Curs 12

Introducere in λ-calcul

λ-calcul

- În 1929-1932 Church a propus λ-calculul ca sistem formal pentru logica matematică. În 1935 a argumentat că orice funcție calculabilă peste numere naturale poate fi calculată in λ-calcul.
- În 1935, independent de Church, Turing a dezvoltat mecanismul de calcul numit astăzi Mașina Turing. În 1936 și el a argumentat câ orice funcție calculabilă peste numere naturale poate fi calculată de o mașină Turing. De asemenea, a arătat echivalența celor două modele de calcul. Această echivalență a constituit o indicație puternică asupra "universalității" celor două modele, conducând la ceea ce numim astăzi "Teza Church-Turing".

Referinţe

- Benjamin C. Pierce, Types and Programming Languages, The MIT Press 2002
- J.R. Hindley, J.P. Seldin, Lambda-Calculus and Combinators, an Introduction, Cambridge University Press, 2008
- R. Nederpelt, H. Geuvers, Type Theory and Formal Proof, an Introduction, Cambridge University Press 2014

λ-calcul: sintaxa

Lambda Calcul - sintaxă

$$\begin{array}{rll} t = & x & \text{(variabilă)} \\ & \mid \lambda x.\, t & \text{(abstractizare)} \\ & \mid t\ \ t & \text{(aplicare)} \end{array}$$

λ-calcul: sintaxa

Lambda Calcul - sintaxă

$$
\begin{array}{rll}
t = & x & \text{(variabilă)} \\
 & \mid \lambda x.\, t & \text{(abstractizare)} \\
 & \mid t\ \ t & \text{(aplicare)}
\end{array}
$$

λ-termeni

Fie $Var = \{x, y, z, \ldots\}$ o mulţime infinită de variabile.
Mulţimea λ-termenilor ΛT este definită inductiv astfel:

[Variabilă] $Var \subseteq \Lambda T$

[Aplicare] dacă t_1, $t_2 \in \Lambda T$ atunci $(t_1 t_2) \in \Lambda T$

[Abstractizare] dacă $x \in Var$ ş i $t \in \Lambda T$ atunci $(\lambda x.t) \in \Lambda T$

λ-termeni: exemple

- x, y, z
- (xy), (yx), $(x(yx))$
- $(\lambda x.x)$, $(\lambda x.(xy))$, $(\lambda z.(xy))$, $(\lambda x.(\lambda z.(xy)))$
- $((\lambda x.x)y)$, $((\lambda x.(xz))y)$, $((\lambda x.x)(\lambda y.y))$

λ-termeni: exemple

- x, y, z
- (xy), (yx), $(x(yx))$
- $(\lambda x.x)$, $(\lambda x.(xy))$, $(\lambda z.(xy))$, $(\lambda x.(\lambda z.(xy)))$
- $((\lambda x.x)y)$, $((\lambda x.(xz))y)$, $((\lambda x.x)(\lambda y.y))$

Convenţii:

- se elimină parantezele exterioare
- aplicarea este asociativă la stînga: $t_1 t_2 t_3$ este $(t_1 t_2) t_3$
- corpul abstractizării este extins la dreapta: $\lambda x.t_1 t_2$ este $\lambda x.(t_1 t_2)$ (nu $(\lambda x.t_1)t_2$)
- scriem $\lambda xyz.t$ în loc de $\lambda x.\lambda y.\lambda z.t$

Lambda termeni. Funcții anonime

λ-termeni: exemple

- x, y, z
- (xy), (yx), $(x(yx))$
- $(\lambda x.x)$, $(\lambda x.(xy))$, $(\lambda z.(xy))$, $(\lambda x.(\lambda z.(xy)))$
- $((\lambda x.x)y)$, $((\lambda x.(xz))y)$, $((\lambda x.x)(\lambda y.y))$

Funcţii anonime în Haskell

În Haskell, \ e folosit în locul simbolului λ și -> în locul punctului.

$\lambda x.x * x$ este `\x -> x * x`

$\lambda x.x > 0$ este `\x -> x > 0`

Variabile libere şi legate

Apariţii libere şi legate

Pentru un termen $\lambda x.t$ spunem că:

- apariţiile variabilei x în t sunt legate (*bound*)
- λx este legătura (*binder*), iar t este domeniul (*scope*) legării
- o apariţie a unei variabile este liberă (*free*) dacă apare într-o poziţie în care nu e legată.

Un termen fără variable libere se numeşte închis (*closed*).

Exemplu:

- $\lambda x.x$ este un termen închis
- $\lambda x.xy$ nu este termen închis, x este legată, y este liberă
- în termenul $x(\lambda x.xy)$ prima apariţie a lui x este liberă, a doua este legată.

