Curs 7

λ-calcul și calculabilitate

- 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".

λ-calcul: sintaxa

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

Convenții:

- 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 $\lambda x.\lambda y.\lambda z.t$

λ-calcul: sintaxa

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

Convenții:

- 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 $\lambda x.\lambda y.\lambda z.t$

Întrebare

Ce putem exprima / calcula folosind **doar** λ-calcul?

Rezumat

- Reprezentarea valorilor de adevăr și a expresiilor condiționale
- Reprezentarea perechilor (tuplurilor) și a funcțiilor proiecție
- Reprezentarea numerelor și a operatiilor aritmetice de bază
- Recursie

Ideea generală

Intuiție

Tipurile de date sunt codificate de <u>capabilități</u>

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` ::= $\lambda c\ then\ else.c\ then\ else$ — pur și simplu folosim valoarea de adevăr pentru a alege între alternative
`if` *false* $(\lambda x.x\ x)\ (\lambda x.x)$

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 ::= $\lambda c\ then\ else.c\ then\ else$ — pur și simplu folosim valoarea de adevăr pentru a alege între alternative
if $false\ (\lambda x.x\ x)\ (\lambda x.x) \rightarrow_\beta^3$
$false\ (\lambda x.x\ x)\ (\lambda x.x) \rightarrow_\beta^2 \lambda x.x$

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

if ::= $\lambda c\ then\ else.c\ then\ else$ — pur și simplu folosim valoarea de adevăr pentru a alege între alternative
if $false\ (\lambda x.x\ x)\ (\lambda x.x) \rightarrow_\beta^3$
$false\ (\lambda x.x\ x)\ (\lambda x.x) \rightarrow_\beta^2 \lambda x.x$

and ::= $\lambda b1\ b2.$ **if** $b1\ b2\ false$ sau $\lambda b1\ b2.b1\ b2\ b1$
and $true\ false \rightarrow_\beta^2 true\ false\ true \rightarrow_\beta^2 false$

or ::= $\lambda b1\ b2.$ **if** $b1\ true\ b2$ sau $\lambda b1\ b2.b1\ b1\ b2$
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

if ::= $\lambda c\ then\ else.c\ then\ else$ — pur și simplu folosim valoarea de adevăr pentru a alege între alternative
if $false\ (\lambda x.x\ x)\ (\lambda x.x) \rightarrow_\beta^3$
$false\ (\lambda x.x\ x)\ (\lambda x.x) \rightarrow_\beta^2 \lambda x.x$

and ::= $\lambda b1\ b2.$ **if** $b1\ b2\ false$ sau $\lambda b1\ b2.b1\ b2\ b1$
and $true\ false \rightarrow_\beta^2 true\ false\ true \rightarrow_\beta^2 false$

or ::= $\lambda b1\ b2.$ **if** $b1\ true\ b2$ sau $\lambda b1\ b2.b1\ b1\ b2$
or $true\ false \rightarrow_\beta^2 true\ true\ false \rightarrow_\beta^2 true$

not ::= $\lambda b.$ **if** $b\ false\ true$ sau $\lambda b\ t\ f.b\ f\ t$
not $true$

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 ::= $\lambda c\ then\ else.c\ then\ else$ — pur și simplu folosim valoarea de adevăr pentru a alege între alternative
$\mathbf{if}\ false\ (\lambda x.x\ x)\ (\lambda x.x) \to_\beta^3$
$false\ (\lambda x.x\ x)\ (\lambda x.x) \to_\beta^2 \lambda x.x$

and ::= $\lambda b1\ b2.\ \mathbf{if}\ b1\ b2\ false$ sau $\lambda b1\ b2.b1\ b2\ b1$
$\mathbf{and}\ true\ false \to_\beta^2 true\ false\ true \to_\beta^2 false$

or ::= $\lambda b1\ b2.\ \mathbf{if}\ b1\ true\ b2$ sau $\lambda b1\ b2.b1\ b1\ b2$
$\mathbf{or}\ true\ false \to_\beta^2 true\ true\ false \to_\beta^2 true$

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

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

$$\begin{aligned}
\mathbf{pair} &::= \lambda x\ y.\lambda f.f\ x\ y\\
\mathbf{pair}\ xy &\equiv_\beta f\ x\ y
\end{aligned}$$

$$\begin{aligned}
\mathbf{fst} &::= \lambda p.p\ \mathit{true} \text{ — } \mathit{true} \text{ alege prima componentă}\\
&\quad \mathbf{fst}\ (\mathbf{pair}\ x\ y) \rightarrow_\beta \mathbf{pair}\ x\ y\ \mathit{true} \rightarrow_\beta^3 \mathit{true}\ x\ y \rightarrow_\beta^2 x\\
\mathbf{snd} &::= \lambda p.p\ \mathit{false} \text{ — } \mathit{false} \text{ alege a doua componentă}\\
&\quad \mathbf{snd}\ (\mathbf{pair}\ x\ y) \rightarrow_\beta \mathbf{pair}\ x\ y\ \mathit{false} \rightarrow_\beta^3 \mathit{false}\ x\ y \rightarrow_\beta^2 y
\end{aligned}$$

