=Q''']$ with $P''', Q''' \in \Lambda$ such that

$P' \to_s P'''$, $P'' \to_s P'''$, $Q' \to_s Q'''$, $Q'' \to_s Q'''$ (IH)

Claim 2 $\to$ $\Rightarrow B_1 \equiv P'[x:=Q'] \to_s P'''[x:=Q'''] \equiv C$,

$B_2 \equiv (\lambda x.P'')Q'' \to_s P'''[x:=Q'''] \equiv C$ ✓

Type Theory, 15

Freitag, 28. Oktober 2022 14:31

Case 4.2: $B_2 \equiv P''[x := Q'']$ with $P \to_s P''$, $Q \to_s Q''$

$\uparrow$ according to Claim 3(2)

$C \equiv P'''[x := Q''']$ with $P''', Q''' \in \Lambda$ such that

$P' \to_s P'''$, $P'' \to_s P'''$, $Q' \to_s Q'''$, $Q'' \to_s Q'''$ (IH)

Claim 2 $\to$ $\Rightarrow B_1 \equiv P'[x := Q'] \to_s P'''[x := Q'''] \equiv C$,

Claim 2 $\to$ $B_2 \equiv P''[x := Q''] \to_s P'''[x := Q'''] \equiv C$ ✓

Claim 1, Claim 4, $\twoheadrightarrow_\beta$ is transitive closure of $\to_s$ $\Rightarrow$ $\twoheadrightarrow_\beta$ satisfies DP ✓

Example 1.36: compare

$$\begin{tikzcd}[column sep=tiny]
& (\lambda xy.x)(\lambda z.zz)b((\lambda uv.v)cd) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
(\lambda z.zz)((\lambda uv.v)cd) \arrow[dr, "\beta"'] & & (\lambda xy.x)(\lambda z.zz)bd \arrow[dl, "\beta"] \\
& (\lambda z.zz)d \arrow[d, "\beta"] & \\
& dd &
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=tiny]
& (3+7)\cdot(8-2) \arrow[dl] \arrow[dr] & \\
10\cdot(8-2) \arrow[dr] & & (3+7)\cdot 6 \arrow[dl] \\
& 10\cdot 6 \arrow[d] & \\
& 60 &
\end{tikzcd}$$

Exercises 3

3.1: Let $A \equiv (\lambda z.zxz)((\lambda y.xy)x)$. Mark all redexes in A, find all reduction paths and the β-nf of A.

$(\lambda z.zxz)((\lambda y.xy)x)$, $(\lambda z.zxz)((\lambda y.xy)x)$

$$\begin{tikzcd}[column sep=tiny]
& & (\lambda z.zxz)((\lambda y.xy)x) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
& ((\lambda y.xy)x)x((\lambda y.xy)x) \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & & (\lambda z.zxz)(xx) \arrow[dd, "\beta"] \\
(xx)x((\lambda y.xy)x) \arrow[drrr, "\beta"'] & & ((\lambda y.xy)x)x(xx) \arrow[dr, "\beta"] & \\
& & & (xx)x(xx)
\end{tikzcd}$$

Type Theory, 16

Freitag, 28. Oktober 2022 20:56

3.2: Let zero $\equiv \lambda fx.x$, one $\equiv \lambda fx.fx$, two $\equiv \lambda fx.f(fx)$,

add $\equiv \lambda mnfx.mf(nfx)$, mult $\equiv \lambda mnfx.m(nf)x$,

suc $\equiv \lambda mfx.f(mfx)$.

Show (1) add one one $\twoheadrightarrow_\beta$ two

(2) add one one $\not\equiv_\beta$ mult one zero

(3) suc zero $\equiv_\beta$ one

(4) suc one $\equiv_\beta$ two

$$
\begin{aligned}
\text{add one one} &\equiv (\lambda mnfx.mf(nfx))(\lambda fx.fx)(\lambda fx.fx) \\
&\twoheadrightarrow_\beta \lambda fx.(\lambda fx.fx)f((\lambda fx.fx)fx) \\
&\twoheadrightarrow_\beta \lambda fx.(\lambda x.fx)(fx) \twoheadrightarrow_\beta \lambda fx.f(fx) \equiv \text{two}
\end{aligned}
$$

$$
\begin{aligned}
\text{mult one zero} &\equiv (\lambda mnfx.m(nf)x)(\lambda fx.fx)(\lambda fx.x) \\
&\twoheadrightarrow_\beta \lambda fx.(\lambda fx.fx)((\lambda fx.x)f)x \\
&\twoheadrightarrow_\beta \lambda fx.(\lambda fx.fx)(\lambda x.x)x \twoheadrightarrow_\beta \lambda fx.(\lambda x.x)x \\
&\twoheadrightarrow_\beta \lambda fx.x \equiv \text{zero} \not\equiv_\beta \text{two} \equiv_\beta \text{add one one}
\end{aligned}
$$

$$
\begin{aligned}
\text{suc zero} &\equiv (\lambda mfx.f(mfx))(\lambda fx.x) \twoheadrightarrow_\beta \lambda fx.f((\lambda fx.x)fx) \\
&\twoheadrightarrow_\beta \lambda fx.fx \equiv \text{one}
\end{aligned}
$$

$$
\begin{aligned}
\text{suc one} &\equiv (\lambda mfx.f(mfx))(\lambda fx.fx) \twoheadrightarrow_\beta \lambda fx.f((\lambda fx.fx)fx) \\
&\twoheadrightarrow_\beta \lambda fx.f(fx) \equiv \text{two}
\end{aligned}
$$

Type Theory, 17

Montag, 31. Oktober 2022 10:53

3.3: Let $\text{true} \equiv \lambda xy.x$, $\text{false} \equiv \lambda xy.y$,
$\text{not} \equiv \lambda z.z\,\text{false}\,\text{true}$, $\text{iszero} \equiv \lambda z.z(\lambda x.\text{false})\text{true}$,
$\text{ifte} \equiv \lambda xuv.xuv$.

Show (1) $\text{not}(\text{not}\,P) =_\beta P$ if $P \in \Lambda$ with $P \twoheadrightarrow_\beta \text{true}$
(2) $\text{iszero zero} \twoheadrightarrow_\beta \text{true}$, $\text{iszero two} \twoheadrightarrow_\beta \text{false}$
(3) $\text{ifte true}\,a\,b \twoheadrightarrow_\beta a$, $\text{ifte false}\,a\,b \twoheadrightarrow_\beta b$

$$\begin{aligned}
\text{not}(\text{not}\,P) &\twoheadrightarrow_\beta \text{not}(\text{not true}) \\
&\equiv (\lambda z.z\,\text{false true})((\lambda z.z\,\text{false true})\text{true}) \\
&\twoheadrightarrow_\beta (\lambda z.z\,\text{false true})(\text{true false true}) \\
&\twoheadrightarrow_\beta \text{true false true false true} \\
&\twoheadrightarrow_\beta \text{false false true} \twoheadrightarrow_\beta \text{true}
\end{aligned}$$

$$\begin{aligned}
\text{iszero zero} &\equiv (\lambda z.z(\lambda x.\text{false})\text{true})(\lambda fx.x) \\
&\twoheadrightarrow_\beta (\lambda fx.x)(\lambda x.\text{false})\text{true} \twoheadrightarrow_\beta \text{true}
\end{aligned}$$

$$\begin{aligned}
\text{iszero two} &\equiv (\lambda z.z(\lambda x.\text{false})\text{true})(\lambda fx.f(fx)) \\
&\twoheadrightarrow_\beta (\lambda fx.f(fx))(\lambda x.\text{false})\text{true} \twoheadrightarrow_\beta \\
&\twoheadrightarrow_\beta (\lambda x.\text{false})((\lambda x.\text{false})\text{true}) \twoheadrightarrow_\beta \text{false}
\end{aligned}$$

$$\text{ifte true}\,a\,b \equiv (\lambda xuv.xuv)(\lambda xy.x)ab \twoheadrightarrow_\beta (\lambda xy.x)ab \twoheadrightarrow_\beta a$$

$$\text{ifte false}\,a\,b \equiv (\lambda xuv.xuv)(\lambda xy.y)ab \twoheadrightarrow_\beta (\lambda xy.y)ab \twoheadrightarrow_\beta b$$

Type Theory, 18

Montag, 31. Oktober 2022 12:00

Corollary 1.37: If $A =_\beta B$, then there is a $C \in \Lambda$ such that
$A \twoheadrightarrow_\beta C$, $B \twoheadrightarrow_\beta C$.

Proof:

i.e. $A_0 \to_\beta A_1 \vee A_1 \to_\beta A_0$

Let $n \in \mathbb{N}_0$, $A_0, \dots, A_n \in \Lambda$ such that $A \equiv A_0 \, {}_\beta\!\overset{\to}{\underset{\leftarrow}{\vee}}{}^\beta \, A_1 \, {}_\beta\!\overset{\to}{\underset{\leftarrow}{\vee}}{}^\beta \, A_2 \, {}_\beta\!\overset{\to}{\underset{\leftarrow}{\vee}}{}^\beta \dots {}_\beta\!\overset{\to}{\underset{\leftarrow}{\vee}}{}^\beta \, A_{n-1} \, {}_\beta\!\overset{\to}{\underset{\leftarrow}{\vee}}{}^\beta \, A_n \equiv B$

Show $\forall i \in \{0, \dots, n\} : \exists C_i \in \Lambda : A_0 \twoheadrightarrow_\beta C_i, \ A_i \twoheadrightarrow_\beta C_i$ by induction on i.

$i = 0$: Let $C_0 \equiv A_0$.

Then: $A_0 \twoheadrightarrow_\beta C_i$, $A_i \twoheadrightarrow_\beta C_i$ ✓

$i-1 \to i > 0$: Let $C_{i-1} \in \Lambda$ such that $A_0 \twoheadrightarrow_\beta C_{i-1}$, $A_{i-1} \twoheadrightarrow_\beta C_{i-1}$ (IH)

Case 1: $A_{i-1} \to_\beta A_i$

Theorem 1.35 → Let $C_i \in \Lambda$ such that $C_{i-1} \twoheadrightarrow_\beta C_i$, $A_i \twoheadrightarrow_\beta C_i$.

Then:

$$\begin{tikzcd}
A_0 \arrow[dr, two heads, "\beta"] & & A_{i-1} \arrow[r, "\beta"] \arrow[dl, two heads, "\beta"'] & A_i \arrow[dl, two heads, "\beta"] \\
& C_{i-1} \arrow[r, two heads, "\beta"] \arrow[ur, phantom, "\mathrm{DP}"] & C_i &
\end{tikzcd}$$

i.e. $A_0 \twoheadrightarrow_\beta C_i$, $A_i \twoheadrightarrow_\beta C_i$ ✓

Case 2: $A_i \to_\beta A_{i-1}$

Let $C_i \equiv C_{i-1}$.

Then:

$$\begin{tikzcd}
A_0 \arrow[dr, two heads, "\beta"] & & A_{i-1} \arrow[dl, two heads, "\beta"'] & A_i \arrow[l, "\beta"'] \\
& C_{i-1} \equiv C_i & &
\end{tikzcd}$$

i.e. $A_0 \twoheadrightarrow_\beta C_i$, $A_i \twoheadrightarrow_\beta C_i$ ✓

the β-nf can always be reached by "forward calculation"

Corollary 1.38: (1) If B is a β-nf of A, then $A \twoheadrightarrow_\beta B$.

the β-nf is unique

(2) A λ-term has at most one β-nf.

Proof: follows directly from Theorem 1.35, Corollary 1.37, and Lemma 1.29.

Type Theory, 19

Montag, 31. Oktober 2022 16:30

Fixed point theorem: Turing completeness of the λ-calculus

Theorem 1.39: Let $A \in \Lambda$. Then there is a $B \in \Lambda$ such that $AB =_\beta B$.

Proof: Let $B \equiv (\lambda x. A(xx))(\lambda x. A(xx))$.

Then $B \to_\beta A((\lambda x. A(xx))(\lambda x. A(xx))) \equiv AB$, so $AB =_\beta B$. ✓

Definition 1.40: Y-combinator

The λ-term $Y \equiv \lambda y.(\lambda x. y(xx))(\lambda x. y(xx))$ is called Y-combinator.

Remark 1.41: If $A \in \Lambda$, then YA is a fixed point of A: $A(YA) =_\beta YA$

Theorem 1.39

Consequence: all recursive equations are solvable!

$X =_\beta \ldots X \ldots$

Define $A \equiv \lambda f. \ldots f \ldots$

Then $AX \to_\beta \ldots X \ldots$

Hence, to solve $X =_\beta \ldots X \ldots$ it suffices to find $X \in \Lambda$ with $AX =_\beta X$, but such an X exists: $X \equiv YA$.

Example 1.42:

(1) Solve $Xy =_\beta yXy$

$Xy =_\beta yXy$ if $X =_\beta \lambda y. yXy$

$A \equiv \lambda f y. yfy$

$X \equiv YA \equiv (\lambda y.(\lambda x. y(xx))(\lambda x. y(xx)))(\lambda f y. yfy)$

$$
\begin{aligned}
Xy &\to_\beta (\lambda x.(\lambda f y. yfy)(xx))(\lambda x.(\lambda f y. yfy)(xx))y \\
&\twoheadrightarrow_\beta (\lambda xy. y(xx)y)(\lambda xy. y(xx)y)y \\
&\twoheadrightarrow_\beta y((\lambda xy. y(xx)y)(\lambda xy. y(xx)y))y \\
&{}_\beta\twoheadleftarrow y(\lambda x.(\lambda f y. yfy)(xx))(\lambda x.(\lambda f y. yfy)(xx))y \\
&{}_\beta\leftarrow yXy
\end{aligned}
$$

Type Theory, 20

Dienstag, 1. November 2022 17:30

(2) $\text{fac}\ n =_\beta \text{ifte}\ (\text{iszero}\ n)\ \text{one}\ (\text{mult}\ n\ (\text{fac}\ (\text{pred}\ n)))$

where $\text{pred} \equiv \lambda mfx.m(\lambda gh.h(gf))(\lambda u.x)(\lambda u.u)$

$\text{fac} =_\beta \lambda n.\text{ifte}\ (\text{iszero}\ n)\ \text{one}\ (\text{mult}\ n\ (\text{fac}\ (\text{pred}\ n)))$

$A \equiv \lambda fn.\text{ifte}\ (\text{iszero}\ n)\ \text{one}\ (\text{mult}\ n\ (f\ (\text{pred}\ n)))$

$\text{fac} \equiv YA \equiv (\lambda y.(\lambda x.y(xx))(\lambda x.y(xx)))(\lambda fn.$
$(\lambda xuv.xuv)((\lambda z.z(\lambda xxy.y)(\lambda xy.x))n)(\lambda fx.fx)$
$((\lambda mnfx.m(nf)x)n(f((\lambda mfx.m(\lambda gh.h(gf))$
$(\lambda u.x)(\lambda u.u))n))))$

$\text{fac}\ (\lambda fx.f(f(fx))) \twoheadrightarrow_\beta \lambda fx.f(f(f(f(f(fx)))))$

reducing "leftmost" redex 704 times

(3) Let div, mod, eq as defined below and

$\text{collatz} \equiv \lambda n.\text{ifte}(\text{iszero}(\text{mod}\ n\ \text{two}))(\text{div}\ n\ \text{two})(\text{suc}(\text{mult}\ n\ \text{three}))$

$\text{isperiodic} \equiv \lambda nk.\text{eq}\ n\ (\underbrace{k\ \text{collatz}\ n}_{\text{k-fold application of collatz to } n})$

$\text{findperiod} \equiv Y(\lambda fnk.\text{ifte}(\text{isperiodic}\ n\ k)\ n\ (\text{ifte}\ (\text{iszero}(\text{pred}\ k))$
$(f\ \text{three}\ (\text{pred}\ n))(f\ (\text{suc}\ n)(\text{pred}\ k))))$.

Then, collatz three $=_\beta$ ten, collatz ten $=_\beta$ five, ...

isperiodic one three $=_\beta$ isperiodic two three $=_\beta$ true

isperiodic three k $=_\beta$ isperiodic four k $=_\beta$ false

for all natural numbers k

findperiod n three $=_\beta$ n for $n \equiv$ one or $n \equiv$ two

findperiod three three has a β-nf iff a counterexample to the 3n+1 problem exists!

Type Theory, 21

Donnerstag, 3. November 2022 17:00

Theorem 1.43: The λ-calculus is Turing complete.