Variabile libere

Mulţimea variabilelor libere $FV(t)$

Pentru un λ-termen t mulţimea variabilelor libere este definită astfel:

[Variabilă] $FV(x) = x$

[Aplicare] $FV(t_1 t_2) = FV(t_1) \cup FV(t_2)$

[Abstractizare] $FV(\lambda x.t) = FV(t) \setminus \{x\}$

Exemplu:

$$
\begin{aligned}
FV(\lambda x.xy) = \quad & FV(xy) \setminus \{x\} \\
& (FV(x) \cup FV(y)) \setminus \{x\} \\
& (\{x\} \cup \{y\}) \setminus \{x\} \\
& \{y\}
\end{aligned}
$$

Variabile libere

Mulţimea variabilelor libere $FV(t)$

Pentru un λ-termen t mulţimea variabilelor libere este definită astfel:

[Variabilă] $FV(x) = x$

[Aplicare] $FV(t_1 t_2) = FV(t_1) \cup FV(t_2)$

[Abstractizare] $FV(\lambda x.t) = FV(t) \setminus \{x\}$

Exemplu:

$$\begin{aligned} FV(\lambda x.xy) = \quad & FV(xy) \setminus \{x\} \\ & (FV(x) \cup FV(y)) \setminus \{x\} \\ & (\{x\} \cup \{y\}) \setminus \{x\} \\ & \{y\} \end{aligned}$$

$$FV(x\lambda x.xy) =$$

Variabile libere

Mulţimea variabilelor libere $FV(t)$

Pentru un λ-termen t mulţimea variabilelor libere este definită astfel:

[Variabilă] $FV(x) = x$

[Aplicare] $FV(t_1 t_2) = FV(t_1) \cup FV(t_2)$

[Abstractizare] $FV(\lambda x.t) = FV(t) \setminus \{x\}$

Exemplu:

$$
\begin{aligned}
FV(\lambda x.xy) = \quad & FV(xy) \setminus \{x\} \\
& (FV(x) \cup FV(y)) \setminus \{x\} \\
& (\{x\} \cup \{y\}) \setminus \{x\} \\
& \{y\} \\
FV(x\lambda x.xy) = \quad & \{x, y\}
\end{aligned}
$$

Substituţii

Fie t un λ-termen $x \in Var$.

Definiţie intuitivă

Pentru un λ-termen u vom nota prin $[u/x]t$ rezultatul înlocuirii tuturor apariţiilor libere ale lui x cu u în t.

Exemple: Dacă x, y, z sunt variabile distincte atunci

- $[y/x]\lambda z.x = \lambda z.y$
- $[(\lambda z.zw)/x](\lambda y.x) = \lambda y.\lambda z.zw$

Substituţii

Definirea substituţiei

Rezultatul substituirii lui x cu u în t este definit astfel:

[Variabilă] $[u/x]x = u$

[Variabilă] $[u/x]y = y$ dacă $x \neq y$

[Aplicare] $[u/x](t_1 t_2) = [u/x]t_1 [u/x]t_2$

[Abstractizare] $[u/x]\lambda y.t = \lambda y.[u/x]t$ unde

$y \neq x$ ş i $y \notin FV(u)$

Substituţii

Exemple: Dacă x, y, z sunt variabile distincte atunci

- $[y/x]\lambda z.x = \lambda z.y$
- Cine este $[y/x]\lambda y.x$?

Substituţii

Exemple: Dacă x, y, z sunt variabile distincte atunci

- $[y/x]\lambda z.x = \lambda z.y$

- Cine este $[y/x]\lambda y.x$?
  Dacă folosim definiţia intuitivă obţinem
  $[y/x]\lambda y.x = \lambda y.y$ ceea ce este greşit!

Substituţii

Exemple: Dacă x, y, z sunt variabile distincte atunci

- $[y/x]\lambda z.x = \lambda z.y$

- Cine este $[y/x]\lambda y.x$?
  Dacă folosim definiţia intuitivă obţinem
  $[y/x]\lambda y.x = \lambda y.y$ ceea ce este greşit!

  Cum procedăm pentru a repara greşeala? Observăm că $\lambda y.x$ desemneaza o funcţie constantă, aceeaşi funcţie putând fi reprezentată prin $\lambda z.x$. Aplicarea corectă a substituţiei este:
  $[y/x]\lambda y.x = [y/x]\lambda z.x = \lambda z.y$

Substituţii

Exemple: Dacă x, y, z sunt variabile distincte atunci

- $[y/x]\lambda z.x = \lambda z.y$
- Cine este $[y/x]\lambda y.x$?
  Dacă folosim definiţia intuitivă obţinem
  $[y/x]\lambda y.x = \lambda y.y$ ceea ce este greşit!

  Cum procedăm pentru a repara greşeala? Observăm că $\lambda y.x$ desemneaza o funcţie constantă, aceeaşi funcţie putând fi reprezentată prin $\lambda z.x$. Aplicarea corectă a substituţiei este:
  $[y/x]\lambda y.x = [y/x]\lambda z.x = \lambda z.y$

  Avem libertatea de a redenumi variabilele legate!

α-conversie (α-echivalenţă)

α-conversia $=_\alpha$

[Reflexivitate] $t =_\alpha t$

[Simetrie] $t_1 =_\alpha t_2$ implică $t_2 =_\alpha t_1$

[Tranzitivitate] $t_1 =_\alpha t_2$ şi $t_2 =_\alpha t_3$ implică $t_1 =_\alpha t_3$

[Redenumire] $\lambda x.t =_\alpha \lambda y.[y/x]t$ dacă $y \notin FV(t)$

[Compatibilitate] $t_1 =_\alpha t_2$ implică
$t\,t_1 =_\alpha t\,t_2$, $t_1 t =_\alpha t_2 t$ şi $\lambda x.t_1 =_\alpha \lambda x.t_2$

α-conversie (α-echivalenţă)

α-conversia $=_\alpha$

[Reflexivitate] $t =_\alpha t$

[Simetrie] $t_1 =_\alpha t_2$ implică $t_2 =_\alpha t_1$

[Tranzitivitate] $t_1 =_\alpha t_2$ şi $t_2 =_\alpha t_3$ implică $t_1 =_\alpha t_3$