Proof:

We define

$Y \equiv \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$

$true \equiv \lambda xy.x$

$false \equiv \lambda xy.y$

$not \equiv \lambda z.z\ false\ true$

$and \equiv \lambda xy.xyx$

$or \equiv \lambda xy.xxy$

$ifte \equiv \lambda xuv.xuv$

$zero \equiv \lambda fx.x$

$suc \equiv \lambda mfx.f(mfx)$

$pred \equiv \lambda mfx.m(\lambda gh.h(gf))(\lambda u.x)(\lambda u.u)$

$add \equiv \lambda mnfx.mf(nfx)$

$subt \equiv \lambda mn.n\ pred\ m$

$mult \equiv \lambda mnfx.m(nf)x$

$iszero \equiv \lambda z.z(\lambda x.false)true$

$leq \equiv \lambda mn.iszero(subt\ m\ n)$

$eq \equiv \lambda mn.and\ (leq\ m\ n)(leq\ n\ m)$

$div \equiv Y(\lambda f\,mn.ifte(leq\ n\ m)(add\ (suc\ zero)(f(subt\ m\ n)n))\,zero)$

$mod \equiv Y(\lambda f\,mn.ifte(leq\ n\ m)(f(subt\ m\ n)n)m)$

$pair \equiv \lambda fsx.xfs$

$fst \equiv \lambda x.x\ true$

$sec \equiv \lambda x.x\ false$

Type Theory, 22

Freitag, 4. November 2022 15:30

Consider the following Turing machine (demonstrated construction works for every TM):

1	_	_	r	halt
1	1	_	r	2
2	_	_	l	3
2	1	1	r	2
3	_	1	r	halt
3	1	_	l	4
4	_	_	r	1
4	1	1	l	4

Initial input: _ or 1 or 11 or ...
Initial state: 1
Output: _ if even number of "1"s initially
1 if odd number of "1"s initially

Try at: https://morphett.info/turing.html

total state including tape

We represent the current state of the TM by a pair of pairs

((state, symbol at head), (tape left of head, tape right of head)),

_, 1 ↔ zero, one

halt, 1, 2, 3, 4 ↔ zero, one, two, three, four

finite lists implemented as concatenated pairs, number of elements - 2 in first entry

e.g.: $\boxed{4}$ ↔ ((four, one), ((zero, (zero, one)), (one, (zero, (one, one)))))

1_1_11

Goal: write program in λ-calculus which updates state like TM would.

We define the following helper functions:

state ≡ λs.fst(fst s) symb ≡ λs.sec(fst s)

tapel ≡ λs.fst(sec s) taper ≡ λs.sec(sec s)

Lenl ≡ λs.fst(tapel s) Lenr ≡ λs.fst(taper s)

headl ≡ λs.fst(sec(tapel s)) headr ≡ λs.fst(sec(taper s))

append ≡ λla.pair (suc(fst l))(pair a (sec l))

drop ≡ λl.ifte(iszero(fst l))
(pair zero (pair (sec(sec l)) zero))
(pair (pred(fst l)) (sec(sec l)))

Type Theory, 23

Freitag, 4. November 2022 20:30

Let updatel ≡ λsabcde. ifte (and (eq (state s) a)(eq (symb s) b))
(pair (pair c (headl s))(pair (drop (tapel s))(append (taper s) d))) e

updater ≡ λsabcde. ifte (and (eq (state s) a)(eq (symb s) b))
(pair (pair c (headr s))(pair (append (tapel s) d)(drop (taper s)))) e

update ≡ λs. ifte (iszero (state s)) s

```
(updater s one   zero zero  zero
(updater s one   one  two   zero
(updatel s two   zero three zero
(updater s two   one  two   one
(updater s three zero zero  one
(updatel s three one  four  zero
(updater s four  zero one   zero
(updatel s four  one  four  one
false ))))))))
```

TM dependent part

updateloop ≡ Y (λfs. ifte (iszero (state s)) s (f (update s)))

Then updateloop (pair (pair one one)(pair (pair zero (pair zero zero))
(pair zero (pair one one))))
has β-nf pair (pair zero zero)(pair (pair two (pair one (pair zero
(pair zero zero))))(pair one (pair zero (pair zero zero))))

and updateloop (pair (pair one one)(pair (pair zero (pair zero zero))
(pair one (pair one (pair one one)))))
has β-nf pair (pair zero zero)(pair (pair three (pair zero (pair zero
(pair zero (pair zero zero)))))(pair one (pair zero (pair zero zero)))).

In general, updateloop mimics the behavior of the corresponding TM and its β-nf exists iff the TM holds and in that case is equal to the final state of the TM.

Type Theory, 24

Freitag, 4. November 2022 23:00

Theorem 1.44: Repeated "leftmost" β-reduction finds the β-nf if it exists.

Remark 1.45: summary and conclusions

- ·) $\Lambda = V \mid (\Lambda\Lambda) \mid (\lambda V.\Lambda)$
- ·) names of binding variables are irrelevant $\rightarrow$ $\equiv$ "=" $=_c$
- ·) computation = β-reduction = substitution
- ·) result = β-nf, can be reached by forward computation at leftmost redex, is unique, may not exist $\rightarrow$ weakly and strongly normalizing λ-terms
- ·) Y-combinator computes fixed point of every funtion, solves recursive equations
- ·) λ-calculus is Turing complete $\rightarrow$ Church-Turing thesis
- ·) negative aspects of λ-calculus as a model for logic:
  - self-application possible, counter-intuitive
  - existence of β-nf not guaranteed
  - every function has a fixed point, counter-intuitive

  will be eliminated by addition of types

Exercises 4

4.1: Construct a λ-term X such that $X =_\beta \lambda xy.xXy$.

$A \equiv \lambda zxy.xzy$

$X \equiv YA \equiv (\lambda y.(\lambda x.y(xx))(\lambda x.y(xx)))(\lambda fxy.xfy)$

$X \rightarrow_\beta (\lambda x.(\lambda fxy.xfy)(xx))(\lambda x.(\lambda fxy.xfy)(xx))$

$\twoheadrightarrow_\beta (\lambda wxy.x(ww)y)(\lambda wxy.x(ww)y)$

$\rightarrow_\beta \lambda xy.x((\lambda wxy.x(ww)y)(\lambda wxy.x(ww)y))y$

${}_\beta\twoheadleftarrow \lambda xy.x((\lambda x.(\lambda fxy.xfy)(xx))(\lambda x.(\lambda fxy.xfy)(xx)))y$

${}_\beta\leftarrow \lambda xy.xXy$, so $X =_\beta \lambda xy.xXy$

Type Theory, 25

Montag, 7. November 2022 01:30

4.2: Construct a λ-term X such that $Xxyz =_\beta xyzX$.

$A \equiv \lambda fxyz.xyzf$

$X \equiv YA \equiv (\lambda y.(\lambda x.y(xx))(\lambda x.y(xx)))(\lambda fxyz.xyzf)$

$$
\begin{aligned}
X &\to_\beta (\lambda x.(\lambda fxyz.xyzf)(xx))(\lambda x.(\lambda fxyz.xyzf)(xx)) \\
&\twoheadrightarrow_\beta (\lambda uxyz.xyz(uu))(\lambda uxyz.xyz(uu)) \\
&\to_\beta \lambda xyz.xyz((\lambda uxyz.xyz(uu))(\lambda uxyz.xyz(uu))) \\
&{}_\beta\twoheadleftarrow \lambda xyz.xyz((\lambda x.(\lambda fxyz.xyzf)(xx))(\lambda x.(\lambda fxyz.xyzf)(xx))) \\
&{}_\beta\leftarrow \lambda xyz.xyzX, \text{ so } Xxyz =_\beta xyzX
\end{aligned}
$$

4.3: Construct a λ-term fib such that fib N is the n-th Fibonacci number if $N \equiv \underbrace{\text{suc}(\text{suc}(\ldots(\text{suc}}_{n \text{ times}}\ \text{zero})\ldots))$.

$$
\begin{aligned}
\text{fib} \equiv Y(\lambda fn.&\text{ifte}\,(\text{iszero}\,(\text{pred}\ n))\ \text{one}\,(\text{ifte}\,(\text{iszero}\,(\text{pred}\,(\text{pred}\ n)))\ \text{one} \\
&(\text{add}\ (f(\text{pred}\ n))(f(\text{pred}\,(\text{pred}\ n))))))
\end{aligned}
$$

fib one $=_\beta$ fib two $=_\beta$ one,

fib three $=_\beta$ two, fib four $=_\beta$ three, fib five $=_\beta$ five, …

Type Theory, 26

Montag, 7. November 2022 16:00

2. Simply typed Lambda Calculus $\lambda\to$

Definition 2.1: The set $\mathbb{T}$ of all simple types

Let $\mathbb{V}$ be an infinite set ... the set of "type variables".

Let $\mathbb{T} = \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$.

(1) variable type / basic type (2) arrow type / function type

Example 2.2: $\overset{(1)}{\alpha},\ \overset{(1)}{\beta},\ \overset{(2)}{(\alpha\to\beta)},\ \overset{(2)}{((\gamma\to\alpha)\to(\alpha\to(\beta\to\gamma)))} \in \mathbb{T}$

Notation 2.3: elements of $\mathbb{V}$: $\alpha, \beta, \gamma, \sigma, \sigma', \sigma'', \tau_1, \tau_2, \tau_3, \ldots$

elements of $\mathbb{T}$: $A, B, C, X, X', X'', Y_1, Y_2, Y_3, \ldots$

Notation 2.4: (1) drop outermost parenthesis : $\alpha\to\beta = (\alpha\to\beta)$

to match associativity of term application

(2) arrows are right associative : $\alpha\to\beta\to\gamma = (\alpha\to(\beta\to\gamma))$

Definition 2.5: syntactical identity $\equiv$

(1) (var) $\alpha\equiv\alpha,\ \alpha\not\equiv\beta$

(2) (arr) $A\to B \equiv C\to D$ iff $A\equiv C$ and $B\equiv D$

Remark 2.6: We have to be careful about different types of variables.

- term variables : $a, b, c, \ldots \in V$
- type variables : $\alpha, \beta, \gamma, \ldots \in \mathbb{V}$
- meta variables : $A, B, C, \ldots \in \Lambda \cup \mathbb{T}$

Definition 2.7: statement, $A : T$, subject, type, declaration

If $A \in \Lambda$, $T \in \mathbb{T}$, then $A : T$ is called a statement with subject A and type T.

A declaration is a statement whose subject is a variable.

We read $A:T$ as "A is of type T" or "A has type T".

Every term is thought of as having a unique type: $A:S,\ A:T \Rightarrow S\equiv T$.

Type Theory, 27

Dienstag, 8. November 2022 15:30

Remark 2.8: **Outlook**

- if $A : \alpha \to \beta$, $B : \alpha$ then $AB : \beta$
- if $x : \alpha$, $A : \beta$ then $\lambda x : \alpha . A : \alpha \to \beta$

Note: if $f : \alpha \to \beta \to \gamma$, $x : \alpha$, $y : \beta$ then $fxy : \gamma$.
no brackets ↑ no brackets ↑ (Currying again guiding idea)

How to type a term: Church-typing and Curry-typing

Church-typing: prescribe a unique type to every variable upon its introduction, also known as **explicit typing**.

Curry-typing: find a type by search process such that rules of typing are satisfied, also known as **implicit typing**.

Example 2.9:
- **Explicit**
  - If $x : \alpha \to \alpha$, $y : (\alpha \to \alpha) \to \beta$ then $yx : \beta$.
  - If furthermore $z : \beta$, $u : \gamma$ then $\lambda z u . z : \beta \to \gamma \to \beta$ and $(\lambda z u . z)(yx) : \gamma \to \beta$.
- **Implicit**
  - Try to type $M \equiv (\lambda z u . z)(yx)$, i.e. find types for x, y, z, u such that M has a valid type:
    (1) M is an application, so $\lambda z u . z : A \to B$, $yx : A$, $M : B$.
    (2) $\lambda z u . z : A \to B$, so $z : A$, $\lambda u . z : B$.
    (3) $\lambda u . z : B$, so $B \equiv C \to D$, $u : C$, $z : D$ and thus $A \equiv D$.
    (4) yx is an application, so $y : E \to F$, $x : E$, $yx : F$ and thus $A \equiv D \equiv F$.
    (5) Choose $E \equiv \alpha$, $A \equiv \beta$, $C \equiv \gamma$, then $x : \alpha$, $y : \alpha \to \beta$, $z : \beta$, $u : \gamma$ and consequently $yx : \beta$, $\lambda u . z : \gamma \to \beta$, $\lambda z u . z : \beta \to \gamma \to \beta$, $M : \gamma \to \beta$.

Notation 2.10: from now on we use **explicit typing**.

Exercises 5

5.1: Give types to the following λ-terms or show that they cannot be typed:

(1) xxy (2) xyy (3) xyx (4) $x(xy)$ (5) $x(yx)$.

xx created by (app) $\Rightarrow x: A\to B, x: A \Rightarrow A \equiv A \to B$ ↯, so xxy cannot be typed

$x:\mathcal{L}\to\mathcal{L}\to\mathcal{L}, y:\mathcal{L} \Rightarrow xy:\mathcal{L}\to\mathcal{L} \Rightarrow xyy:\mathcal{L}$

xy created by (app) $\Rightarrow x: A\to B, y: A, xy: B$

xyx created by (app) $\Rightarrow xy: C\to D, x: C \Rightarrow$

$C \equiv A\to B \Rightarrow B \equiv A\to B\to D$ ↯, so xyx cannot be typed

$x:\mathcal{L}\to\mathcal{L}, y:\mathcal{L} \Rightarrow xy:\mathcal{L} \Rightarrow x(xy):\mathcal{L}$

$x:\mathcal{L}\to\mathcal{L}, y:(\mathcal{L}\to\mathcal{L})\to\mathcal{L} \Rightarrow yx:\mathcal{L} \Rightarrow x(yx):\mathcal{L}$

5.2: Find types for zero, one, and two.

zero $\equiv \lambda fx.x$, one $\equiv \lambda fx.fx$, two $\equiv \lambda fx.f(fx)$

$x:\mathcal{L}, f:\mathcal{L}\to\mathcal{L} \Rightarrow fx:\mathcal{L}, f(fx):\mathcal{L} \Rightarrow$ zero, one, two $: (\mathcal{L}\to\mathcal{L})\to\mathcal{L}\to\mathcal{L}$

5.3: Find types for $K \equiv \lambda xy.x$ and $S \equiv \lambda xyz.xz(yz)$.

$x:\mathcal{L}\to\mathcal{L}\to\mathcal{L}, y:\mathcal{L}\to\mathcal{L}, z:\mathcal{L} \Rightarrow K:(\mathcal{L}\to\mathcal{L}\to\mathcal{L})\to(\mathcal{L}\to\mathcal{L})\to\mathcal{L}\to\mathcal{L}\to\mathcal{L}$

$S:(\mathcal{L}\to\mathcal{L}\to\mathcal{L})\to(\mathcal{L}\to\mathcal{L})\to\mathcal{L}\to\mathcal{L}$

5.4: Find types for $A \equiv \lambda xyz.x(yz)$ and $B \equiv \lambda xyz.y(xz)x$.

$x:\mathcal{L}\to\mathcal{L}, y:\mathcal{L}\to\mathcal{L}, z:\mathcal{L} \Rightarrow A:(\mathcal{L}\to\mathcal{L})\to(\mathcal{L}\to\mathcal{L})\to\mathcal{L}\to\mathcal{L}$

$x:\mathcal{L}\to\mathcal{L}, y:\mathcal{L}\to(\mathcal{L}\to\mathcal{L})\to\mathcal{L}, z:\mathcal{L} \Rightarrow B:(\mathcal{L}\to\mathcal{L})\to(\mathcal{L}\to(\mathcal{L}\to\mathcal{L})\to\mathcal{L})\to\mathcal{L}\to\mathcal{L}$

Type Theory, 29

Dienstag, 8. November 2022 17:30

Definition 2.11: The set $\Lambda_{\mathbb{T}}$ of all pre-typed λ-terms

Let $\Lambda_{\mathbb{T}} = V \mid (\Lambda_{\mathbb{T}} \Lambda_{\mathbb{T}}) \mid (\lambda V : \mathbb{T}. \Lambda_{\mathbb{T}})$.

Definition 2.12: context, judgement

A context is a finite ordered list of declarations with different subjects:

$V_1 : B_1, \ldots, V_n : B_n$ with $V_1, \ldots, V_n \in V$ "turnstile"

If Γ is a context and $A:B$ is a statement, then $\Gamma \vdash A:B$ is called a judgement which we read as "in the context Γ, A has type B".

Derivation rules for $\lambda\!\to$

Definition 2.13:

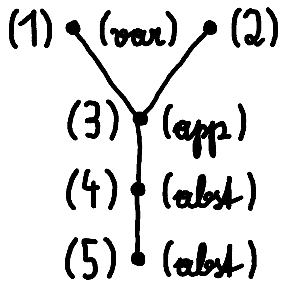
(var) $\Gamma \vdash X:T$ if $X:T \in \Gamma$

(app) $$\frac{\Gamma \vdash A : S \to T \qquad \Gamma \vdash B : S}{\Gamma \vdash AB : T}$$

(abst) $$\frac{\Gamma, X:S \vdash A:T}{\Gamma \vdash \lambda X{:}S.A : S \to T}$$

Example 2.14:

$$\cfrac{\cfrac{(1)\ y:\mathcal{L}\to\mathcal{B}, z:\mathcal{L} \vdash y:\mathcal{L}\to\mathcal{B} \qquad (2)\ y:\mathcal{L}\to\mathcal{B}, z:\mathcal{L} \vdash z:\mathcal{L}}{\cfrac{(3)\ y:\mathcal{L}\to\mathcal{B}, z:\mathcal{L} \vdash yz:\mathcal{B}}{(4)\ y:\mathcal{L}\to\mathcal{B} \vdash \lambda z{:}\mathcal{L}.yz : \mathcal{L}\to\mathcal{B}}}}{(5)\ () \vdash \lambda y{:}\mathcal{L}\to\mathcal{B}.\lambda z{:}\mathcal{L}.yz : (\mathcal{L}\to\mathcal{B})\to\mathcal{L}\to\mathcal{B}}$$



Flag format: flags represent context — first hint at **Curry-Howard isomorphism** — **Compare**: logic

(a) $\boxed{y:\mathcal{L}\to\mathcal{B}}$

(b) $\boxed{z:\mathcal{L}}$

(1) $y:\mathcal{L}\to\mathcal{B}$ — (var) on (a)

(2) $z:\mathcal{L}$ — (var) on (b)

(3) $yz:\mathcal{B}$ — (app) on (1),(2)

(4) $\lambda z{:}\mathcal{L}.yz : \mathcal{L}\to\mathcal{B}$ — (abst) on (3)

(5) $\lambda y{:}\mathcal{L}\to\mathcal{B}.\lambda z{:}\mathcal{L}.yz : (\mathcal{L}\to\mathcal{B})\to\mathcal{L}\to\mathcal{B}$ — (abst) on (4)

Logic:

$\boxed{\text{Assume } A}$

$\vdots$

B

$A \Rightarrow B$ ($\Rightarrow$-intro)

$$\frac{A\Rightarrow B \qquad A}{B} \quad (\Rightarrow\text{-elim})$$

Type Theory, 30

Mittwoch, 9. November 2022 12:30

Definition 2.14: **legal**

A pretyped λ-term $A \in \Lambda_{\mathbb{T}}$ is called legal if $\Gamma \vdash A : T$ for some context Γ and type T.