[Redenumire] $\lambda x.t =_\alpha \lambda y.[y/x]t$ dacă $y \notin FV(t)$

[Compatibilitate] $t_1 =_\alpha t_2$ implică

$tt_1 =_\alpha tt_2$, $t_1 t =_\alpha t_2 t$ şi $\lambda x.t_1 =_\alpha \lambda x.t_2$

$t_1 =_\alpha t_2$ şi $u_1 =_\alpha u_2$ implică $[u_1/x]t_1 =_\alpha [u_2/x]t_2$

Exemplu:

$[xy/x](\lambda y.yx) =_\alpha [xy/x](\lambda z.zx) =_\alpha \lambda z.z(xy)$

α-conversie (α-echivalenţă)

α-conversia $=_\alpha$

[Reflexivitate] $t =_\alpha t$

[Simetrie] $t_1 =_\alpha t_2$ implică $t_2 =_\alpha t_1$

[Tranzitivitate] $t_1 =_\alpha t_2$ şi $t_2 =_\alpha t_3$ implică $t_1 =_\alpha t_3$

[Redenumire] $\lambda x.t =_\alpha \lambda y.[y/x]t$ dacă $y \notin FV(t)$

[Compatibilitate] $t_1 =_\alpha t_2$ implică

$$tt_1 =_\alpha tt_2,\ t_1t =_\alpha t_2t \text{ şi } \lambda x.t_1 =_\alpha \lambda x.t_2$$

$$t_1 =_\alpha t_2 \text{ şi } u_1 =_\alpha u_2 \text{ implică } [u_1/x]t_1 =_\alpha [u_2/x]t_2$$

Exemplu:

$[xy/x](\lambda y.yx) =_\alpha [xy/x](\lambda z.zx) =_\alpha \lambda z.z(xy)$

Vom lucra modulo α-conversie, doi termeni α-echivalenţi vor fi consideraţi "egali".

α-conversie

Exemplu:

- $[xy/x]\lambda x.yx =_\alpha [xy/x]\lambda z.yz = \lambda z.[xy/x](yz) = \lambda z.yz$

  Observăm că $\lambda z.yz =_\alpha \lambda x.yx$

Exemplu:

- $[xy/x]\lambda x.yx =_\alpha [xy/x]\lambda z.yz = \lambda z.[xy/x](yz) = \lambda z.yz$

  Observăm că $\lambda z.yz =_\alpha \lambda x.yx$

α-conversie

Exemplu:

- $[xy/x]\lambda x.yx =_\alpha [xy/x]\lambda z.yz = \lambda z.[xy/x](yz) = \lambda z.yz$

  Observăm că $\lambda z.yz =_\alpha \lambda x.yx$

- $[y/z]\lambda xy.zzx = \lambda x.[y/z]\lambda y.zzx =_\alpha \lambda x.[y/z]\lambda v.zzx = \lambda x.\lambda v.[y/z](zzx) = \lambda xv.yyx$

β-reducţie

β-reducţia este o relaţie pe mulţimea α-termenilor.

β-reducţia $\rightarrow_\beta$, $\stackrel{*}{\rightarrow}_\beta$

- un singur pas $\rightarrow_\beta \subseteq \Lambda T \times \Lambda T$

  [Aplicarea] $(\lambda x.t)u \rightarrow_\beta [u/x]t$

  [Compatibilitatea] $t_1 \rightarrow_\beta t_2$ implică

  $t\, t_1 \rightarrow_\beta t\, t_2$, $t_1 t \rightarrow_\beta t_2 t$ şi $\lambda x.t_1 \rightarrow_\beta \lambda x.t_2$

- zero sau mai mulţi paşi $\stackrel{*}{\rightarrow}_\beta \subseteq \Lambda T \times \Lambda T$

  $t_1 \stackrel{*}{\rightarrow}_\beta t_2$ dacă există $n \geq 0$ şi $u_0, \ldots, u_n$ astfel încât

  $t_1 =_\alpha u_0 \rightarrow_\beta u_1 \rightarrow_\beta \cdots \rightarrow_\beta u_n =_\alpha t_2$

β-reducţie

Să considerăm termenul $(\lambda x.(\lambda y.yx)z)v$

- $(\lambda x.(\lambda y.yx)z)v \to_\beta (\lambda y.yv)z \to_\beta zv$

β-reducţie

Să considerăm termenul $(\lambda x.(\lambda y.yx)z)v$

- $(\lambda x.(\lambda y.yx)z)v \to_\beta (\lambda y.yv)z \to_\beta zv$
- $(\lambda x.(\lambda y.yx)z)v \to_\beta (\lambda x.zx)v \to_\beta zv$

β-reducţie

Să considerăm termenul $(\lambda x.(\lambda y.yx)z)v$

- $(\lambda x.(\lambda y.yx)z)v \rightarrow_\beta (\lambda y.yv)z \rightarrow_\beta zv$
- $(\lambda x.(\lambda y.yx)z)v \rightarrow_\beta (\lambda x.zx)v \rightarrow_\beta zv$

Observăm că un termen poate fi β-redus în mai multe moduri.

Proprietatea de confluenţă ne asigură că vom ajunge întotdeauna la acelaşi rezultat.