Example 2.15: $\lambda z : \mathcal{L}. yz$ is legal because $y : \mathcal{L} \to \mathcal{B} \vdash \lambda z : \mathcal{L}. yz : \mathcal{L} \to \mathcal{B}$.

Kinds of problems in type theory

(1) Well-typedness: $? \vdash \text{term} : ?$ (typability, legality)

(1a) Type assignment: $\text{context} \vdash \text{term} : ?$

(2) Type checking: $\text{context} \overset{?}{\vdash} \text{term} : \text{type}$

(3) Term finding: $\text{context} \vdash ? : \text{type}$ (the real problem: proving!)

In $\lambda\to$: algorithms for all 3 problems

Later: (3) undecidable!

Example 2.16:

(1) Find Γ, T such that $\Gamma \vdash \lambda y : \mathcal{L} \to \mathcal{B}. \lambda z : \mathcal{L}. yz : T$.

- Choose $\Gamma = \emptyset$ (no free variables)
- $\lambda y : \mathcal{L} \to \mathcal{B}. \lambda z : \mathcal{L}. yz : ?$
- (a) $\boxed{y : \mathcal{L} \to \mathcal{B}}$
  $\vdots$
  (y) $\lambda z : \mathcal{L}. yz : ?$
  (z) $\lambda y : \mathcal{L} \to \mathcal{B}. \lambda z : \mathcal{L}. yz : \ldots$ (abst) on (y)
- (a) $\boxed{y : \mathcal{L} \to \mathcal{B}}$
  (b) $\boxed{z : \mathcal{L}}$
  $\vdots$
  (x) $yz : ?$
  (y) $\lambda z : \mathcal{L}. yz : \ldots$ (abst) on (x)
  (z) $\lambda y : \mathcal{L} \to \mathcal{B}. \lambda z : \mathcal{L}. yz : \ldots$ (abst) on (y)

Type Theory, 31

Mittwoch, 9. November 2022 15:30

·) (a) $\boxed{y:\alpha\to\beta}$

(b) $\boxed{z:\alpha}$

$\vdots$

(w_1) $y:\,?$

$\vdots$

(w_2) $z:\,?$

(x) $yz:\ldots$ (app) on $(w_1),(w_2)$

(y) $\lambda z:\alpha.yz:\ldots$ (abst) on (x)

(z) $\lambda y:\alpha\to\beta.\lambda z:\alpha.yz:\ldots$ (abst) on (y)

·) (a) $\boxed{y:\alpha\to\beta}$

(b) $\boxed{z:\alpha}$

(w_1) $y:\alpha\to\beta$ (var) on (a)

(w_2) $z:\alpha$ (var) on (b)

(x) $yz:\beta$ (app) on $(w_1),(w_2)$

(y) $\lambda z:\alpha.yz:\alpha\to\beta$ (abst) on (x)

(z) $\lambda y:\alpha\to\beta.\lambda z:\alpha.yz:(\alpha\to\beta)\to\alpha\to\beta$ (abst) on (y)

Note: ·) Derivations are not unique in general.

·) Derivation fails iff term is not legal.

(2) Verify that $x:\alpha\to\alpha, y:(\alpha\to\alpha)\to\beta\vdash(\lambda z:\beta.\lambda u:\gamma.z)(yx):\gamma\to\beta$.

·) (a) $\boxed{x:\alpha\to\alpha}$

(b) $\boxed{y:(\alpha\to\alpha)\to\beta}$

$\vdots$

(z) $(\lambda z:\beta.\lambda u:\gamma.z)(yx):\gamma\to\beta$

·) (a) $\boxed{x:\alpha\to\alpha}$

(b) $\boxed{y:(\alpha\to\alpha)\to\beta}$

$\vdots$

(y_1) $\lambda z:\beta.\lambda u:\gamma.z:\,?$

$\vdots$

(y_2) $yx:\,?$

(z) $(\lambda z:\beta.\lambda u:\gamma.z)(yx):\gamma\to\beta$ (app) on $(y_1),(y_2)$

·) (a) $\boxed{x : \mathcal{L} \to \mathcal{L}}$

(b) $\boxed{y : (\mathcal{L} \to \mathcal{L}) \to \beta}$

(c) $\boxed{z : \beta}$

(d) $\boxed{u : \gamma}$

(w) $z : \beta$ (var) on (c)

(x) $\lambda u : \gamma . z : \gamma \to \beta$ (abst) on (w)

(y_1) $\lambda z : \beta . \lambda u : \gamma . z : \beta \to \gamma \to \beta$ (abst) on (x)

(y_2) $yx : \beta$ (app) on (b),(a)

(z) $(\lambda z : \beta . \lambda u : \gamma . z)(yx) : \gamma \to \beta$ (app) on (y_1),(y_2)

Convention: suppress non-essential uses of the (var)-rule

(3) Find A such that $() \vdash A : P \to Q \to P$ (find "inhabitant" of $P \to Q \to P$)

·) (z) $? : P \to Q \to P$

·) (a) $\boxed{x : P}$

⋮

(y) $? : Q \to P$

(z) $\ldots : P \to Q \to P$ (abst) on (y)

·) (a) $\boxed{x : P}$

(b) $\boxed{y : Q}$

⋮

(x) $? : P$

(y) $\ldots : Q \to P$ (abst) on (x)

(z) $\ldots : P \to Q \to P$ (abst) on (y)

·) (a) $\boxed{x : P}$

(b) $\boxed{y : Q}$

(x) $x : P$ (var) on (a)

(y) $\lambda y : Q . x : Q \to P$ (abst) on (x)

(z) $\lambda x : P . \lambda y : Q . x : P \to Q \to P$ (abst) on (y)

Type Theory, 33

Freitag, 18. November 2022 12:40

Exercises 6

6.1: Find Γ, T such that $\Gamma \vdash \lambda x:((\alpha\to\beta)\to\alpha).x(\lambda z:\alpha.y):T$ and give its complete derivation in flag format.

(a)	$\boxed{y:\beta}$	
(b)	$\quad\boxed{x:(\alpha\to\beta)\to\alpha}$	
(c)	$\quad\quad\boxed{z:\alpha}$	
(w)	$\quad\quad y:\beta$	(var) on (a)
(x)	$\quad\lambda z:\alpha.y:\alpha\to\beta$	(abst) on (w)
(y)	$\quad x(\lambda z:\alpha.y):\alpha$	(app) on (b),(x)
(z)	$\lambda x:(\alpha\to\beta)\to\alpha.x(\lambda z:\alpha.y):((\alpha\to\beta)\to\alpha)\to\alpha$	(abst) on (y)

6.2: Verify that $x:\delta\to\delta\to\alpha, y:\gamma\to\alpha, z:\alpha\to\beta \vdash \lambda u:\delta.\lambda v:\gamma.z(xuu):\delta\to\gamma\to\beta$ by giving its complete derivation in flag format.

(a)	$\boxed{x:\delta\to\delta\to\alpha}$	
(b)	$\quad\boxed{y:\gamma\to\alpha}$	
(c)	$\quad\quad\boxed{z:\alpha\to\beta}$	
(d)	$\quad\quad\quad\boxed{u:\delta}$	
(e)	$\quad\quad\quad\quad\boxed{v:\gamma}$	
(v)	$\quad\quad\quad\quad xu:\delta\to\alpha$	(app) on (a),(d)
(w)	$\quad\quad\quad\quad xuu:\alpha$	(app) on (v),(d)
(x)	$\quad\quad\quad\quad z(xuu):\beta$	(app) on (c),(w)
(y)	$\quad\quad\quad\lambda v:\gamma.z(xuu):\gamma\to\beta$	(abst) on (x)
(z)	$\quad\quad\lambda u:\delta.\lambda v:\gamma.z(xuu):\delta\to\gamma\to\beta$	(abst) on (y)

6.3: Find A such that $() \vdash A:((\alpha\to\gamma)\to\alpha)\to(\alpha\to\gamma)\to\beta\to\gamma$ and give its complete derivation in flag format.

(a)	$\boxed{x:(\alpha\to\gamma)\to\alpha}$	
(b)	$\quad\boxed{y:\alpha\to\gamma}$	
(c)	$\quad\quad\boxed{z:\beta}$	
(v)	$\quad\quad xy:\alpha$	(app) on (a),(b)
(w)	$\quad\quad y(xy):\gamma$	(app) on (b),(v)
(x)	$\quad\lambda z:\beta.y(xy):\beta\to\gamma$	(abst) on (w)
(y)	$\lambda y:\alpha\to\gamma.\lambda z:\beta.y(xy):(\alpha\to\gamma)\to\beta\to\gamma$	(abst) on (x)
(z)	$\lambda x:(\alpha\to\gamma)\to\alpha.\lambda y:\alpha\to\gamma.\lambda z:\beta.y(xy):((\alpha\to\gamma)\to\alpha)\to(\alpha\to\gamma)\to\beta\to\gamma$	(abst) on (y)

Type Theory, 34

Freitag, 11. November 2022 16:20

Types and logical statements: the Curry-Howard-isomorphism

Idea: read $\rightarrow$ as $\Rightarrow$, so $P \rightarrow Q \rightarrow P$ becomes $P \Rightarrow Q \Rightarrow P$ (in the empty context, i.e. for arbitrary P, Q).

$$
\begin{array}{ll}
(a) & \boxed{x : P} \\
(b) & \quad \boxed{y : Q} \\
(x) & \quad x : P \\
(y) & \lambda y{:}Q.x : Q \rightarrow P \\
(z) & \lambda x{:}P.\lambda y{:}Q.x : P \rightarrow Q \rightarrow P
\end{array}
$$

(a) Assume x is a proof of P.

(b) Assume y is a proof of Q.

(x) Then x is (still) a proof of P.

(y) So the function mapping y to x sends a proof of Q to a proof of P.

(z) Thus, $\lambda x{:}P.\lambda y{:}Q.x$ proves $P \Rightarrow Q \Rightarrow P$ in the empty context.

The above observation is called Curry-Howard-isomorphism or PAT-interpretation where PAT stands for both "propositions as types" and "proofs as terms".

The final term $\lambda x{:}P.\lambda y{:}Q.x$ encodes its derivation and the statement it proves (both can be recovered by the "well-typedness"-algorithm).

General properties of $\lambda\rightarrow$

Remark 2.17: the notions syntactical identity, $\equiv$, Sub, (proper) subterm, free, bound, binding, FV, BV, BiV, closed, combinator, $\Lambda^0_{\mathbb{T}}$, $\Lambda^{x\rightarrow y}$, renaming, alpha conversion, $=_\alpha$, alpha convertibles, and alpha equivalent generalize trivially from λ.

Definition 2.18: domain, subcontext, permutation, projection

- The domain $\mathrm{dom}(\Gamma)$ of a context $\Gamma \equiv v_1 : A_1, \dots, v_n : A_n$ is the list of variables $(v_1, \dots, v_n)$.
- Γ' is a subcontext of Γ ($\Gamma' \subseteq \Gamma$) if all declarations in Γ' also occur in Γ in the same order.
- Γ' is a permutation of Γ if Γ and Γ' contain the same declarations.
- For a context Γ and a list of variables Φ, the projection of Γ on Φ ($\Gamma \upharpoonright \Phi$) is the subcontext Γ' of Γ satisfying $\mathrm{dom}(\Gamma') = \mathrm{dom}(\Gamma) \cap \Phi$.

Type Theory, 35

Montag, 14. November 2022 12:30

Example 2.19:
- $\cdot)$ $dom(()) = ()$, $dom(y:\sigma, x_1:\tau_1, x_2:\tau_2) = (y, x_1, x_2)$
- $\cdot)$ $() \subseteq x_1:\tau_1 \subseteq y:\sigma, x_1:\tau_1, x_2:\tau_2$
- $\cdot)$ $x_1:\tau_1, y:\sigma, x_2:\tau_2$ is a permutation of $y:\sigma, x_1:\tau_1, x_2:\tau_2$
- $\cdot)$ $y:\sigma, x_1:\tau_1, x_2:\tau_2 \restriction (y, x_2) = y:\sigma, x_2:\tau_2$

Lemma 2.20 (free variable, thinning, condensing, permutation):

(1) If $\Gamma \vdash A:T$, then $FV(A) \subseteq dom(\Gamma)$

(2) If $\Gamma' \vdash A:T$, $\Gamma' \subseteq \Gamma''$, then $\Gamma'' \vdash A:T$ ← after potentially renaming binding variables in A as to not collide with declarations in Γ''

(3) If $\Gamma \vdash A:T$, then $\Gamma \restriction FV(A) \vdash A:T$

(4) If $\Gamma' \vdash A:T$, Γ'' permutation of Γ', then $\Gamma'' \vdash A:T$

Proof:

(1) Proof by induction: assume statement is true for all judgements that went into the derivation of $J \equiv \Gamma \vdash A:T$.

induction base, since (var) rule has no premisses

Case 1: J is the conclusion of the (var) rule.

Then J is of the form $V_1:T_1, \dots, V_n:T_n \vdash A:T$ with $A \equiv V_i$, $T \equiv T_i$ for some $i \in \{1, \dots, n\}$ and $FV(A) = \{A\} \subset \{V_1, \dots, V_n\} = dom(\Gamma)$. ✓

Case 2: J is the conclusion of the (app) rule.

Then J is of the form $\Gamma \vdash BC:T$ with $A \equiv BC$ and J is the conclusion of $\Gamma \vdash B:S \to T$, $\Gamma \vdash C:S$.

(IH)

Consequently, $FV(A) = FV(B) \cup FV(C) \subset dom(\Gamma)$. ✓

Case 3: J is the conclusion of the (abst) rule.

Then J is of the form $\Gamma \vdash \lambda B:R.C : R \to S$ with $A \equiv \lambda B:R.C$, $T \equiv R \to S$ and J is the conclusion of $\Gamma, B:R \vdash C:S$.

(IH)

Consequently, $FV(A) = FV(C) \setminus \{B\} \subset dom(\Gamma, B:R) \setminus \{B\} = dom(\Gamma)$. ✓

$\Gamma, B:R$ is a context and thus its subjects are pairwise different

(2), (3), (4) can be proved analogously.

Type Theory, 36

Samstag, 19. November 2022 14:54

Remark 2.21: (4) implies that in $\lambda\to$ we could define contexts using sets instead of lists but in later systems lists are necessary because the order of declarations will be important.

Lemma 2.22 (generation):

(1) $\Gamma \vdash A : T$ with $A \in V \Rightarrow A : T \in \Gamma$

(2) $\Gamma \vdash AB : T \Rightarrow \exists S \in \mathbb{T}: \Gamma \vdash A : S \to T,\ \Gamma \vdash B : S$

(3) $\Gamma \vdash \lambda A : R . B : T \Rightarrow \exists S \in \mathbb{T}: \Gamma, A : R \vdash B : S,\ T \equiv R \to S$

Proof: follows directly from the definitions.

Lemma 2.23 (subterm): if $A \in \Lambda_{\mathbb{T}}$ is legal, then every subterm of A is legal.

Proof: easy induction on A and Generation Lemma.

Example 2.24: Let $A \equiv (\lambda z : \beta . \lambda u : \gamma . z)(yx)$ and $B \equiv \lambda u : \gamma . z \in Sub(A)$.

Then $x : \mathcal{L} \to \mathcal{L}, y : (\mathcal{L} \to \mathcal{L}) \to \beta \vdash A : \gamma \to \beta$,

$x : \mathcal{L} \to \mathcal{L}, y : (\mathcal{L} \to \mathcal{L}) \to \beta, z : \beta \vdash B : \gamma \to \beta$ or $z : \mathcal{L} \vdash B : \gamma \to \mathcal{L}$.

Lemma 2.25 (uniqueness of types): $\Gamma \vdash A : S, \Gamma \vdash A : T \Rightarrow S \equiv T$.

Proof: easy induction on A.

Theorem 2.26: In $\lambda\to$ the following problems are decidable:

(1) Well-typedness: $? \vdash \text{term} : ?$

(1a) Type assignment: $\text{context} \vdash \text{term} : ?$

(2) Type checking: $\text{context} \overset{?}{\vdash} \text{term} : \text{type}$

(3) Term finding: $\text{context} \vdash ? : \text{type}$.