Confluenţa β-reducţiei

Teorema Church-Rosser

Dacă $t \xrightarrow{*}_\beta t_1$ şi $t \xrightarrow{*}_\beta t_2$

$$\begin{tikzcd}
 & t \arrow[dl, "*"'] \arrow[dr, "*"] & \\
t_1 & & t_2
\end{tikzcd}$$

Confluenţa β-reducţiei

Teorema Church-Rosser

Dacă $t \xrightarrow{*}_{\beta} t_1$ şi $t \xrightarrow{*}_{\beta} t_2$

$$\begin{tikzcd}
 & t \arrow[dl, "*"'] \arrow[dr, "*"] & \\
t_1 & & t_2
\end{tikzcd}$$

atunci există u astfel încât $t_1 \xrightarrow{*}_{\beta} u$ şi $t_2 \xrightarrow{*}_{\beta} u$.

$$\begin{tikzcd}
 & t \arrow[dl, "*"'] \arrow[dr, "*"] & \\
t_1 \arrow[dr, "*"'] & & t_2 \arrow[dl, "*"] \\
 & u &
\end{tikzcd}$$

β-forma normală

Intuitiv, o formă normală este un termen care nu mai poate fi redus (sau punctul final al unui calcul).

Formă normală

- un λ-termen căruia nu i se mai poate aplica reducerea într-un pas $\rightarrow_\beta$ se numeş te *β-formă normală*
- dacă $t \stackrel{*}{\rightarrow}_\beta u_1$, $t \stackrel{*}{\rightarrow}_\beta u_2$ ş i u_1, u_2 sunt β-forme normale atunci, datorită confluenţei, $u_1 =_\alpha u_2$
- un λ-termen poate avea cel mult o β-formă normală (modulo α-echivalenţă)

β-forma normală

Formă normal ă

- un λ-termen căruia nu i se mai poate aplica reducerea într-un pas $\to_\beta$ se numeş te *β-formă normală*
- dacă $t \overset{*}{\to}_\beta u_1$, $t \overset{*}{\to}_\beta u_2$ ş i u_1, u_2 sunt η-forme normale atunci, datorită confluenţei, $u_1 =_\alpha u_2$
- un λ-termen poate avea cel mult o β-formă normală (modulo α-echivalenţă)

Exemplu:

- zv este β-formă normală pentru $(\lambda x.(\lambda y.yx)z)v$
  $(\lambda x.(\lambda y.yx)z)v \to_\beta (\lambda y.yv)z \to_\beta zv$
- există termeni care **nu** pot fi reduş i la o β-formă normală, de exemplu $(\lambda x.xx)(\lambda x.xx)$

β-conversia

Intuitiv, β-conversia extinde β-reducţia în ambele direcţii.

- $(\lambda y.yv)z \to_{\beta} zv \leftarrow_{\beta} (\lambda x.zx)v$

β-conversia

Intuitiv, β-conversia extinde β-reducţia în ambele direcţii.

- $(\lambda y.yv)z \to_\beta zv \leftarrow_\beta (\lambda x.zx)v$
- $(\lambda y.yv)z \leftarrow_\beta (\lambda x.(\lambda y.yx)z)v \to_\beta (\lambda x.zx)v$

β-conversia

Intuitiv, β-conversia extinde β-reducţia în ambele direcţii.

- $(\lambda y.yv)z \to_\beta zv \leftarrow_\beta (\lambda x.zx)v$
- $(\lambda y.yv)z \leftarrow_\beta (\lambda x.(\lambda y.yx)z)v \to_\beta (\lambda x.zx)v$

β-conversia $=_\beta$

- $=_\beta \subseteq \Lambda T \times \Lambda T$
  $t_1 =_\beta t_2$ dacă există $n \geq 0$ ş i $u_0, \ldots, u_n$ astfel încât
  $t_1 =_\alpha u_0$, $u_n =_\alpha t_2$ ş i, pentru orice i, $u_i \to_\beta u_{i+1}$ sau $u_{i+1} \to_\beta u_i$

β-conversia

Intuitiv, β-conversia extinde β-reducţia în ambele direcţii.

- $(\lambda y.yv)z \to_\beta zv \leftarrow_\beta (\lambda x.zx)v$
- $(\lambda y.yv)z \leftarrow_\beta (\lambda x.(\lambda y.yx)z)v \to_\beta (\lambda x.zx)v$

β-conversia $=_\beta$

- $=_\beta \subseteq \Lambda T \times \Lambda T$
  $t_1 =_\beta t_2$ dacă există $n \geq 0$ ş i $u_0, \ldots, u_n$ astfel încât
  $t_1 =_\alpha u_0$, $u_n =_\alpha t_2$ ş i, pentru orice i, $u_i \to_\beta u_{i+1}$ sau $u_{i+1} \to_\beta u_i$

Exemplu: $(\lambda y.yv)z =_\beta (\lambda x.zx)v$

β-conversia

β-conversia $=_\beta$

- $=_\beta \subseteq \Lambda T \times \Lambda T$
  $t_1 =_\beta t_2$ dacă există $n \geq 0$ ş i $u_0, \ldots, u_n$ astfel încât
  $t_1 =_\alpha u_0$, $u_n =_\alpha t_2$ ş i, pentru orice i, $u_i \to_\beta u_{i+1}$ sau $u_{i+1} \to_\beta u_i$