Proof: formalize algorithms demonstrated before.

Type Theory, 37

Samstag, 19. November 2022 16:43

Reduction in $\lambda\!\to$

Definition 2.27: substitution

(1)(var) $x[x:=A]=A$, $y[x:=A]=y$

(2)(app) $(BC)[x:=A]=(B[x:=A])(C[x:=A])$

(3)(abst) $(\lambda x:T.B)[x:=A]=\lambda x:T.B$

$(\lambda y:T.B)[x:=A]=\lambda z:T.(B^{y\to z}[x:=A])$ where $z\notin FV(A)$

Lemma 2.28 (substitution):

note: $\Gamma'\vdash B:S$ implies $X\notin FV(B)$ by the Free Variable Lemma (2.20(1))

If $\Gamma',X:S,\Gamma''\vdash A:T$ and $\Gamma'\vdash B:S$, then $\Gamma',\Gamma''\vdash A[X:=B]:T$.

Exercises 7

7.1: Prove the Substitution Lemma by induction.

Note that $\Gamma',\Gamma''\vdash B:S$ by the Thinning Lemma (2.20(2)).

Proof by induction: assume statement is true for all judgements that went into the derivation of $J\equiv\Gamma',X:S,\Gamma''\vdash A:T$ and show $J'\equiv\Gamma',\Gamma''\vdash A[X:=B]:T$.

induction base, since (var) rule has no premises

Case 1: J is the conclusion of the (var) rule.

Then J is of the form $A_1:T_1,\dots,A_m:T_m, X:S, A_{m+1}:T_{m+1},\dots,A_{m+n}:T_{m+n}\vdash A:T$ with $A\equiv A_i$, $T\equiv T_i$ for some $i\in\{1,\dots,m+n\}$ or $A\equiv X$, $T\equiv S$.

If $A\not\equiv X$ then $A[X:=B]=A$, otherwise $A[X:=B]=B$ and $S\equiv T$. In both cases J' holds. ✓

Case 2: J is the conclusion of the (app) rule.

Then J is of the form $\Gamma',X:S,\Gamma''\vdash EF:T$ with $A\equiv EF$ and J is the conclusion of $\Gamma',X:S,\Gamma''\vdash E:R\to T$, $\Gamma',X:S,\Gamma''\vdash F:R$.

Consequently, $\Gamma',\Gamma''\underset{(IH)}{\vdash}\underbrace{(E[X:=B])(F[X:=B])}_{=A[X:=B]}:T$ and J' holds. ✓

Case 3: J is the conclusion of the (abst) rule.

Then J is of the form $\Gamma',X:S,\Gamma''\vdash\lambda Y:U.E:U\to V$ with $A\equiv\lambda Y:U.E$, $T\equiv U\to V$

$\Gamma'\vdash B:S$ and Free Variable Lemma (2.20(1))

and J is the conclusion of $\Gamma',X:S,\Gamma'',Y:U\vdash E:V$ and $Y\notin FV(B)$.

Consequently, $\Gamma',\Gamma'',Y:U\underset{(IH)}{\vdash}E[X:=B]:V$, so $\Gamma',\Gamma''\underset{(abst)}{\vdash}\underbrace{\lambda Y:U.E[X:=B]}_{=A[X:=B]\ \leftarrow\ Y\notin FV(B)}:T$ and J' holds. ✓

Definition 2.29: one step β-reduction, $\to_\beta$

(1) (basis) $(\lambda x:T.A)B \to_\beta A[x:=B]$

(2) (compatibility) If $A \to_\beta B$, then $AC \to_\beta BC$, $CA \to_\beta CB$, $\lambda x:T.A \to_\beta \lambda x:T.B$.

Remark 2.30: The notions redex, contractum, β-reduction, $\twoheadrightarrow_\beta$, β-conversion, $=_\beta$, beta convertibles, beta equivalent, β-normal form, β-normalizing, reduction path, weak normalization, strong normalization generalize trivially from λ.

Remark 2.31: The Church-Rosser Theorem and its corollaries hold in $\lambda\!\to$.

Lemma 2.32 (subject reduction):

If $\Gamma \vdash A:T$ and $A \twoheadrightarrow_\beta B$, then $\Gamma \vdash B:T$.

Proof:

It suffices to show $\Gamma \vdash B:T$ if $A \to_\beta B$ and we only consider the (basis) case (full statement then follows by induction).

Then $A \equiv (\lambda x:S.C)D$ with $\Gamma, x:S \vdash C:T$, $\Gamma \vdash D:S$ and $B \equiv C[x:=D]$.

By the Substitution Lemma we get $\Gamma \vdash C[x:=D]:T$, i.e. $\Gamma \vdash B:T$. ✓

Example 2.33: $x:\mathcal{L}\to\mathcal{L}, y:(\mathcal{L}\to\mathcal{L})\to\beta \vdash (\lambda z:\beta.\lambda u:\gamma.z)(yx):\gamma\to\beta$ and

$x:\mathcal{L}\to\mathcal{L}, y:(\mathcal{L}\to\mathcal{L})\to\beta \vdash \lambda u:\gamma.yx:\gamma\to\beta$.

Theorem 2.34 (strong normalization): every legal term in $\lambda\!\to$ is strongly normalizing.

Proof:

A function $f:\mathrm{dom}(f)\to\Lambda_\mathbb{T}$ is called substitution if $\mathrm{dom}(f)\subseteq V$. simultaneous substitution

For a substitution f we define $\bar{f}:\Lambda_\mathbb{T}\to\Lambda_\mathbb{T}$, $A\mapsto A[V_1:=f(V_1),\dots,V_n:=f(V_n)]$ where $FV(A)\cap\mathrm{dom}(f)=\{V_1,\dots,V_n\}$.

For a type T let $\mathcal{L}(T) := \begin{cases}\{A\in\Lambda_\mathbb{T} \mid A \text{ is strongly normalizing}\} =: SN & \text{if } T\in\mathbb{V}\\ \{A\in\Lambda_\mathbb{T} \mid \forall B\in\mathcal{L}(R): AB\in\mathcal{L}(S)\} & \text{if } T\equiv R\to S\end{cases}$.

For a context Γ let $\mathcal{L}(\Gamma) := \{f \text{ substitution} \mid \mathrm{dom}(f)=\mathrm{dom}(\Gamma) \wedge \forall X:T\in\Gamma: f(X)\in\mathcal{L}(T)\}$.

Type Theory, 39

Sonntag, 27. November 2022 17:00

Claim 1: $\{XN_1 \dots N_k \mid X \in \mathbb{V} \wedge k \in \mathbb{N}_0 \wedge N_1, \dots, N_k \in SN\} \subset \mathcal{L}(T) \subset SN$ for all types T.

Proof by induction on T:

Case 1: $T \in \mathbb{V}$.

Then $\mathcal{L}(T) = SN$. ✓

Case 2: $T \equiv R \to S$.

If $X \in \mathbb{V}$, $k \in \mathbb{N}_0$, $N_1, \dots, N_k \in SN$ and $B \in \mathcal{L}(R)$, then $B \overset{(IH)}{\in} SN$ and thus $XN_1 \dots N_k B \overset{(IH)}{\in} \mathcal{L}(S)$. Consequently, $XN_1 \dots N_k \in \mathcal{L}(T)$. ✓

If $A \in \mathcal{L}(T)$ and $B \in \mathcal{L}(R) \overset{(IH)}{\neq} \emptyset$, then $AB \in \mathcal{L}(S) \overset{(IH)}{\subset} SN$ and thus $A \in SN$. ✓

Claim 2: $\mathcal{L}(T)$ is closed under "β-expansion": if $C[x := D] \in \mathcal{L}(T)$, $x \in FV(C)$, then $(\lambda x : U . C) D \in \mathcal{L}(T)$.

i.e. the substitution actually does something

Case 1: $T \in \mathbb{V}$.

Then $\mathcal{L}(T) = SN$ and clearly $(\lambda x : U . C) D \in SN$ if $C[x := D] \in SN$. ✓

Case 2: $T \equiv R \to S$.

Let $(R_0, S_0) \equiv (R, S)$, $\forall k \in \mathbb{N} : (R_k, S_k) \equiv \begin{cases} (R_{k-1}, S_{k-1}) & \text{if } S_{k-1} \in \mathbb{V} \\ (P, Q) & \text{if } S_{k-1} \equiv P \to Q \end{cases}$,

$K := \min \{k \in \mathbb{N}_0 \mid S_k \in \mathbb{V}\}$,

i.e. $T \equiv R_0 \to S_0 \equiv R_0 \to R_1 \to S_1 \equiv \dots \equiv R_0 \to \dots \to R_K \to S_K \in \mathbb{V}$.

Then $C[x := D] \in \mathcal{L}(T) \Rightarrow$

$\Rightarrow \forall B_0 \in \mathcal{L}(R_0) : C[x := D] B_0 \in \mathcal{L}(S_0)$

$\Rightarrow \forall B_0 \in \mathcal{L}(R_0) : \forall B_1 \in \mathcal{L}(R_1) : C[x := D] B_0 B_1 \in \mathcal{L}(S_1)$

$\Rightarrow \dots$

$\Rightarrow \forall B_0 \in \mathcal{L}(R_0) : \dots \forall B_K \in \mathcal{L}(R_K) : C[x := D] B_0 \dots B_K \in \mathcal{L}(S_K) \overset{S_K \in \mathbb{V}}{=} SN$.

$x \in FV(C)$

Thus $\forall B_0 \in \mathcal{L}(R_0) : \dots \forall B_K \in \mathcal{L}(R_K) : (\lambda x : U . C) D B_0 \dots B_K \in SN = \mathcal{L}(S_K) \Rightarrow$

$\Rightarrow \forall B_0 \in \mathcal{L}(R_0) : \dots \forall B_{K-1} \in \mathcal{L}(R_{K-1}) : (\lambda x : U . C) D B_0 \dots B_{K-1} \in \mathcal{L}(S_{K-1})$

$\Rightarrow \dots$

$\Rightarrow \forall B \in \mathcal{L}(R_0) : (\lambda x : U . C) D B_0 \in \mathcal{L}(S_0)$

$\Rightarrow (\lambda x : U . C) D \in \mathcal{L}(T)$. ✓

Claim 3: if $f_\Gamma : dom(\Gamma) \to \Lambda_\pi, X \mapsto X$ then $f_\Gamma \in \mathcal{L}(\Gamma)$.

If $X : T \in \Gamma$ then $f_\Gamma(X) = X \in \mathcal{L}(T)$ by Claim 1. ✓

Claim 4: if $\Gamma \vdash A : T$ and $f \in \mathcal{L}(\Gamma)$ then $\bar{f}(A) \in \mathcal{L}(T)$.

Proof by **induction** on A:

Case 1: $A \in V$.

$A \in dom(\Gamma)$ by Generation Lemma (2.22(1))

Then $\bar{f}(A) = f(A) \in \mathcal{L}(T)$. ✓

Case 2: $A \equiv BC$.

Generation Lemma (2.22(2))

Then $\Gamma \vdash B : S \to T$, $\Gamma \vdash C : S$ for some type S and consequently
$\bar{f}(B) \overset{(IH)}{\in} \mathcal{L}(S \to T)$, $\bar{f}(C) \overset{(IH)}{\in} \mathcal{L}(S)$.

By definition of $\mathcal{L}$ we thus get $\bar{f}(BC) = \bar{f}(B)\bar{f}(C) \in \mathcal{L}(T)$. ✓

Case 3: $A \equiv \lambda X:R.C$.

Generation Lemma (2.22(3))

Then $\Gamma, X:R \vdash C : S$, $T \equiv R \to S$ for some type S.

Let $B \in \mathcal{L}(R)$ and $g : dom(f) \cup \{X\} \to \Lambda_{\mathbb{T}}$, $X \not\equiv Y \mapsto f(Y)$, $X \mapsto B$.

Then $g \in \mathcal{L}(\Gamma, X:R)$ and thus $\bar{f}(C)[X := B] \equiv \bar{g}(C) \overset{(IH)}{\in} \mathcal{L}(S)$.

after possible $\mathcal{L}$-conversion of A
such that $X \notin FV(f(Y))$ for all $Y \in dom(f)$

Claim 2 then implies $\bar{f}(\lambda X:R.C)B = (\lambda X:R.\bar{f}(C))B \in \mathcal{L}(S)$

and consequently $\bar{f}(\lambda X:R.C) \in \mathcal{L}(T)$. ✓

Alltogether we get $\Gamma \vdash A : T \Rightarrow A = \bar{f}_{\Gamma}(A) \overset{\text{Claim 3,4}}{\in} \mathcal{L}(T) \overset{\text{Claim 1}}{\subseteq} SN$. ✓

Remark 2.35: Consequences for $\lambda\to$:

- There is no self-application in $\lambda\to$ (Generation Lemma).
- Existence of β-nfs is guaranteed (Strong Normaliztion Theorem).
- Not every legal $\lambda\to$-term has a fixed point.
- $\lambda\to$ is not Turing complete, but natural numbers, $+$, $\cdot$ can be defined.

  Class of functions on natural numbers definable in $\lambda\to$: generalized polynomials.

  piecewise polynomial functions where cases are defined by whether variables do or do not vanish

  Make $\lambda\to$ Turing complete by introducing Y-combinator:

  $Y_T : (T \to T) \to T$, $Y_T f \to_{D} f(Y_T f)$.

Type Theory, 41

Mittwoch, 30. November 2022 16:31

3. Second order typed Lambda Calculus $\lambda 2$

In $\lambda\!\rightarrow$:
- Terms depending on terms.
- Abstraction from terms.
- Application of terms.

In $\lambda 2$:
- Terms depending on types.
- Abstraction from types.
- Application of (to) types.

Example 3.1:

(1) "The" identity function:

$\lambda x:\alpha.x$, $\lambda x:nat.x$, $\lambda x:nat\to bool.x$ ... identity functions on α, nat, $nat\to bool$.

$\lambda\alpha:*.\lambda x:\alpha.x$ ... "the" identity function.

$\lambda\alpha$: second order abstraction; $*$: type of all types

need second order abstraction, application and β-reduction

β-reduction: $(\lambda\alpha:*.\lambda x:\alpha.x)\,nat \to_\beta \lambda x:nat.x$

(2) Iteration:

$\lambda x:\alpha.f(fx)$ ... iteration of f.

$D :\equiv \lambda\alpha:*.\lambda f:\alpha\to\alpha.\lambda x:\alpha.f(fx)$ ... general iteration.

$suc: nat\to nat \Rightarrow D\,nat\,suc \twoheadrightarrow_\beta \lambda x:nat.suc(suc\,x)$.

(3) Composition:

$\circ :\equiv \lambda\alpha:*.\lambda\beta:*.\lambda\gamma:*.\lambda f:\alpha\to\beta.\lambda g:\beta\to\gamma.\lambda x:\alpha.g(fx)$ ... general composition.

$F:A\to B, G:B\to C \Rightarrow \circ ABCFG \twoheadrightarrow_\beta \lambda x:A.G(Fx)$.

Product types (Π-types)

What is the type of $\lambda\alpha:*.\lambda x:\alpha.x$? Maybe $\lambda\alpha:*.\lambda x:\alpha.x : * \to(\alpha\to\alpha)$?

Problem: We want to identify $\lambda\alpha:*.\lambda x:\alpha.x$ with $\lambda\beta:*.\lambda x:\beta.x$ (α-conversion)

But then:
$$\begin{array}{ccc} \lambda\alpha:*.\lambda x:\alpha.x & : & *\to(\alpha\to\alpha). \\ \equiv & & \not\equiv \\ \lambda\beta:*.\lambda x:\beta.x & : & *\to(\beta\to\beta) \end{array}$$

new binder in addition to λ (binds types)

Solution: Introduce product types:
$$\begin{array}{ccc} \lambda\alpha:*.\lambda x:\alpha.x & : & \Pi\alpha:*.(\alpha\to\alpha). \\ \equiv & & \equiv \\ \lambda\beta:*.\lambda x:\beta.x & : & \Pi\beta:*.(\beta\to\beta) \end{array}$$

Example 3.2: ·) $\lambda\alpha:*.\lambda f:\alpha\to\alpha.\lambda x:\alpha.f(fx):\Pi\alpha:*.(\alpha\to\alpha)\to\alpha\to\alpha$.

·) $\lambda\alpha:*.\lambda\beta:*.\lambda\gamma:*.\lambda f:\alpha\to\beta.\lambda g:\beta\to\gamma.\lambda x:\alpha.g(fx):$

$$\Pi\alpha:*.\Pi\beta:*.\Pi\gamma:*.(\alpha\to\beta)\to(\beta\to\gamma)\to\alpha\to\gamma.$$

The system λ2

Definition 3.3: λ2-types, $\mathbb{T}_2$, λ2-terms, $\Lambda_{\mathbb{T}_2}$

$$\mathbb{T}_2 = \underbrace{\mathbb{V}}_{\text{variable}} \mid \underbrace{(\mathbb{T}_2\to\mathbb{T}_2)}_{\text{arrow type}} \mid \underbrace{(\Pi\mathbb{V}:*.\mathbb{T}_2)}_{\text{product type}}.$$

$$\Lambda_{\mathbb{T}_2} = \underbrace{\mathbb{V}}_{\text{variable}} \mid \underbrace{(\Lambda_{\mathbb{T}_2}\Lambda_{\mathbb{T}_2})}_{\text{first order application}} \mid \underbrace{(\Lambda_{\mathbb{T}_2}\mathbb{T}_2)}_{\text{second order application}} \mid \underbrace{(\lambda\mathbb{V}:\mathbb{T}_2.\Lambda_{\mathbb{T}_2})}_{\text{first order abstraction}} \mid \underbrace{(\lambda\mathbb{V}:*.\Lambda_{\mathbb{T}_2})}_{\text{second order abstraction}}.$$

Notation 3.4:
·) Outermost parenthesis may be omitted.
·) Application is left-associative, abstraction is right-associative.
·) Application and $\to$ take precedence over λ- and Π-abstraction.
·) Successive λ- and Π-abstractions of the same type may be combined.
·) Arrow types are right-associative.