β-conversia

β-conversia $=_\beta$

- $=_\beta \subseteq \Lambda T \times \Lambda T$
  $t_1 =_\beta t_2$ dacă există $n \geq 0$ ş i $u_0, \ldots, u_n$ astfel încât
  $t_1 =_\alpha u_0$, $u_n =_\alpha t_2$ ş i, pentru orice i, $u_i \to_\beta u_{i+1}$ sau $u_{i+1} \to_\beta u_i$

Observaţii

- $=_\beta$ este o relaţie de echivalenţă

β-conversia

β-conversia $=_\beta$

- $=_\beta \subseteq \Lambda T \times \Lambda T$
  $t_1 =_\beta t_2$ dacă există $n \geq 0$ şi $u_0, \ldots, u_n$ astfel încât
  $t_1 =_\alpha u_0$, $u_n =_\alpha t_2$ şi, pentru orice i, $u_i \rightarrow_\beta u_{i+1}$ sau $u_{i+1} \rightarrow_\beta u_i$

Observaţii

- $=_\beta$ este o relaţie de echivalenţă
- pentru t_1, t_2 λ-termeni şi u_1, u_2 β-forme normale
  dacă $t_1 \stackrel{*}{\rightarrow}_\beta u_1$, $t_2 \stackrel{*}{\rightarrow}_\beta u_2$ şi $u_1 =_\alpha u_2$ atunci $t_1 =_\beta t_2$

β-conversia

β-conversia $=_\beta$

- $=_\beta \subseteq \Lambda T \times \Lambda T$
  $t_1 =_\beta t_2$ dacă există $n \geq 0$ ş i $u_0, \ldots, u_n$ astfel încât
  $t_1 =_\alpha u_0$, $u_n =_\alpha t_2$ ş i, pentru orice i, $u_i \rightarrow_\beta u_{i+1}$ sau $u_{i+1} \rightarrow_\beta u_i$
- $=_\beta$ este o relaţie de echivalenţă

β-conversia

β-conversia $=_\beta$

- $=_\beta \subseteq \Lambda T \times \Lambda T$
  $t_1 =_\beta t_2$ dacă există $n \geq 0$ ş i $u_0, \ldots, u_n$ astfel încât
  $t_1 =_\alpha u_0$, $u_n =_\alpha t_2$ ş i, pentru orice i, $u_i \to_\beta u_{i+1}$ sau $u_{i+1} \to_\beta u_i$
- $=_\beta$ este o relaţie de echivalenţă
- pentru t_1, t_2 λ-termeni ş i u_1, u_2 β-forme normale
  dacă $t_1 \stackrel{*}{\to}_\beta u_1$, $t_2 \stackrel{*}{\to}_\beta u_2$ şi $u_1 =_\alpha u_2$ atunci $t_1 =_\beta t_2$

β-conversia reprezintă "egalitatea prin calcul", iar β-reducţia (modulo α-conversie) oferă o procedură de decizie pentru aceasta.

Codificări în λ-calcul

Ideea generală

Intuiție

Tipurile de date sunt codificate de capabilități

Boole	capabilitatea de a alege între două alternative
Perechi	capabilitatea de a calcula ceva bazat pe două valori
Numere naturale	capabilitatea de a itera de un număr dat de ori

Valori de adevăr

Intuiție: Capabilitatea de a alege între două alternative.

Codificare: Un Boolean este o funcție cu 2 argumente reprezentând ramurile unei alegeri.

true ::= $\lambda t\ f.t$ — din cele două alternative o alege pe prima

false ::= $\lambda t\ f.f$ — din cele două alternative o alege pe a doua

Operații Booleene

true ::= $\lambda t\ f.t$ — din cele două alternative o alege pe prima

false ::= $\lambda t\ f.f$ — din cele două alternative o alege pe a doua

`if` ::= $\lambda c\ t\ e.c\ t\ e$ — pur și simplu folosim valoarea de adevăr pentru a alege între alternative

Operații Booleene

true ::= $\lambda t\ f.t$ — din cele două alternative o alege pe prima

false ::= $\lambda t\ f.f$ — din cele două alternative o alege pe a doua

`if` ::= $\lambda c\ t\ e.c\ t\ e$ — pur și simplu folosim valoarea de adevăr pentru a alege între alternative

`if` $\mathit{false}\ (\lambda x.x\ x)\ (\lambda x.x) \to_\beta^3$
$\mathit{false}\ (\lambda x.x\ x)\ (\lambda x.x) \to_\beta^2 \lambda x.x$

Operații Booleene

true ::= $\lambda t\ f.t$ — din cele două alternative o alege pe prima

false ::= $\lambda t\ f.f$ — din cele două alternative o alege pe a doua

`if` ::= $\lambda c\ t\ e.c\ t\ e$

`and` ::= $\lambda b1\ b2.$ `if` $b1\ b2$ *false* sau $\lambda b1\ b2.b1\ b2\ b1$

`and` *true false*

Operații Booleene

$true ::= \lambda t\ f.t$ — din cele două alternative o alege pe prima

$false ::= \lambda t\ f.f$ — din cele două alternative o alege pe a doua

$\texttt{if} ::= \lambda c\ t\ e.c\ t\ e$

$\texttt{and} ::= \lambda b1\ b2.\ \texttt{if}\ b1\ b2\ false$ sau $\lambda b1\ b2.b1\ b2\ b1$

$\texttt{and}\ true\ false \rightarrow_{\beta}^{2} true\ false\ true \rightarrow_{\beta}^{2} false$

$\texttt{or} ::= \lambda b1\ b2.\ \texttt{if}\ b1\ true\ b2$ sau $\lambda b1\ b2.b1\ b1\ b2$

$\texttt{or}\ true\ false$

Operații Booleene

$true ::= \lambda t\ f.t$ — din cele două alternative o alege pe prima

$false ::= \lambda t\ f.f$ — din cele două alternative o alege pe a doua

$\texttt{if} ::= \lambda c\ t\ e.c\ t\ e$

$\texttt{and} ::= \lambda b1\ b2.\ \texttt{if}\ b1\ b2\ false$ sau $\lambda b1\ b2.b1\ b2\ b1$

$\texttt{and}\ true\ false \rightarrow^2_\beta true\ false\ true \rightarrow^2_\beta false$

$\texttt{or} ::= \lambda b1\ b2.\ \texttt{if}\ b1\ true\ b2$ sau $\lambda b1\ b2.b1\ b1\ b2$

$\texttt{or}\ true\ false \rightarrow^2_\beta true\ true\ false \rightarrow^2_\beta true$

$\texttt{not} ::= \lambda b.\ \texttt{if}\ b\ false\ true$ sau $\lambda b\ t\ f.b\ f\ t$

$\texttt{not}\ true \rightarrow_\beta \lambda t\ f.true\ f\ t \rightarrow_\beta \lambda t\ f.f$

Perechi

Intuiție: Capabilitatea de a aplica o funcție componentelor perechii

Codificare: O funcție cu 3 argumente reprezentând componentele perechii și funcția ce vrem să o aplicăm lor.

`pair` ::= $\lambda x\ y.\lambda f.f\ x\ y$
Constructorul de perechi

Exemplu: `pair` $x\ y \rightarrow_{\beta}^{2} \lambda f.f\ x\ y$

perechea (x, y) reprezintă capabilitatea de a aplica o funcție de două argumente lui x si apoi lui y.

Operații pe perechi

$$\texttt{pair} ::= \lambda x\, y.\lambda f.f\ x\ y$$

$$\texttt{pair}\ x\ y \equiv_\beta f\ x\ y$$

$\texttt{fst} ::= \lambda p.p\ \mathit{true}$ — *true* alege prima componentă

$$\texttt{fst}\ (\texttt{pair}\ x\ y) \rightarrow_\beta \texttt{pair}\ x\ y\ \mathit{true} \rightarrow_\beta^3 \mathit{true}\ x\ y \rightarrow_\beta^2 x$$

$\texttt{snd} ::= \lambda p.p\ \mathit{false}$ — *false* alege a doua componentă

$$\texttt{snd}\ (\texttt{pair}\ x\ y) \rightarrow_\beta \texttt{pair}\ x\ y\ \mathit{false} \rightarrow_\beta^3 \mathit{false}\ x\ y \rightarrow_\beta^2 y$$

Numere naturale

Intuiție: Capabilitatea de a itera o funcție de un număr de ori peste o valoare inițială

Codificare: Un număr natural este o funcție cu 2 argumente

- s funcția care se iterează
- z valoarea inițială

0 ::= $\lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

Numere naturale

Intuiție: Capabilitatea de a itera o funcție de un număr de ori peste o valoare inițială

Codificare: Un număr natural este o funcție cu 2 argumente

- s funcția care se iterează
- z valoarea inițială

0 ::= $\lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

1 ::= $\lambda s\ z.s\ z$ — funcția iterată o dată aplicată valorii inițiale

Numere naturale

Intuiție: Capabilitatea de a itera o funcție de un număr de ori peste o valoare inițială

Codificare: Un număr natural este o funcție cu 2 argumente

- s funcția care se iterează
- z valoarea inițială

0 ::= $\lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

1 ::= $\lambda s\ z.s\ z$ — funcția iterată o dată aplicată valorii inițiale

2 ::= $\lambda s\ z.s(s\ z)$ — s iterată de 2 ori, aplicată valorii inițiale

Numere naturale

Intuiție: Capabilitatea de a itera o funcție de un număr de ori peste o valoare inițială

Codificare: Un număr natural este o funcție cu 2 argumente

- s funcția care se iterează
- z valoarea inițială

0 ::= $\lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

1 ::= $\lambda s\ z.s\ z$ — funcția iterată o dată aplicată valorii inițiale

2 ::= $\lambda s\ z.s(s\ z)$ — s iterată de 2 ori, aplicată valorii inițiale

...

8 ::= $\lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

...

Numere naturale

Intuiție: Capabilitatea de a itera o funcție de un număr de ori peste o valoare inițială

Codificare: Un număr natural este o funcție cu 2 argumente

- s funcția care se iterează
- z valoarea inițială

0 ::= $\lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

1 ::= $\lambda s\ z.s\ z$ — funcția iterată o dată aplicată valorii inițiale

2 ::= $\lambda s\ z.s(s\ z)$ — s iterată de 2 ori, aplicată valorii inițiale

...

8 ::= $\lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

...