Example 3.5: $\Pi\alpha,\beta:*.\alpha\to\beta\to\alpha = (\Pi\alpha:*.(\Pi\beta:*.(\alpha\to(\beta\to\alpha))))$.

Remark 3.6: all notions of simply typed λ-calculus $\lambda\to$ that aren't explicitly redefined generalize trivially to λ2 and all upcoming typing systems.

Definition 3.7: statement, $A:T$, $T:*$, (λ2-) context, domain

If $A\in\Lambda_{\mathbb{T}_2}$, $T\in\mathbb{T}_2$, then $A:T$ and $T:*$ are called statements.

λ2-contexts and domains are defined recursively:

·) $()$ is a λ2-context, $\mathrm{dom}(()) = ()$. ← empty list

·) If Γ is a λ2-context, $T\in\mathbb{V}$ with $T\notin\mathrm{dom}(\Gamma)$,
then $\Gamma,T:*$ is a λ2-context, $\mathrm{dom}(\Gamma,T:*) = \mathrm{dom}(\Gamma)\overset{\text{list concatenation}}{\cdot}(T)$.

·) If Γ is a λ2-context, $T\in\mathbb{T}_2$ with $FV(T)\subset\mathrm{dom}(\Gamma)$, $X\in\mathbb{V}\setminus\mathrm{dom}(\Gamma)$,
then $\Gamma,X:T$ is a λ2-context, $\mathrm{dom}(\Gamma,X:T) = \mathrm{dom}(\Gamma)\cdot(X)$.

Type Theory, 43

Dienstag, 6. Dezember 2022 13:16

Example 3.8: $()$; $\mathcal{L}:*$; $\mathcal{L}:*, x:\mathcal{L}\to\mathcal{L}$; $\mathcal{L}:*, x:\mathcal{L}\to\mathcal{L}, \beta:*$; $\Gamma \equiv \mathcal{L}:*, x:\mathcal{L}\to\mathcal{L}, \beta:*, y:\mathcal{L}\to\beta$ are $\lambda 2$-contexts and $\mathrm{dom}(\Gamma) = (\mathcal{L}, x, \beta, y)$.

Derivation rules for $\lambda 2$

Definition 3.9:

(var) $\Gamma \vdash X:T$ if Γ is a $\lambda 2$-context and $X:T \in \Gamma$

(silent) (app)
$$\frac{\Gamma \vdash A : S \to T \qquad \Gamma \vdash B:S}{\Gamma \vdash AB:T}$$

(abst)
$$\frac{\Gamma, X:S \vdash A:T}{\Gamma \vdash \lambda X:S.A : S\to T}$$

might as well be called (var2)

(form) $\Gamma \vdash T:*$ if Γ is a $\lambda 2$-context, $T \in \mathbb{T}_2$, and $FV(T) \subset \mathrm{dom}(\Gamma)$

(silent) (app2)
$$\frac{\Gamma \vdash A : \Pi X:*.C \qquad \Gamma \vdash T:*}{\Gamma \vdash AT : C[X:=T]}$$

(abst2)
$$\frac{\Gamma, X:* \vdash A:T}{\Gamma \vdash \lambda X:*.A : \Pi X:*.T}$$

Example 3.10: Find $T \in \mathbb{T}_2$ such that $() \vdash \lambda\mathcal{L}:*.\lambda f:\mathcal{L}\to\mathcal{L}.\lambda x:\mathcal{L}.f(fx) : T$.

·) (a) $\boxed{\mathcal{L}:*}$
$\vdots$
(y) $\lambda f:\mathcal{L}\to\mathcal{L}.\lambda x:\mathcal{L}.f(fx) : ?$
(z) $\lambda\mathcal{L}:*.\lambda f:\mathcal{L}\to\mathcal{L}.\lambda x:\mathcal{L}.f(fx) : \ldots$ (abst2) on (y)

·) (a) $\boxed{\mathcal{L}:*}$
(b) $\boxed{f:\mathcal{L}\to\mathcal{L}}$
(c) $\boxed{x:\mathcal{L}}$
$\vdots$
(w) $f(fx) : ?$
(x) $\lambda x:\mathcal{L}.f(fx) : \ldots$ (abst) on (w)
(y) $\lambda f:\mathcal{L}\to\mathcal{L}.\lambda x:\mathcal{L}.f(fx) : \ldots$ (abst) on (x)
(z) $\lambda\mathcal{L}:*.\lambda f:\mathcal{L}\to\mathcal{L}.\lambda x:\mathcal{L}.f(fx) : \ldots$ (abst2) on (y)

Type Theory, 44

Dienstag, 6. Dezember 2022 15:53

·) (a) $\boxed{\mathcal{L} : *}$

(b) $\boxed{f : \mathcal{L} \to \mathcal{L}}$

(c) $\boxed{x : \mathcal{L}}$

(v) $fx : \mathcal{L}$ (appl) on (b), (c)

(w) $f(fx) : \mathcal{L}$ (appl) on (b), (v)

(x) $\lambda x : \mathcal{L}. f(fx) : \mathcal{L} \to \mathcal{L}$ (abst) on (w)

(y) $\lambda f : \mathcal{L} \to \mathcal{L}. \lambda x : \mathcal{L}. f(fx) : (\mathcal{L} \to \mathcal{L}) \to \mathcal{L} \to \mathcal{L}$ (abst) on (x)

(z) $\lambda \mathcal{L} : *. \lambda f : \mathcal{L} \to \mathcal{L}. \lambda x : \mathcal{L}. f(fx) :$ (abst2) on (y)

$\Pi \mathcal{L} : *. (\mathcal{L} \to \mathcal{L}) \to \mathcal{L} \to \mathcal{L}$

So $() \vdash \lambda \mathcal{L} : *. \lambda f : \mathcal{L} \to \mathcal{L}. \lambda x : \mathcal{L}. f(fx) : \Pi \mathcal{L} : *. (\mathcal{L} \to \mathcal{L}) \to \mathcal{L} \to \mathcal{L}$

and thus $\Gamma \vdash \lambda \mathcal{L} : *. \lambda f : \mathcal{L} \to \mathcal{L}. \lambda x : \mathcal{L}. f(fx) : \Pi \mathcal{L} : *. (\mathcal{L} \to \mathcal{L}) \to \mathcal{L} \to \mathcal{L}$

for every $\lambda 2$ - context Γ by the Thinning Lemma. (still true in $\lambda 2$)

(A) $\Gamma \vdash nat : *$ (assumption)

(B) $\Gamma \vdash (\lambda \mathcal{L} : *. \lambda f : \mathcal{L} \to \mathcal{L}. \lambda x : \mathcal{L}. f(fx)) nat :$ (appl2) on (z), (A)

$(nat \to nat) \to nat \to nat$

(C) $\Gamma \vdash suc : nat \to nat$ (assumption)

(D) $\Gamma \vdash (\lambda \mathcal{L} : *. \lambda f : \mathcal{L} \to \mathcal{L}. \lambda x : \mathcal{L}. f(fx)) nat\ suc :$ (appl) on (B), (C)

$nat \to nat$

(E) $\Gamma \vdash two : nat$ (assumption)

(F) $\Gamma \vdash (\lambda \mathcal{L} : *. \lambda f : \mathcal{L} \to \mathcal{L}. \lambda x : \mathcal{L}. f(fx)) nat\ suc\ two :$ (appl) on (D), (E)

nat

$\mathcal{L}$-conversion and B-reduction in $\lambda 2$

Definition 3.11: $=_{\mathcal{L}}$

(1a) (renaming, term) If $y \notin FV(A) \cup Bi(A)$, then $\lambda x : T. A =_{\mathcal{L}} \lambda y : T. A^{x \to y}$

(1b) (renaming, type) If $B \notin FV(A) \cup Bi(A)$, then $\lambda \mathcal{L} : *. A =_{\mathcal{L}} \lambda B : *. A^{\mathcal{L} \to B}$

If $B \notin FV(T) \cup Bi(T)$, then $\Pi \mathcal{L} : *. T =_{\mathcal{L}} \Pi B : *. T^{\mathcal{L} \to B}$

(2), (3a), (3b), (3c) (compatibility), (reflexivity), (symmetry), (transitivity) As in $\lambda \to$.

Type Theory, 45

Mittwoch, 7. Dezember 2022 19:53

Definition 3.12: $\to_\beta$

(1a) (basis, first order) $(\lambda x : T.\, A)B \to_\beta A[x := B]$

(1b) (basis, second order) $(\lambda \alpha : *.\, A)T \to_\beta A[\alpha := T]$

(2) (compatibility) As in $\lambda \to$.

Example 3.13: $(\lambda \alpha : *.\, \lambda f : \alpha \to \alpha.\, \lambda x : \alpha.\, f(f\,x))\, nat\ suc\ two$

$\to_\beta (\lambda f : nat \to nat.\, \lambda x : nat.\, f(f\,x))\, suc\ two$

$\to_\beta (\lambda x : nat.\, suc(suc\ x))\, two$

$\to_\beta suc(suc\ two))$

General properties of $\lambda 2$

Remark 3.13: The following lemmas and theorems still hold in $\lambda 2$:

- Free Variable Lemma
- Thinning Lemma
- Condensing Lemma
- Permutation Lemma (if the permuted context is still a valid $\lambda 2$-context)
- Generation Lemma
- Subterm Lemma
- Uniqueness of Types Lemma
- Substitution Lemma
- Church-Rosser Theorem
- Subject Reduction Lemma
- Strong Normalization Theorem.

Type inference is no longer decidable in $\lambda 2$.

Exercises 8

8.1: Find Γ, T such that $\Gamma \vdash \lambda \alpha, \beta, \gamma : *. \lambda f : \alpha \to \beta. \lambda g : \beta \to \gamma. \lambda x : \alpha. g(fx) : T$ and give its complete derivation in flag format.

(a) $\boxed{\alpha : *}$

(b) $\boxed{\beta : *}$

(c) $\boxed{\gamma : *}$

(d) $\boxed{f : \alpha \to \beta}$

(e) $\boxed{g : \beta \to \gamma}$

(f) $\boxed{x : \alpha}$

(s) $fx : \beta$ (appl) on (d), (f)

(t) $g(fx) : \gamma$ (appl) on (e), (s)

(u) $\lambda x : \alpha. g(fx) : \alpha \to \gamma$ (abst) on (t)

(v) $\lambda g : \beta \to \gamma. \lambda x : \alpha. g(fx) : (\beta \to \gamma) \to \alpha \to \gamma$ (abst) on (u)

(w) $\lambda f : \alpha \to \beta. \lambda g : \beta \to \gamma. \lambda x : \alpha. g(fx) : (\beta \to \gamma) \to (\beta \to \gamma) \to \alpha \to \gamma$ (abst) on (v)

(x) $\lambda \gamma : *. \lambda f : \alpha \to \beta. \lambda g : \beta \to \gamma. \lambda x : \alpha. g(fx) :$ (abst2) on (w)
$\Pi \gamma : *. (\beta \to \gamma) \to (\beta \to \gamma) \to \alpha \to \gamma$

(y) $\lambda \beta : *. \lambda \gamma : *. \lambda f : \alpha \to \beta. \lambda g : \beta \to \gamma. \lambda x : \alpha. g(fx) :$ (abst2) on (x)
$\Pi \beta : *. \Pi \gamma : *. (\beta \to \gamma) \to (\beta \to \gamma) \to \alpha \to \gamma$

(z) $\lambda \alpha : *. \lambda \beta : *. \lambda \gamma : *. \lambda f : \alpha \to \beta. \lambda g : \beta \to \gamma. \lambda x : \alpha. g(fx) :$ (abst2) on (y)
$\Pi \alpha : *. \Pi \beta : *. \Pi \gamma : *. (\beta \to \gamma) \to (\beta \to \gamma) \to \alpha \to \gamma$

Short version:

(a) $\boxed{\alpha : *, \beta : *, \gamma : *, f : \alpha \to \beta, g : \beta \to \gamma, x : \alpha}$

(x) $fx : \beta$

(y) $g(fx) : \gamma$

(z) $\lambda \alpha : *. \lambda \beta : *. \lambda \gamma : *. \lambda f : \alpha \to \beta. \lambda g : \beta \to \gamma. \lambda x : \alpha. g(fx) :$
$\Pi \alpha : *. \Pi \beta : *. \Pi \gamma : *. (\beta \to \gamma) \to (\beta \to \gamma) \to \alpha \to \gamma$

Type Theory, 47

Freitag, 16. Dezember 2022 12:37

8.2: Find T such that $nat : *, bool : * \vdash (\lambda \alpha, \beta : * . \lambda f : \alpha \to \alpha . \lambda g : \alpha \to \beta . \lambda x : \alpha . g(f(fx)))\ nat\ bool : T$ and give its complete derivation in flag format.

(a) $\boxed{nat : *, bool : *}$

(b) $\boxed{\alpha : *, \beta : *, f : \alpha \to \alpha, g : \alpha \to \beta, x : \alpha}$

(u) $fx : \alpha$

(v) $f(fx) : \alpha$

(w) $g(f(fx)) : \beta$

(x) $\lambda \alpha : * . \lambda \beta : * . \lambda f : \alpha \to \alpha . \lambda g : \alpha \to \beta . \lambda x : \alpha . g(f(fx)) : \Pi \alpha : * . \Pi \beta : * . (\alpha \to \alpha) \to (\alpha \to \beta) \to \alpha \to \beta$

(y) $(\lambda \alpha : * . \lambda \beta : * . \lambda f : \alpha \to \alpha . \lambda g : \alpha \to \beta . \lambda x : \alpha . g(f(fx)))\ nat :$
$\Pi \beta : * . (nat \to nat) \to (nat \to \beta) \to nat \to \beta$

(z) $(\lambda \alpha : * . \lambda \beta : * . \lambda f : \alpha \to \alpha . \lambda g : \alpha \to \beta . \lambda x : \alpha . g(f(fx)))\ nat\ bool :$
$(nat \to nat) \to (nat \to bool) \to nat \to bool$

8.3: Find A such that $nat : * \vdash A : \Pi \alpha, \beta : * . (nat \to \alpha) \to (\alpha \to nat \to \beta) \to nat \to \beta$ and give its complete derivation in flag format.

(a) $\boxed{nat : *}$

(b) $\boxed{\alpha : *, \beta : *, f : nat \to \alpha, g : \alpha \to nat \to \beta, x : nat}$

(w) $fx : \alpha$

(x) $g(fx) : nat \to \beta$

(y) $g(fx)x : \beta$

(z) $\lambda \alpha : * . \lambda \beta : * . \lambda f : nat \to \alpha . \lambda g : \alpha \to nat \to \beta . \lambda x : nat . g(fx)x :$
$\Pi \alpha : * . \Pi \beta : * . (nat \to \alpha) \to (\alpha \to nat \to \beta) \to nat \to \beta$

Type Theory, 48

Dienstag, 10. Jänner 2023 14:41

8.4: Find A such that $() \vdash A : \Pi \alpha, \beta, \gamma : *. (\alpha \to (\beta \to \alpha) \to \gamma) \to \alpha \to \gamma$ and give its complete derivation in flag format.

(a) $\boxed{\alpha : *, \beta : *, \gamma : *, f : \alpha \to (\beta \to \alpha) \to \gamma, x : \alpha}$

(b) $\quad \boxed{y : \beta}$

(v) $\quad x : \alpha$

(w) $\lambda y : \beta . x : \beta \to \alpha$

(x) $f x : (\beta \to \alpha) \to \gamma$

(y) $f x (\lambda y : \beta . x) : \gamma$

(z) $\lambda \alpha : * . \lambda \beta : * . \lambda \gamma : * . \lambda f : \alpha \to (\beta \to \alpha) \to \gamma . \lambda x : \alpha . f x (\lambda y : \beta . x) :$

$\qquad \Pi \alpha : * . \Pi \beta : * . \Pi \gamma : * . (\alpha \to (\beta \to \alpha) \to \gamma) \to \alpha \to \gamma$

8.5: Let $nat \equiv \Pi \alpha : * . (\alpha \to \alpha) \to \alpha \to \alpha$, $zero \equiv \lambda \alpha : * . \lambda f : \alpha \to \alpha . \lambda x : \alpha . x$,

$one \equiv \lambda \alpha : * . \lambda f : \alpha \to \alpha . \lambda x : \alpha . f x$, $suc \equiv \lambda n : nat . \lambda \beta : * . \lambda f : \beta \to \beta . \lambda x : \beta . f (n \beta f x)$

and show that $() \vdash zero : nat$, $() \vdash suc : nat \to nat$, $suc\ zero =_\beta one$.

(a) $\boxed{\alpha : *, f : \alpha \to \alpha, x : \alpha}$

(y) $x : \alpha$

(z) $zero : nat$

(a) $\boxed{n : nat, \beta : *, f : \beta \to \beta, x : \beta}$

(v) $n \beta : (\beta \to \beta) \to \beta \to \beta$

(w) $n \beta f : \beta \to \beta$

(x) $n \beta f x : \beta$

(y) $f (n \beta f x) : \beta$

(z) $suc : nat \to nat$

$suc\ zero \equiv (\lambda n : nat . \lambda \beta : * . \lambda f : \beta \to \beta . \lambda x : \beta . f (n \beta f x))(\lambda \alpha : * . \lambda f : \alpha \to \alpha . \lambda x : \alpha . x)$

$\to_\beta \lambda \beta : * . \lambda f : \beta \to \beta . \lambda x : \beta . f ((\lambda \alpha : * . \lambda f : \alpha \to \alpha . \lambda x : \alpha . x) \beta f x)$

$\twoheadrightarrow_\beta \lambda \beta : * . \lambda f : \beta \to \beta . \lambda x : \beta . f x \equiv one$

8.6: Let $\perp \equiv \Pi\alpha:*.\alpha$ and show that in the context $x:\perp, \beta:*$, β is inhabitated. Interpret this observation in the context of the Curry-Howard-isomorphism.

(a) $\boxed{x:\perp, \beta:*}$
(z) $x\beta:\beta$

If $\perp$ is inhabitated then every proposition is true, so $\perp$ defines "contradiction".

8.7: Let $\text{bool} \equiv \Pi\alpha:*.\alpha\to\alpha\to\alpha$, $\text{true} \equiv \lambda\alpha:*.\lambda x,y:\alpha.x$, $\text{false} \equiv \lambda\alpha:*.\lambda x,y:\alpha.y$, show that $() \vdash \text{true}:\text{bool}$, $() \vdash \text{false}:\text{bool}$, and find $\text{not}:\text{bool}\to\text{bool}$ such that $\text{not true} \twoheadrightarrow_\beta \text{false}$ and $\text{not false} \twoheadrightarrow_\beta \text{true}$ (hint: cf. Exercise 3.3).

(a) $\boxed{\alpha:*, x:\alpha, y:\alpha}$
(y) $x:\alpha$
(z) $\text{true}:\text{bool}$

(a) $\boxed{\alpha:*, x:\alpha, y:\alpha}$
(y) $y:\alpha$
(z) $\text{false}:\text{bool}$

$\text{not} \equiv \lambda z:\text{bool}.z\ \text{bool false true}$

$$
\begin{aligned}
\text{not true} &\equiv (\lambda z:\text{bool}.z\ \text{bool false true})\ \text{true} \\
&\to_\beta \text{true bool false true} \\
&\equiv (\lambda\alpha:*.\lambda x,y:\alpha.x)\ \text{bool false true} \\
&\twoheadrightarrow_\beta \text{false}
\end{aligned}
$$

$$
\begin{aligned}
\text{not false} &\equiv (\lambda z:\text{bool}.z\ \text{bool false true})\ \text{false} \\
&\to_\beta \text{false bool false true} \\
&\equiv (\lambda\alpha:*.\lambda x,y:\alpha.y)\ \text{bool false true} \\
&\twoheadrightarrow_\beta \text{true}
\end{aligned}
$$

4. Types depending on types: the system $\lambda\underline{\omega}$

In $\lambda 2$: abstracting terms from types, $\lambda x:\sigma.x \to \lambda\alpha:*.\lambda x:\alpha.x$.

In $\lambda\underline{\omega}$: abstracting types from types, $\alpha\to\alpha$, $\beta\to\beta$, $(\gamma\to\delta)\to\gamma\to\delta$ are all of the form $\diamond\to\diamond$.

Introduce type constructors: $\lambda\alpha:*.\alpha\to\alpha$ with $(\lambda\alpha:*.\alpha\to\alpha)\beta \to_\beta \beta\to\beta$.

The types of type constructors

What are the types of $\lambda\alpha:*.\alpha\to\alpha$ and $\lambda\alpha:*.\lambda\beta:*.\alpha\to\beta$?

Educated guess: $\lambda\alpha:*.\alpha\to\alpha : *\to*$ and $\lambda\alpha:*.\lambda\beta:*.\alpha\to\beta : *\to*\to*$.

Definition 4.1: kinds, $\mathbb{K}$, $\square$, sorts, s, (proper) type constructors

we use the same conventions as for arrow types

- $\mathbb{K} = * \mid (\mathbb{K}\to\mathbb{K})$ denotes the set of all kinds ("super types").
- $\square$ denotes the type of all kinds (the only "super super type").
- $*$ and $\square$ are called sorts and we take s to either stand for $*$ or $\square$.
- If $B:\square$ and $A:B$ (and $B\neq *$) then A is called a (proper) type constructor.

Example 4.2:
- $*$, $*\to*$, $*\to*\to*$, $(*\to*)\to* \in \mathbb{K}$.
- We have 4 levels of statements and judgements:

  Level 1 (terms): $x:\alpha$, $\lambda x:\alpha.x : \alpha\to\alpha$.

  Level 2 (constructors): $\alpha:*$, $\alpha\to\beta:*$, $\lambda\alpha:*.\alpha\to\alpha : *\to*$.

  Level 3 (kinds): $*:\square$, $*\to*:\square$, $*\to*\to*:\square$, $(*\to*)\to*:\square$.

  Level 4 (super super type): $\square$.

  We may form a "judgement chain": $x:\alpha:*\to*:\square$.

The system $\lambda\underline{\omega}$

Definition 4.3: $\Lambda_{\underline{\omega}}$

$\Lambda_{\underline{\omega}} = \mathbb{V} \mid (\Lambda_{\underline{\omega}}\,\Lambda_{\underline{\omega}}) \mid (\lambda\mathbb{V}:*.\Lambda_{\underline{\omega}})$ ... the set of type constructors.

Remark 4.4: $V = \{a, b, c, \dots\}$, $\mathbb{V} = \{\alpha, \beta, \gamma, \dots\}$, $\mathbb{T} = \mathbb{V} \mid (\mathbb{T}\to\mathbb{T})$, and $\Lambda_{\mathbb{T}} = V \mid (\Lambda_{\mathbb{T}}\Lambda_{\mathbb{T}}) \mid (\lambda V:\mathbb{T}.\Lambda_{\mathbb{T}})$ are as in $\lambda\to$.

Derivation rules for $\lambda\underline{\omega}$

Definition 4.5:

two rules in one: $s = *$ or $s = \square$

(silent) (sort) $\quad () \vdash * : \square$

(silent) (var) $\quad \dfrac{\Gamma \vdash A : s}{\Gamma, X : A \vdash X : A}$ if $X \notin \mathrm{dom}(\Gamma)$

(silent) (weak) $\quad \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, X : C \vdash A : B}$ if $X \notin \mathrm{dom}(\Gamma)$

(silent) (form) $\quad \dfrac{\Gamma \vdash A : s \qquad \Gamma \vdash B : s}{\Gamma \vdash A \to B : s}$

(app) $\quad \dfrac{\Gamma \vdash A : S \to T \qquad \Gamma \vdash B : S}{\Gamma \vdash AB : T}$

(silent)

(abst) $\quad \dfrac{\Gamma, X : S \vdash A : T \qquad \Gamma \vdash S \to T : s}{\Gamma \vdash \lambda X : S . A : S \to T}$

(conv) $\quad \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}$ if $B =_\beta B'$

(silent)

Remark 4.6:

·) (var) allows for two things:
- extending a context
- deriving the last declaration in a context as a statement.

This way we avoid the need to define valid contexts as in $\lambda 2$. (var) is less general than in $\lambda\to$ and $\lambda 2$ (last declaration only).

·) (weak) allows to weaken a context by appending new declarations. Without it we couldn't derive, e.g., $\mathcal{L} : *, \beta : * \vdash \mathcal{L} : *$ or even $\mathcal{L} : *, \beta : * \vdash \beta : *$ (since we couldn't obtain $\mathcal{L} : * \vdash * : \square$).

·) Unlike Π-types in the (app2) rule in $\lambda 2$ we leave redexes involving type constructors unevaluated (c.f. (app) rule if $S \to T$ is a kind). For that reason we introduce the conversion rule (conv) to conclude, say, $\Gamma \vdash x : \beta \to \beta$ from $\Gamma \vdash x : (\lambda \mathcal{L} : * . \mathcal{L} \to \mathcal{L})\beta$ and $\Gamma \vdash \beta \to \beta : *$. The second premiss of (conv) is necessary because, e.g., for all A one has $\beta \to \gamma =_\beta (\lambda \mathcal{L} : * . \beta \to \gamma) A$ even if A is not well-formed.

Type Theory, 52

Donnerstag, 12. Jänner 2023 12:26

Example 4.7:

(1)	$* : \square$	(sort)
	$\vert\ \boxed{\alpha : *}$	
(2)	$\vert\ \alpha : *$	(var) on (1)
	$\vert\ \vert\ \boxed{x : \alpha}$	
(3)	$\vert\ \vert\ x : \alpha$	(var) on (2)
(4)	$\vert\ \vert\ \alpha : *$	(weak) on (2),(2)
(5)	$\vert\ * : \square$	(weak) on (1),(1)
	$\vert\ \vert\ \boxed{\beta : *}$	
(6)	$\vert\ \vert\ \alpha : *$	(weak) on (2),(5)
(7)	$\vert\ \vert\ \beta : *$	(var) on (5)
(8)	$\vert\ \vert\ \alpha \to \beta : *$	(form) on (6),(7)
(9)	$\vert\ * \to * : \square$	(form) on (5),(5)
	$\vert\ \boxed{\beta : *}$	
(10)	$\vert\ * : \square$	(weak) on (1),(1)
	$\vert\ \vert\ \boxed{\alpha : *}$	
(11)	$\vert\ \vert\ \alpha : *$	(var) on (10)
(12)	$\vert\ \vert\ \alpha \to \alpha : *$	(form) on (11),(11)
(13)	$\vert\ * \to * : \square$	(form) on (10),(10)
(14)	$\vert\ \lambda \alpha : * . \alpha \to \alpha : * \to *$	(abst) on (12),(13)
(15)	$\vert\ \beta : *$	(var) on (1)
(16)	$\vert\ (\lambda \alpha : * . \alpha \to \alpha)\beta : *$	(app) on (14),(15)
(17)	$\vert\ \beta \to \beta : *$	(form) on (15),(15)
	$\vert\ \vert\ \boxed{x : (\lambda \alpha : * . \alpha \to \alpha)\beta}$	
(18)	$\vert\ \vert\ x : (\lambda \alpha : * . \alpha \to \alpha)\beta$	(var) on (16)
(19)	$\vert\ \vert\ \beta \to \beta : *$	(weak) on (17),(16)
(20)	$\vert\ \vert\ x : \beta \to \beta$	(conv) on (18),(19)

Shortened derivation:

(a)	$\boxed{\beta : *}$
(b)	$\vert\ \boxed{\alpha : *}$
(12)	$\vert\ \vert\ \alpha \to \alpha : *$
(14)	$\vert\ \lambda \alpha : * . \alpha \to \alpha : * \to *$
(16)	$\vert\ (\lambda \alpha : * . \alpha \to \alpha)\beta : *$
(c)	$\vert\ \boxed{x : (\lambda \alpha : * . \alpha \to \alpha)\beta}$
(18)	$\vert\ \vert\ x : (\lambda \alpha : * . \alpha \to \alpha)\beta$
(20)	$\vert\ \vert\ x : \beta \to \beta$

Type Theory, 53

Donnerstag, 12. Jänner 2023 14:48

General properties of $\lambda\underline{\omega}$

Remark 4.8: The following lemmas and theorems still hold in $\lambda\underline{\omega}$:

- ·) Free Variable Lemma
- ·) Thinning Lemma
- ·) Condensing Lemma
- ·) Permutation Lemma ?
- ·) Generation Lemma (apart from (conv))
- ·) Subterm Lemma
- ·) Uniqueness of Types Lemma (due to (conv)): $\Gamma \vdash A : S,\ \Gamma \vdash A : T \Rightarrow S =_\beta T$) (instead of $S \equiv T$)
- ·) Substitution Lemma
- ·) Church-Rosser Theorem
- ·) Subject Reduction Lemma
- ·) Strong Normalization Theorem.

Lemma	(conv) rule
$\Gamma \vdash A : B$	$\Gamma \vdash A : B$
$\downarrow_\beta$	$\downarrow_\beta$
$\Gamma \vdash A' : B$	$\Gamma \vdash A : B'$

Exercises 9

9.1: Give complete derivations of $() \vdash (* \to *) \to * : \square$ and $\mathcal{L} : *, \mathcal{B} : * \vdash (\mathcal{L} \to \mathcal{B}) \to \mathcal{L} : *$.

(1)	$* : \square$	(sort)
(2)	$* \to * : \square$	(form) on (1),(1)
(3)	$(* \to *) \to * : \square$	(form) on (2),(1)
	$\boxed{\mathcal{L} : *}$	
(4)	$\mathcal{L} : *$	(var) on (1)
(5)	$* : \square$	(weak) on (1),(1)
	$\boxed{\mathcal{B} : *}$	
(6)	$\mathcal{L} : *$	(weak) on (4),(5)
(7)	$\mathcal{B} : *$	(var) on (5)
(8)	$\mathcal{L} \to \mathcal{B} : *$	(form) on (6),(7)
(9)	$(\mathcal{L} \to \mathcal{B}) \to \mathcal{L} : *$	(form) on (8),(6)

Type Theory, 54

Donnerstag, 12. Jänner 2023 16:57

9.2: Give shortened derivations of $\mathcal{L}, \beta : *, x : \mathcal{L}, y : \mathcal{L} \to \beta, z : \beta \to \mathcal{L} \vdash z(yx) : \mathcal{L}$ and $\mathcal{L} : *, \beta : * \to *, x : \beta(\beta\mathcal{L}) \vdash \lambda y : \mathcal{L}. x : \mathcal{L} \to \beta(\beta\mathcal{L})$.

(a) $\boxed{\mathcal{L} : *, \beta : *, x : \mathcal{L}, y : \mathcal{L} \to \beta, z : \beta \to \mathcal{L}}$

(1) $yx : \beta$

(2) $z(yx) : \mathcal{L}$

(a) $\boxed{\mathcal{L} : *, \beta : * \to *, x : \beta(\beta\mathcal{L})}$

(b) $\boxed{y : \mathcal{L}}$

(1) $\beta\mathcal{L} : *$

(2) $\beta(\beta\mathcal{L}) : *$

(3) $x : \beta(\beta\mathcal{L})$

(4) $\lambda y : \mathcal{L}. x : \mathcal{L} \to \beta(\beta\mathcal{L})$

9.3: Give a shortened derivation of $\mathcal{L} : *, x : \mathcal{L} \vdash \lambda y : \mathcal{L}. x : (\lambda \beta : *. \beta \to \beta)\mathcal{L}$.

(a) $\boxed{\mathcal{L} : *, x : \mathcal{L}}$

(b) $\boxed{y : \mathcal{L}}$

(1) $x : \mathcal{L}$

(2) $\lambda y : \mathcal{L}. x : \mathcal{L} \to \mathcal{L}$

(3) $\lambda y : \mathcal{L}. x : (\lambda \beta : *. \beta \to \beta)\mathcal{L}$

Type Theory, 55

Dienstag, 17. Jänner 2023 16:12

"predicate"

5. Types depending on terms: the system λP

In $\lambda \underline{\omega}$: type constructors depending on types, $\lambda \alpha : *. \alpha \to \alpha$.

In λP: type constructors depending on terms, $\lambda x : T. A$ where A is a type.

Application in logic: sets and propositions.

(1) Sets: Let S_n be a set for all $n : nat$ (may think of sets as types).

Then $\lambda n : nat. S_n$ is a type (type constructor) depending on the term n.

$\lambda n : nat. S_n$ maps n to a set S_n (set-valued function).

Other interpretations: $\lambda n : nat. S_n$ is a "family of types" or an "indexed type".

(2) Propositions: Let P_n be a proposition for all $n : nat$ (i.e. P_n is a type by PAT).

Then $\lambda n : nat. P_n$ is again a type (type constructor) depending on the term n.

$\lambda n : nat. P_n$ maps n to a proposition P_n on n, i.e. $\lambda n : nat. P_n$ is a predicate.

Example 5.1:

(1a) If $S_n = \{0, n, 2n, \dots\}$, then $(\lambda n : nat. S_n) 0 \to_\beta \{0\}$, $(\lambda n : nat. S_n) 1 \to_\beta \{0, 1, 2, \dots\}, \dots$

What is the type of $\lambda n : nat. S_n$? Should be $nat \to *$.

(1b) If $V_n = \{(v_1, \dots, v_n) \mid v_i \in \mathbb{N}\}$, then $(\lambda n : nat. S_n) 5$ reduces to the set of all sequences of natural numbers of length 5. Again its type should be $nat \to *$.

(2) If $P_n \dots n$ is prime, then $(\lambda n : nat. P_n) 3 \to_\beta true$, $(\lambda n : nat. P_n) 4 \to_\beta false, \dots$ and $\lambda n : nat. P_n$ is the predicate "to be prime", again with potential type $nat \to *$.