Observație: $0 = \mathit{false}$

Operații aritmetice de bază

$0 ::= \lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

$8 ::= \lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

$S ::= \lambda n\ s\ z.s\ (n\ s\ z)$ sau $\lambda n\ s\ z.n\ s\ (sz)$

$S\ 0$

Operații aritmetice de bază

$0 ::= \lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

$8 ::= \lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

$S ::= \lambda n\ s\ z.s\ (n\ s\ z)$ sau $\lambda n\ s\ z.n\ s\ (sz)$

$$S\ 0 \to_\beta \lambda s\ z.0s(sz) \to_\beta^2 \lambda s\ z.sz = 1$$

Operații aritmetice de bază

$0 ::= \lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

$8 ::= \lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

$S ::= \lambda n\ s\ z.s\ (n\ s\ z)$ sau $\lambda n\ s\ z.n\ s\ (sz)$

$S\ 0 \rightarrow_\beta \lambda s\ z.0s(sz) \rightarrow_\beta^2 \lambda s\ z.sz = 1$

Observăm că $m\ s$ aplică funcţia s de m ori.

Operații aritmetice de bază

0 ::= $\lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

8 ::= $\lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

S ::= $\lambda n\ s\ z.s\ (n\ s\ z)$ sau $\lambda n\ s\ z.n\ s\ (sz)$

$S\ 0 \to_\beta \lambda s\ z.0s(sz) \to_\beta^2 \lambda s\ z.sz = 1$

Observăm că $m\ s$ aplică funcţia s de m ori.

$+$ ::= $\lambda m\ n.(m\ \ S)\ n$ sau $\lambda m\ n.\lambda s\ z.m\ s\ (n\ s\ z)$

$+\ 3\ 2$

Operații aritmetice de bază

$0 ::= \lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

$8 ::= \lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

$\mathsf{S} ::= \lambda n\ s\ z.s\ (n\ s\ z)$ sau $\lambda n\ s\ z.n\ s\ (sz)$

$$\mathsf{S}\ 0 \rightarrow_\beta \lambda s\ z.0s(sz) \rightarrow_\beta^2 \lambda s\ z.sz = 1$$

Observăm că $m\ s$ aplică funcţia s de m ori.

$+ ::= \lambda m\ n.(m\ \ \mathsf{S})\ n$ sau $\lambda m\ n.\lambda s\ z.m\ s\ (n\ s\ z)$

$$+\ 3\ 2 \rightarrow_\beta^2 \lambda s\ z.3\ s\ (2\ s\ z) \rightarrow_\beta^2$$
$$\lambda s\ z.s(s(s(2\ s\ z))) \rightarrow_\beta^2 \lambda s\ z.s(s(s(s(s\ z)))) = 5$$

Operații aritmetice de bază

0 ::= $\lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

8 ::= $\lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

S ::= $\lambda n\ s\ z.s\ (n\ s\ z)$ sau $\lambda n\ s\ z.n\ s\ (sz)$

$S\ 0 \rightarrow_\beta \lambda s\ z.0s(sz) \rightarrow^2_\beta \lambda s\ z.sz = 1$

Observăm că $m\ s$ aplică funcţia s de m ori.

+ ::= $\lambda m\ n.(m\ \ S)\ n$ sau $\lambda m\ n.\lambda s\ z.m\ s\ (n\ s\ z)$

$+\ 3\ 2 \rightarrow^2_\beta \lambda s\ z.3\ s\ (2\ s\ z) \rightarrow^2_\beta$
$\lambda s\ z.s(s(s(2\ s\ z))) \rightarrow^2_\beta \lambda s\ z.s(s(s(s(s\ z)))) = 5$

* ::= $\lambda m\ n.m\ (+\ n)\ 0$ sau $\lambda m\ n.\lambda s.m\ (n\ s)$

$*\ 3\ 2$

Operații aritmetice de bază

0 ::= $\lambda s\ z.z$ — s se iterează de 0 ori, deci valoarea inițială

8 ::= $\lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

S ::= $\lambda n\ s\ z.s\ (n\ s\ z)$ sau $\lambda n\ s\ z.n\ s\ (sz)$

$\mathsf{S}\ 0 \rightarrow_\beta \lambda s\ z.0s(sz) \rightarrow^2_\beta \lambda s\ z.sz = 1$

Observăm că $m\ s$ aplică funcţia s de m ori.

+ ::= $\lambda m\ n.(m\ \ \mathsf{S})\ n$ sau $\lambda m\ n.\lambda s\ z.m\ s\ (n\ s\ z)$

$+\ 3\ 2 \rightarrow^2_\beta \lambda s\ z.3\ s\ (2\ s\ z) \rightarrow^2_\beta$
$\lambda s\ z.s(s(s(2\ s\ z))) \rightarrow^2_\beta \lambda s\ z.s(s(s(s(s\ z)))) = 5$

* ::= $\lambda m\ n.m\ (+\ n)\ 0$ sau $\lambda m\ n.\lambda s.m\ (n\ s)$

$*\ 3\ 2 \rightarrow^2_\beta 3\ (+\ 2)0 \rightarrow^2_\beta +\ 2(+\ 2(+\ 2\ 0)) \rightarrow^4_\beta$
$+\ 2(+\ 2\ 2) \rightarrow^4_\beta +\ 2\ 4 \rightarrow^4_\beta 6$

Liste

Intuiție: Capabilitatea de a agrega o listă

Codificare: O funcție cu 2 argumente:
funcția de agregare și *valoarea inițială*
Lista $[3, 5]$ este reprezentată prin $a\ 3\ (a\ 5\ i)$

Liste

Intuiție: Capabilitatea de a agrega o listă

Codificare: O funcție cu 2 argumente:
funcția de agregare și *valoarea inițială*
Lista $[3, 5]$ este reprezentată prin $a\ 3\ (a\ 5\ i)$

`null` ::= $\lambda a\ i.i$ — lista vidă

`cons` ::= $\lambda x\ l.\lambda a\ i.a\ x\ (l\ a\ i)$
Constructorul de liste

Exemplu: $\texttt{cons}\ 3\ (\texttt{cons}\ 5\ \texttt{null}) \rightarrow_{\beta}^{2} \lambda a\ i.a\ 3\ (\texttt{cons}\ 5\ \texttt{null}\ a\ i) \rightarrow_{\beta}^{4} \lambda a\ i.a\ 3\ (a\ 5\ (\texttt{null}\ a\ i)) \rightarrow_{\beta}^{2} \lambda a\ i.a\ 3\ (a\ 5\ i)$

Lista $[3, 5]$ reprezintă capabilitatea de a agrega elementele 3 si apoi 5 dată fiind o funcție de agregare a și o valoare implicită i.