Derivation rules for λP

Definition 5.2: Comparison with $\lambda\underline{\omega}$:

(silent) (sort) $() \vdash * : \square$ — two rules in one: $s = *$ or $s = \square$ — identical

(silent) (var) $$\frac{\Gamma \vdash A : s}{\Gamma, X : A \vdash X : A} \quad \text{if } X \notin \mathrm{dom}(\Gamma)$$ identical

(silent) (weak) $$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, X : C \vdash A : B} \quad \text{if } X \notin \mathrm{dom}(\Gamma)$$ identical

(silent) (form) $$\frac{\Gamma \vdash A : * \qquad \Gamma, X : A \vdash B : s}{\Gamma \vdash \Pi X : A . B : s} \qquad \frac{\Gamma \vdash A : s \qquad \Gamma \vdash B : s}{\Gamma \vdash A \to B : s}$$

(app) $$\frac{\Gamma \vdash A : \Pi X : S . T \qquad \Gamma \vdash B : S}{\Gamma \vdash AB : T[X := B]} \text{ (silent)} \qquad \frac{\Gamma \vdash A : S \to T \qquad \Gamma \vdash B : S}{\Gamma \vdash AB : T} \text{ (silent)}$$

(abst) $$\frac{\Gamma, X : S \vdash A : T \qquad \Gamma \vdash \Pi X : S . T : s}{\Gamma \vdash \lambda X : S . A : \Pi X : S . T} \qquad \frac{\Gamma, X : S \vdash A : T \qquad \Gamma \vdash S \to T : s}{\Gamma \vdash \lambda X : S . A : S \to T}$$

(conv) $$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \quad \text{if } B =_\beta B'$$ (silent) — identical

Remark 5.3: Differences between λP and $\lambda\underline{\omega}$:

- Upgrade of $\to$-types to Π-types in (form), (appl), (abst).
- Downgrade of implied types in (form) ($s = *$ so A must be a term, no $\lambda \mathcal{L} : * . \mathcal{L}$ in λP).
  Double role of (form):
  $s = *$: $A : *$, $B : *$, so $\Pi A : * . B : *$ (e.g. $\Pi n : nat . nat : *$ with inhabitant $\lambda n : nat . f n$).
  $s = \square$: $A : *$, $B : \square$, so $\Pi A : * . B : \square$ (e.g. $\Pi n : nat . * : \square$ with inhabitant $\lambda n : nat . P n$).
- Extend context of second premiss of (form).
- Automatically reduce output type in (appl) ($T[x := B]$).
- There are no $\to$-types in λP but we write $A \to B$ for $\Pi x : A . B$ if $x \notin FV(B)$.
- λP has all the "nice properties" of $\lambda\underline{\omega}$.

Remark 5.4: The formation rule is also called "product rule" because a Π-type is considered a Cartesian product of a family of types. If A is a finite type consisting of two elements a_1, a_2, then "$\Pi x : A . B = B[x := a_1] \times B[x := a_2]$". Π-types are thus a generalization of the Cartesian product and of the space of functions $A \to B$.

Example 5.5:

(1) $* : \square$ (sort)

$\boxed{A : *}$

(2) $A : *$ (var) on (1)

(3) $* : \square$ (weak) on (1),(1)

$\boxed{x : A}$

(4) $* : \square$ (weak) on (3),(2)

(5) $A \to * : \square$ (form) on (2),(4)

$= \Pi x : A . *$, kind depending on term, not derivable in $\lambda\underline{\omega}$

$\boxed{P : A \to *}$

assuming inhabitant of $A \to *$, type depending on term

(6) $P : A \to *$ (var) on (5)

(7) $A : *$ (weak) on (2),(5)

(8) $* : \square$ (weak) on (3),(5)

$\boxed{x : A}$

(9) $x : A$ (var) on (7)

(10) $P : A \to *$ (weak) on (6),(7)

(11) $Px : *$ (app) on (10),(9)

"real" dependent type

(12) $\Pi x : A . Px : *$ (form) on (7),(11)

$\boxed{x : A}$

$\boxed{y : Px}$

(13) $Px : *$ (weak) on (11),(11)

(14) $Px \to Px : *$ (form) on (11),(13)

(15) $\Pi x : A . Px \to Px : *$ (form) on (7),(14)

$\boxed{x : A}$

$\boxed{y : Px}$

(16) $y : Px$ (var) on (11)

(17) $\lambda y : Px . y : Px \to Px$ (abst) on (16),(14)

(18) $\lambda x : A . \lambda y : Px . y : \Pi x : A . Px \to Px$ (abst) on (17),(15)

finding inhabitant of $\Pi x : A . Px \to Px$

Shortened derivation:

(a) $\boxed{A : *}$

(b) $\boxed{P : A \to *}$

(c) $\boxed{x : A}$

(11) $Px : *$

(d) $\boxed{y : Px}$

(16) $y : Px$

(17) $\lambda y : Px . y : Px \to Px$

(18) $\lambda x : A . \lambda y : Px . y : \Pi x : A . Px \to Px$

Type Theory, 58

Mittwoch, 18. Jänner 2023 17:30

Minimal predicate logic in λP: encoding $\Rightarrow$, $\forall$, sets, and predicates

PAT: ·) If a term p inhabits a type P (i.e. $p : P$) where P is interpreted as a proposition, then we interpret p as a proof of P and p is called a "proof object".

·) If no inhabitant of a proposition P exists, then there is no proof of P, so P must be false.

How to encode basic entities of minimal predicate logic in λP?

(1) Sets

Encode sets as types, so $S : *$.

Elements of sets are terms, so a is an element of S if $a : S$.

Examples: $nat : *$, $nat \to nat : *$, $3 : nat$, $\lambda n : nat . n : nat \to nat$.

(2) Propositions

Propositions are also encoded as types, so $P : *$.

According to PAT, a term p with $p : P$ encodes a proof of P (iff P is true).

(3) Predicates

A predicate P is a function from a set to the set of all propositions, so $P : S \to *$ is a predicate on the set S.

For each $a : S$ we then have $Pa : *$.

All of these Pa are propositions which are types, so Pa may be inhabitated:

·) If Pa is inhabitated, so $p : Pa$ for some p, P holds for a.

·) If Pa is not inhabitated, P does not hold for a.

(4) Implication

Identify $A \Rightarrow B$ with $A \to B$ ($= \Pi x : A . B$ if $x \notin FV(B)$).

Justification: ·) $A \Rightarrow B$ is true.

·) If A is true, then B is true.

·) If A is inhabitated then B is inhabitated.

·) There is a function mapping inhabitants of A to inhabitants of B.

·) There is an f with $f : A \to B$.

·) $A \to B$ is inhabitated.

Thus, the truth of $A \Rightarrow B$ is equivalent to the inhabitation of $A \to B$.

We get $\Rightarrow$-introduction and $\Rightarrow$-elimination for free!

(abst) $\leftrightarrow$ ($\Rightarrow$-intro) $\begin{array}{l} \boxed{\text{Assume } A} \\ \vdots \\ B \\ \hline A \Rightarrow B \end{array}$ (app) $\leftrightarrow$ ($\Rightarrow$-elim) $\dfrac{A \Rightarrow B \quad A}{B}$

Type Theory, 59

Mittwoch, 18. Jänner 2023 19:06

(5) **Universal quantification**

Identify $\forall x \in S : P(x)$ with $\Pi x : S . Px$.

Justification:
- ·) $\forall x \in S : P(x)$ is true.
- ·) For each x in the set S, the proposition $P(x)$ is true.
- ·) For each x in S, the type Px is inhabitated.
- ·) There is a function mapping each x in S to an inhabitant of Px.
- ·) There is an f with $f : \Pi x : S . Px$.
- ·) $\Pi x : S . Px$ is inhabitated.

Thus, the truth of $\forall x \in S : P(x)$ is equivalent to the inhabitation of $\Pi x : S . Px$.

We get $\forall$-introduction and $\forall$-elimination for free!

(abst) $\leftrightarrow$ ($\forall$-intro)
$$\frac{\begin{array}{|l}\boxed{\text{Let } x \in S}\\ \vdots \\ P(x)\end{array}}{\forall x \in S : P(x)}$$

(app) $\leftrightarrow$ ($\forall$-elim)
$$\frac{\forall x \in S : P(x) \quad N \in S}{P(N)}$$

the second premiss makes sure that $\Pi X : S . T$ is well-formed, is taken for granted in $\forall$-intro

Compare (abst)
$$\frac{\Gamma, X : S \vdash A : T \quad \Gamma \vdash \Pi X : S . T : s}{\Gamma \vdash \lambda X : S . A : \Pi X : S . T}$$

Γ part of context is usually left implicit in $\forall$-intro and $\forall$-elim

proof object of $\forall x \in S : P(x)$ $\quad$ $\forall x \in S : P(x)$

(app)
$$\frac{\Gamma \vdash A : \Pi x : S . T \quad \Gamma \vdash B : S}{\Gamma \vdash AB : T[x := B]}$$

proof object of $P(N)$ $\quad$ $P(N)$

Remark 5.6: we have no **negation**, **conjunction**, **disjunction**, or **existential quantification** in λP (need to combine several systems).

Example 5.7: PAT interpretation of Example 5.5.

(12) $A : *, P : A \to * \vdash \Pi x : A . Px : *$

If A is a set and P is a predicate on A, then $\forall x \in A : P(x)$ is a proposition.

(15) $A : *, P : A \to * \vdash \Pi x : A . Px \to Px : *$

In the same setting, $\forall x \in A : P(x) \Rightarrow P(x)$ is a proposition.

(18) $A : *, P : A \to * \vdash \lambda x : A . \lambda y : Px . y : \Pi x : A . Px \to Px$

In the same setting, $\lambda x : A . \lambda y : Px . y$ is an inhabitant / proof of $\forall x \in A : P(x) \Rightarrow P(x)$ and thus $\forall x \in A : P(x) \Rightarrow P(x)$ is true.

Type Theory, 60

Mittwoch, 18. Jänner 2023 20:54

Example 5.8: A logical derivation in λP.

Show $\forall x \in S : \forall y \in S : Q(x,y) \Rightarrow \forall u \in S : Q(u,u)$.

Natural deduction:

(a) $\boxed{\text{Assume: } \forall x \in S : \forall y \in S : Q(x,y)}$

(b) $\boxed{\text{Let } u \in S}$

(1) $\forall y \in S : Q(u,y)$ (∀-elim) on (a), (b)

(2) $Q(u,u)$ (∀-elim) on (1), (b)

(3) $\forall u \in S : Q(u,u)$ (∀-intro) on (2)

(4) $\forall x \in S : \forall y \in S : Q(x,y) \Rightarrow \forall u \in S : Q(u,u)$ (∀-intro) on (3)

λP:

(a) $\boxed{S : *}$

(b) $\boxed{Q : S \to S \to *}$

(c) $\boxed{z : \Pi x : S . \Pi y : S . Qxy}$

(d) $\boxed{u : S}$

(1) $zu : \Pi y : S . Quy$ (app) on (c), (d)

(2) $zuu : Quu$ (app) on (1), (d)

(3) $\lambda u : S . zuu : \Pi u : S .$ (abst) on (2)

(4) $\lambda z : (\Pi x : S . \Pi y : S . Qxy) . \lambda u : S . zuu :$ (abst) on (3)

$\Pi x : S . \Pi y : S . Qxy \to \Pi u : S . Quu$

Type Theory, 61

Mittwoch, 18. Jänner 2023 21:43

Exercises 10

10.1: Give a complete derivation of $S:*,\ Q:S\to S\to * \vdash \Pi x:S.\ \Pi y:S.\ Qxy : *$.

(1)	$* : \square$	(sort)
	$\boxed{S:*}$	
(2)	$S:*$	(var) on (1)
(3)	$*:\square$	(weak) on (1),(1)
	$\boxed{x:S}$	
(4)	$*:\square$	(weak) on (3),(2)
(5)	$S\to *:\square$	(form) on (2),(4)
	$\boxed{y:S}$	
(6)	$S\to *:\square$	(weak) on (5),(2)
(7)	$S\to S\to *:\square$	(form) on (2),(6)
	$\boxed{Q:S\to S\to *}$	
(8)	$Q:S\to S\to *$	(var) on (3)
(9)	$S:*$	(weak) on (2),(3)
	$\boxed{x:S}$	
(10)	$x:S$	(var) on (9)
(11)	$Q:S\to S\to *$	(weak) on (8),(9)
(12)	$S:*$	(weak) on (9),(9)
	$\boxed{y:S}$	
(13)	$y:S$	(var) on (12)
(14)	$x:S$	(weak) on (10),(12)
(15)	$Q:S\to S\to *$	(weak) on (11),(12)
(16)	$S:*$	(weak) on (12),(12)
(17)	$Qx:S\to *$	(app) on (15),(14)
(18)	$Qxy:*$	(app) on (17),(13)
(19)	$\Pi y:S.Qxy:*$	(form) on (16),(18)
(20)	$\Pi x:S.\Pi y:S.Qxy:*$	(form) on (12),(19)

10.2: Let $\Gamma \equiv S : *,\ P : S \to *,\ f : S \to S,\ g : S \to S,\ u : \Pi x : S.\,(P(fx) \to P(gx)),$
$v : \Pi x, y : S.\,((Px \to Py) \to P(fx))$ and $A \equiv \lambda x : S.\, v(fx)(gx)(ux)$.

What proposition $B : *$ is A a proof object of? Give a shortened derivation of $\Gamma \vdash A : B$.

(a) $\boxed{S : *,\ P : S \to *}$

(b) $\boxed{f : S \to S,\ g : S \to S}$

(c) $\boxed{u : \Pi x : S.\,(P(fx) \to P(gx)),\ v : \Pi x, y : S.\,((Px \to Py) \to P(fx))}$

(d) $\boxed{x : S}$

(1) $fx : S,\ gx : S$

(2) $ux : P(fx) \to P(gx)$

(3) $v(fx) : \Pi y : S.\,((P(fx) \to Py) \to P(f(fx)))$

(4) $v(fx)(gx) : (P(fx) \to P(gx)) \to P(f(fx))$

(5) $v(fx)(gx)(ux) : P(f(fx))$

(6) $\lambda x : S.\, v(fx)(gx)(ux) : \overbrace{\Pi x : S.\, P(f(fx))}^{\equiv B}$

10.3: Let S be a set, Q, R binary relations on S, f, g functions from S to S, and assume
$\forall x, y \in S : Q(x, f(y)) \Rightarrow Q(g(x), y)$, $\forall x, y \in S : Q(x, f(y)) \Rightarrow R(x, y)$, and $\forall x \in S : Q(x, f(f(x)))$.

Show $\forall x \in S : R(g(g(x)), g(x))$ in λP.

(a) $\boxed{S : *,\ Q : S \to S \to *,\ R : S \to S \to *}$

(b) $\boxed{f : S \to S,\ g : S \to S}$

(c) $\boxed{u : \Pi x, y : S.\,(Qx(fy) \to Q(gx)y),\ v : \Pi x, y : S.\,(Qx(fy) \to Rxy),\ w : \Pi x : S.\, Qx(f(fx))}$

(d) $\boxed{x : S}$

(1) $fx : S,\ gx : S,\ f(gx) : S,\ g(gx) : S$

(2) $v(g(gx))(gx) : Q(g(gx))(f(gx)) \to R(g(gx))(gx)$

(3) $u(gx)(f(gx)) : Q(gx)(f(f(gx))) \to Q(g(gx))(f(gx))$

(4) $w(gx) : Q(gx)(f(f(gx)))$

(5) $u(gx)(f(gx))(w(gx)) : Q(g(gx))(f(gx))$

(6) $v(g(gx))(gx)(u(gx)(f(gx))(w(gx))) : R(g(gx))(gx)$

(7) $\lambda x : S.\, v(g(gx))(gx)(u(gx)(f(gx))(w(gx))) : \Pi x : S.\, R(g(gx))(gx)$

6. $\lambda C = \lambda 2 + \lambda \underline{\omega} + \lambda P$: The Calculus of Constructions

We use one set of derivation rules to combine all systems:

Definition 6.1:

(sort) (silent) $\quad () \vdash * : \square$ $\quad$ two rules in one: $s = *$ or $s = \square$

(var) (silent) $\quad \dfrac{\Gamma \vdash A : s}{\Gamma, X : A \vdash X : A}$ if $X \notin \mathrm{dom}(\Gamma)$

(weak) (silent) $\quad \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, X : C \vdash A : B}$ if $X \notin \mathrm{dom}(\Gamma)$

four rules in one

only difference to λP: s_1 instead of $*$

(form) (silent) $\quad \dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, X : A \vdash B : s_2}{\Gamma \vdash \Pi X : A . B : s_2}$ $\leftarrow = A \to B$ if $X \notin FV(B)$

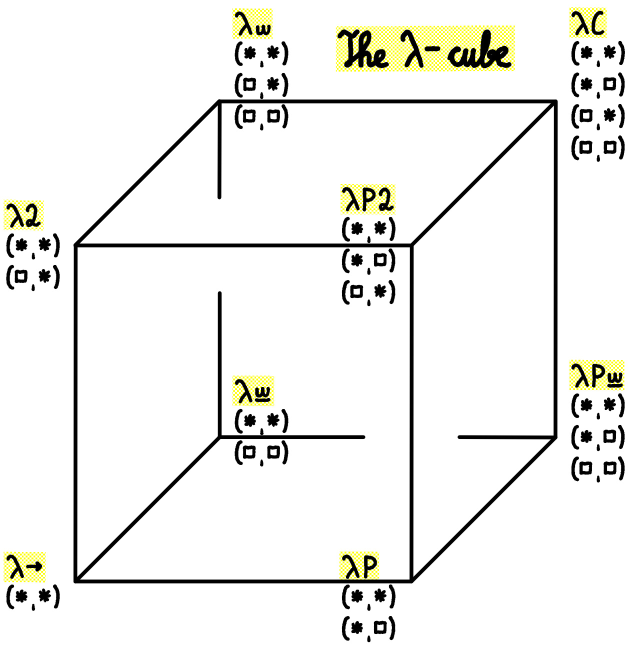
(app) $\quad \dfrac{\Gamma \vdash A : \Pi X : S . T \qquad \Gamma \vdash B : S}{\Gamma \vdash AB : T[X := B]}$

(silent)

(abst) $\quad \dfrac{\Gamma, X : S \vdash A : T \qquad \Gamma \vdash \Pi X : S . T : s}{\Gamma \vdash \lambda X : S . A : \Pi X : S . T}$

(conv) $\quad \dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}$ if $B =_\beta B'$ $\quad$ (silent)

Remark 6.2: The allowed combinations of s_1 and s_2 define which system we are in.