Operații pe liste

`null` $::= \lambda a\ i.i$ — lista vidă

`cons` $::= \lambda x\ l.\lambda a\ i.a\ x\ (l\ a\ i)$

Operații pe liste

$$\texttt{null} ::= \lambda a\ i.i \text{ — lista vidă}$$
$$\texttt{cons} ::= \lambda x\ l.\lambda a\ i.a\ x\ (l\ a\ i)$$

$$\texttt{?null} ::= \lambda l.l\ (\lambda x\ v.\mathit{false})\ \mathit{true}$$

Operații pe liste

`null` ::= $\lambda a\ i.i$ — lista vidă

`cons` ::= $\lambda x\ l.\lambda a\ i.a\ x\ (l\ a\ i)$

`?null` ::= $\lambda l.l\ (\lambda x\ v.\mathit{false})\ \mathit{true}$

`head` ::= $\lambda d\ l.l\ (\lambda x\ v.x)\ d$

primul element al listei, sau d dacă lista e vidă

Operații pe liste

$\texttt{null} ::= \lambda a\ i.i$ — lista vidă

$\texttt{cons} ::= \lambda x\ l.\lambda a\ i.a\ x\ (l\ a\ i)$

$\texttt{?null} ::= \lambda l.l\ (\lambda x\ v.\mathit{false})\ \mathit{true}$

$\texttt{head} ::= \lambda d\ l.l\ (\lambda x\ v.x)\ d$
primul element al listei, sau d dacă lista e vidă

$\texttt{tail} ::= \lambda l.\ \texttt{fst}\ (l\ (\lambda x\ p.\ \texttt{pair}\ (\texttt{snd}\ p)\ (\texttt{cons}\ x\ (\texttt{snd}\ p)))\ (\texttt{pair}\ \texttt{null}\ \texttt{null}))$
coada listei, sau lista vidă dacă lista e vidă

Liste ca perechi

Intuiție: putem reprezenta o lista ca o pereche formată din primul element si restul listei

Lista vidă: folosim şi o valoare booleana care indică dacă lista este vidă sau nevidă
Lista $[3, 5]$ este reprezentată prin
$\textit{pair false } (\textit{pair } 3\ (\textit{pair false } (\textit{pair } 5\ (\textit{pair true true}))))$

Liste ca perechi

Intuiție: putem reprezenta o lista ca o pereche formată din primul element si restul listei

Lista vidă: folosim şi o valoare booleana care indică dacă lista este vidă sau nevidă

Lista $[3, 5]$ este reprezentată prin

$pair\ false\ (pair\ 3\ (pair\ false\ (pair\ 5\ (pair\ true\ true))))$

Liste ca perechi

Intuiție: putem reprezenta o lista ca o pereche formată din primul element si restul listei

Lista vidă: folosim şi o valoare booleana care indică dacă lista este vidă sau nevidă
Lista $[3, 5]$ este reprezentată prin
pair false (*pair* 3 (*pair false* (*pair* 5 (*pair true true*))))

`null` ::= `pair` *true true* — lista vidă

`?null` ::= `fst`

Liste ca perechi

Intuiție: putem reprezenta o lista ca o pereche formată din primul element si restul listei

Lista vidă: folosim şi o valoare booleana care indică dacă lista este vidă sau nevidă
Lista $[3, 5]$ este reprezentată prin
$\mathit{pair}\ \mathit{false}\ (\mathit{pair}\ 3\ (\mathit{pair}\ \mathit{false}\ (\mathit{pair}\ 5\ (\mathit{pair}\ \mathit{true}\ \mathit{true}))))$

`null` ::= `pair` *true true* — lista vidă

`?null` ::= `fst`

`cons` ::= $\lambda x\ l.$ `pair` *false* (`pair` $x\ l$)

Liste ca perechi

Intuiție: putem reprezenta o lista ca o pereche formată din primul element si restul listei

Lista vidă: folosim şi o valoare booleana care indică dacă lista este vidă sau nevidă
Lista $[3, 5]$ este reprezentată prin
$pair\ false\ (pair\ 3\ (pair\ false\ (pair\ 5\ (pair\ true\ true))))$

null ::= $\texttt{pair}\ \ true\ true$ — lista vidă

?null ::= $\texttt{fst}$

cons ::= $\lambda x\ l.\ \texttt{pair}\ \ false\ (\texttt{pair}\ \ x\ l)$

head ::= $\lambda l.\ \texttt{fst}\ \ (\texttt{snd}\ \ l)$

tail ::= $\lambda l.\ \texttt{snd}\ \ (\texttt{snd}\ \ l)$

Pe săptămâna viitoare!