$(*,*)$, term depending on term:

$() \vdash \lambda x : \mathcal{L} . x : \mathcal{L} \to \mathcal{L}$

$(*,\square)$, type depending on term:

$A : *, P : A \to * \vdash \Pi x : A . Px : *$

$(\square,*)$, term depending on type:

$() \vdash \lambda \mathcal{L} : * . \lambda f : \mathcal{L} \to \mathcal{L} . \lambda x : \mathcal{L} . fx :$
$\Pi \mathcal{L} : * . (\mathcal{L} \to \mathcal{L}) \to \mathcal{L} \to \mathcal{L}$

$(\square,\square)$, type depending on type:

$() \vdash \lambda \mathcal{L} : * . \mathcal{L} \to \mathcal{L} : * \to *$

In λC the boundaries between terms and types are blurred.

Definition 6.3: the set $\mathcal{E}$ of λC-epressions

$$\mathcal{E} = V \mid * \mid \square \mid (\mathcal{E}\mathcal{E}) \mid (\lambda V : \mathcal{E}.\mathcal{E}) \mid (\Pi V : \mathcal{E}.\mathcal{E}).$$

Remark 6.4: The following lemmas and theorems still hold in λC:

- ·) Free Variable Lemma
- ·) Thinning Lemma
- ·) Condensing Lemma
- ·) Permutation Lemma ?
- ·) Generation Lemma (apart from (conv))
- ·) Subterm Lemma
- ·) Uniqueness of Types Lemma (due to (conv): $\Gamma \vdash A : S, \Gamma \vdash A : T \Rightarrow S =_\beta T$) instead of $S \equiv T$
- ·) Substitution Lemma
- ·) Church-Rosser Theorem
- ·) Subject Reduction Lemma
- ·) Strong Normalization Theorem
- ·) Decidability of well-typedness and type checking.

Logic in λC: encoding $\bot$, $\neg$, $\wedge$, $\vee$, and $\exists$

(1)-(5): as in λP (Sets, Propositions, Predicates, Implication, Universal quantification).

(6) Absurdity

lives in $\lambda 2$, i.e. derivation of $() \vdash \bot : *$ possible in $\lambda 2$ (c.f. Exercise 8.6)

Set $\bot \equiv \Pi \mathcal{L} : *. \mathcal{L}$.

Justification: ·) If $\bot$ is true then all propositions are true.

·) If $\bot$ is inhabitated then all propositions are inhabitated.

·) If $\bot$ is inhabitated then there is a function mapping an arbitrary proposition $\mathcal{L}$ to an inhabitant of $\mathcal{L}$. Such a function has type $\bot$.

Thus, absurdity is equivalent to the inhabitation of $\bot$.

We get $\bot$-introduction and $\bot$-elimination for free!

$$(app) \leftrightarrow (\bot\text{-intro}) \ \frac{A \quad \neg A}{\bot} \qquad (app) \leftrightarrow (\bot\text{-elim}) \ \frac{\bot}{A}$$

defined next

Type Theory, 65

Montag, 23. Jänner 2023 17:54

(7) Negation

Set $\neg \equiv \Pi A:*.\, A \to \bot$.

We get $\neg$-introduction and $\neg$-elimination for free!

$(abst) \leftrightarrow (\neg\text{-intro})$ $\dfrac{\begin{array}{|l}\boxed{\text{Assume } A}\\ \vdots \\ \bot\end{array}}{\neg A}$ $(app) \leftrightarrow (\neg\text{-elim})$ $\dfrac{\neg A \quad A}{\bot}$

(8) Conjunction

If A and B together imply C, then C holds on its own, i.e. that A and B hold is redundant.
Second order encoding of $\wedge$.
First order: $\wedge \equiv \Pi A:*.\Pi B:*.\neg(A \to \neg B)$, only works in classical logic.

Set $\wedge \equiv \Pi A:*.\Pi B:*.\Pi C:*.(A \to B \to C) \to C$ and write $A \wedge B$ for $\wedge AB$.

We get $\wedge$-introduction and $\wedge$-elimination for free!

$(\wedge\text{-intro})$ $\dfrac{A \quad B}{A \wedge B}$ $(\wedge\text{-elim-}l)$ $\dfrac{A \wedge B}{A}$ $(\wedge\text{-elim-}r)$ $\dfrac{A \wedge B}{B}$

$(\wedge\text{-intro-sec})$ $\dfrac{A \quad B}{\forall C:(A \Rightarrow B \Rightarrow C) \Rightarrow C}$ $\leftrightarrow$ $(\wedge\text{-intro-sec-}M)$ $\dfrac{\Gamma \vdash u:A \quad \Gamma \vdash v:B}{\Gamma \vdash \lambda C:*.\lambda x:A \to B \to C.\, xuv : \Pi C:*.(A \to B \to C) \to C}$

$(\wedge\text{-elim-sec-}l)$ $\dfrac{\forall C:(A \Rightarrow B \Rightarrow C) \Rightarrow C}{A}$ $\leftrightarrow$ $(\wedge\text{-elim-sec-}l\text{-}M)$ $\dfrac{\Gamma \vdash u:\Pi C:*.(A \to B \to C) \to C}{\Gamma \vdash uA(\lambda x:A.\lambda y:B.x):A}$

$(\wedge\text{-elim-sec-}r)$ $\dfrac{\forall C:(A \Rightarrow B \Rightarrow C) \Rightarrow C}{B}$ $\leftrightarrow$ $(\wedge\text{-elim-sec-}r\text{-}M)$ $\dfrac{\Gamma \vdash u:\Pi C:*.(A \to B \to C) \to C}{\Gamma \vdash uB(\lambda x:A.\lambda y:B.y):B}$

(9) Disjunction

If A or B implies C, then C holds on its own, i.e. that A or B holds is redundant.
Second order encoding of $\vee$. First order: $\vee \equiv \Pi A:*.\Pi B:*.\neg A \to B$, only works in classical logic.

Set $\vee \equiv \Pi A:*.\Pi B:*.\Pi C:*.(A \to C) \to (B \to C) \to C$ and write $A \vee B$ for $\vee AB$.

We get $\vee$-introduction and $\vee$-elimination for free!

$(\vee\text{-intro-}l)$ $\dfrac{A}{A \vee B}$ $(\vee\text{-intro-}r)$ $\dfrac{B}{A \vee B}$ $(\vee\text{-elim})$ $\dfrac{A \vee B \quad A \Rightarrow C \quad B \Rightarrow C}{C}$

$(\vee\text{-intro-sec-}l)$ $\dfrac{A}{\forall C:(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C}$

$\leftrightarrow$ $(\vee\text{-intro-sec-}l\text{-}M)$ $\dfrac{\Gamma \vdash u:A}{\Gamma \vdash \lambda C:*.\lambda x:A \to C.\lambda y:B \to C.\, xu : \Pi C:*.(A \to C) \to (B \to C) \to C}$

$(\vee\text{-intro-sec-}r)$ $\dfrac{B}{\forall C:(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C}$

$\leftrightarrow$ $(\vee\text{-intro-sec-}r\text{-}M)$ $\dfrac{\Gamma \vdash u:B}{\Gamma \vdash \lambda C:*.\lambda x:A \to C.\lambda y:B \to C.\, yu : \Pi C:*.(A \to C) \to (B \to C) \to C}$

$(\vee\text{-elim-sec})$ $\dfrac{\forall C:(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C \quad A \Rightarrow C \quad B \Rightarrow C}{C}$

$\leftrightarrow$ $(\vee\text{-elim-sec-}M)$ $\dfrac{\Gamma \vdash u:\Pi C:*.(A \to C) \to (B \to C) \to C \quad \Gamma \vdash v:A \to C \quad \Gamma \vdash w:B \to C}{uCvw:C}$

If Px implies C for all x in S, then C holds on its own, i.e. that Px implies C for all x in S is redundant.
Second order encoding of $\exists$.
First order: $\exists \equiv \Pi S:*.\Pi P:S\to *.\neg\Pi x:S.\neg Px$, only works in classical logic.

(10) Existential quantification

Set $\exists \equiv \Pi S:*.\Pi P:S\to *.\Pi C:*.((\Pi x:S.(Px\to C))\to C)$ and write $\exists x\in S:P(x)$ for $\exists SP$.

We could also set $\forall \equiv \Pi S:*.\Pi P:S\to *.\Pi x:S.Px$ and write $\forall x\in S:P(x)$ for $\forall SP$.

We get $\exists$-introduction and $\exists$-elimination for free!

$(\exists\text{-intro})\ \dfrac{A\in S \quad P(A)}{\exists x\in S:P(x)}$ $\quad(\exists\text{-elim})\ \dfrac{\exists x\in S:P(x) \quad \forall x\in S:(P(x)\Rightarrow A)}{A}$

$(\exists\text{-intro-sec})\ \dfrac{A\in S \quad P(A)}{\forall C:((\forall x\in S:(P(x)\Rightarrow C))\Rightarrow C)}$

$\leftrightarrow (\exists\text{-intro-sec-M})\ \dfrac{\Gamma\vdash A:S \quad \Gamma\vdash u:PA}{\Gamma\vdash \lambda C:*.\lambda y:(\Pi x:S.(Px\to C)).yAu : \Pi C:*.((\Pi x:S.(Px\to C))\to C)}$

$(\exists\text{-elim-sec})\ \dfrac{\forall C:((\forall x\in S:(P(x)\Rightarrow C))\Rightarrow C) \quad \forall x\in S:(P(x)\Rightarrow A)}{A}$

$\leftrightarrow (\exists\text{-elim-sec-M})\ \dfrac{\Gamma\vdash u:\Pi C:*.((\Pi x:S.(Px\to C))\to C) \quad \Gamma\vdash v:\Pi x:S.(Px\to A)}{uAv:A}$

(11) Axioms

Add assumption in front of context.

Summary

(1) Sets $S:*$.

(2) Propositions $P:*$.

(3) Predicates $P:S\to *$.

(4) Implication $A\Rightarrow B = A\to B$ ($=\Pi x:A.B$ if $x\notin FV(B)$).

(5) Universal q. $\forall x\in S:P(x) = \Pi x:S.Px$.

(6) Absurdity $\bot \equiv \Pi \mathcal{L}:*.\mathcal{L}$.

(7) Negation $\neg \equiv \Pi A:*.A\to\bot$.

(8) Conjunction $\wedge \equiv \Pi A:*.\Pi B:*.\Pi C:*.(A\to B\to C)\to C$, $A\wedge B = \wedge AB$.

(9) Disjunction $\vee \equiv \Pi A:*.\Pi B:*.\Pi C:*.(A\to C)\to(B\to C)\to C$, $A\vee B = \vee AB$.

(10) Existential q. $\exists \equiv \Pi S:*.\Pi P:S\to *.\Pi C:*.((\Pi x:S.(Px\to C))\to C)$,
$\exists x\in S:P(x) = \exists SP$.

(11) Axioms Add assumption in front of context.

Example 6.5: $(A \vee B) \Rightarrow (\neg A \Rightarrow B)$

In λC: $(\Pi C:*.(A \to C) \to (B \to C) \to C) \to (A \to \bot) \to B$.

(a) $\boxed{A : *}$

(b) $\boxed{B : *}$

(c) $\boxed{x : \Pi C:*.(A \to C) \to (B \to C) \to C}$

(d) $\boxed{y : A \to \bot}$

(1) $xB : (A \to B) \to (B \to B) \to B$ — (app) on (c), (b)

(e) $\boxed{u : A}$

(2) $yu : \bot$ — (app) on (d), (e)

(3) $yuB : B$ — (app) on (2), (b)

(4) $\lambda u:A.yuB : A \to B$ — (abst) on (3)

(5) $xB(\lambda u:A.yuB) : (B \to B) \to B$ — (app) on (1), (4)

(f) $\boxed{v : B}$

(6) $v : B$ — (var) on (f)

(7) $\lambda v:B.v : B \to B$ — (abst) on (6)

(8) $xB(\lambda u:A.yuB)(\lambda v:B.v) : B$ — (app) on (5), (7)

(9) $\lambda y:A \to \bot.xB(\lambda u:A.yuB)(\lambda v:B.v) : (A \to \bot) \to B$ — (abst) on (8)

(10) $\lambda x:(\Pi C:*.(A \to C) \to (B \to C) \to C).\lambda y:A \to \bot.xB(\lambda u:A.yuB)(\lambda v:B.v) : (\Pi C:*.(A \to C) \to (B \to C) \to C) \to (A \to \bot) \to B$ — (abst) on (9)

Alternative for (c) and (d) for full encoding of $\neg$ and $\vee$:

(c') $\boxed{x : (\Pi \mathcal{L}:*.\Pi \mathcal{B}:*.\Pi C:*.(\mathcal{L} \to C) \to (\mathcal{B} \to C) \to C)AB}$

(d') $\boxed{y : (\Pi \mathcal{L}:*.\mathcal{L} \to \bot)A}$

(c) $x : \Pi C:*.(A \to C) \to (B \to C) \to C$ — (conv) on (c')

(d) $y : A \to \bot$ — (conv) on (d')

Type Theory, 68

Mittwoch, 25. Jänner 2023 19:57

Example 6.6: Proof of double negation from excluded middle.

(a) $\boxed{Ax_{EM} : \Pi \mathcal{L} : *. \mathcal{L} \vee \neg \mathcal{L}}$

(b) $\boxed{\beta : *}$

(c) $\boxed{x : \neg\neg\beta}$

(1) $Ax_{EM}\,\beta : \beta \vee \neg\beta$ — (app) on (a),(b)

(2) $Ax_{EM}\,\beta : \Pi C : *. (\beta \to C) \to (\neg\beta \to C) \to C$ — (app) on (a),(b)

(3) $Ax_{EM}\,\beta\beta : (\beta \to \beta) \to (\neg\beta \to \beta) \to \beta$ — (app) on (2),(b)

(4) $\lambda y : \beta . y : \beta \to \beta$ — (ad hoc)

(5) $Ax_{EM}\,\beta\beta(\lambda y : \beta . y) : (\neg\beta \to \beta) \to \beta$ — (app) on (3),(4)

(d) $\boxed{z : \neg\beta}$

(6) $xz : \bot$ — (app) on (c),(d)

(7) $xz\beta : \beta$ — (app) on (6),(b)

(8) $\lambda z : \neg\beta . xz\beta : \neg\beta \to \beta$ — (abst) on (7)

(9) $Ax_{EM}\,\beta\beta(\lambda y : \beta . y)(\lambda z : \neg\beta . xz\beta) : \beta$ — (app) on (5),(8)

(10) $\lambda x : \neg\neg\beta . Ax_{EM}\,\beta\beta(\lambda y : \beta . y)(\lambda z : \neg\beta . xz\beta) : \neg\neg\beta \to \beta$ — (abst) on (9)

(11) $\lambda \beta : * . \lambda x : \neg\neg\beta . Ax_{EM}\,\beta\beta(\lambda y : \beta . y)(\lambda z : \neg\beta . xz\beta) : \Pi \beta : * . \neg\neg\beta \to \beta$ — (abst) on (10)

Potential extensions of λC

- **Universes**: Set $\square_0 = \square$ and assume $\square_i \subset \square_{i+1}$ for all $i \in \mathbb{N}_0$.
  Add rules $\dfrac{\Gamma \vdash A : *}{\Gamma \vdash A : \square_0}$, $\dfrac{\Gamma \vdash A : \square_i}{\Gamma \vdash A : \square_{i+1}}$ and

  replace (form) with $\dfrac{\Gamma \vdash A : \square_i \quad \Gamma, X : A \vdash B : \square_j}{\Gamma \vdash \Pi X : A . B : \square_{\max\{i,j\}}}$.

- **Sum types**: $\dfrac{\Gamma \vdash A : * \quad \Gamma, X : A \vdash B : *}{\Gamma \vdash \Sigma X : A . B : *}$, $\dfrac{\Gamma \vdash A : \square_i \quad \Gamma, X : A \vdash B : \square_j}{\Gamma \vdash \Sigma X : A . B : \square_{\max\{i,j\}}}$.

- **Definitions**: $\to \lambda D$.
- **Inductive types**: $\to$ Calculus of Inductive Constructions.