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

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

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

$\mathbf{S}\ 0$

Operații aritmetice de bază

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

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

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

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

Operații aritmetice de bază

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

$\mathbf{8} ::= \lambda s\ z.s(s(s(s(s(s(s(s\ z)))))))$

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

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

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

Operații aritmetice de bază

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

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

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

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

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

$+ ::= \lambda m\ n.(m\ \ \mathbf{S})\ n$ 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

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

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

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

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

$+ ::= \lambda m\ n.(m\ \ \mathbf{S})\ n$ 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

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

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

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

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

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

$+\ 3\ 2 \to_\beta^2 \lambda s\ z.3\ s\ (2\ s\ z) \to_\beta^2$

$\lambda s\ z.s(s(s(2\ s\ z))) \to_\beta^2 \lambda s\ z.s(s(s(s(s\ z)))) = 5$

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

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

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

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

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

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

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

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

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

$*\ 3\ 2 \rightarrow_\beta^2 3\ (+\ 2)0 \rightarrow_\beta^2 +\ 2(+\ 2(+\ 2\ 0)) \rightarrow_\beta^4$

$+\ 2(+\ 2\ 2) \rightarrow_\beta^4 +\ 2\ 4 \rightarrow_\beta^4 6$

Comparații

Comparații

- $- ::= \quad \lambda m\, n.n\, \mathbf{pred}\, m$ — dă 0 dacă $m \leq n$

Comparații

$\texttt{-} ::= \lambda m\ n.n\ \textbf{pred}\ m$ — dă 0 dacă $m \leq n$

$\texttt{isZero} ::= \lambda n.n(\lambda x.\mathit{false})\mathit{true}$ — testează dacă n e 0

Comparații

- ::= $\lambda m\ n.n$ **pred** m — dă 0 dacă $m \leq n$

isZero ::= $\lambda n.n(\lambda x.false)true$ — testează dacă n e 0

<= ::= $\lambda m\ n.$ **isZero** (**-** $m\ n$)

Comparații

$$\mathtt{-} ::= \lambda m\ n.n\ \mathbf{pred}\ m \text{ — dă 0 dacă } m \leq n$$

$$\mathtt{isZero} ::= \lambda n.n(\lambda x.\mathit{false})\mathit{true} \text{ — testează dacă } n \text{ e 0}$$

$$\mathtt{<=} ::= \lambda m\ n.\ \mathtt{isZero}\ (\mathtt{-}\ m\ n)$$

$$\mathtt{>} ::= \lambda m\ n.\ \mathtt{not}\ (\mathtt{<=}\ m\ n)$$

$$\mathtt{>=} ::= \lambda m\ n.\ \mathtt{<=}\ n\ m$$

$$\mathtt{<} ::= \lambda m\ n.\ \mathtt{>}\ n\ m$$

Comparații

$\mathbf{-} ::= \lambda m\, n.n\, \mathbf{pred}\, m$ — dă 0 dacă $m \leq n$

$\mathbf{isZero} ::= \lambda n.n(\lambda x.false)true$ — testează dacă n e 0

$\mathbf{<=} ::= \lambda m\, n.\, \mathbf{isZero}\, (\mathbf{-}\, m\, n)$

$\mathbf{>} ::= \lambda m\, n.\, \mathbf{not}\, (\mathbf{<=}\, m\, n)$

$\mathbf{>=} ::= \lambda m\, n.\, \mathbf{<=}\, n\, m$

$\mathbf{<} ::= \lambda m\, n.\, \mathbf{>}\, n\, m$

$\mathbf{eq} ::= \lambda m\, n.\, \mathbf{and}\, (\mathbf{<=}\, m\, n)\, (\mathbf{>=}\, m\, n)$

$\mathbf{neq} ::= \lambda m\, n.\, \mathbf{not}\, (\mathbf{eq}\, m\, n)$

Comparații

$$\begin{aligned}
\texttt{-} &::= \lambda m\ n.n\ \mathbf{pred}\ m \text{ — dă 0 dacă } m \leq n \\
\texttt{isZero} &::= \lambda n.n(\lambda x.\mathit{false})\mathit{true} \text{ — testează dacă } n \text{ e } 0 \\
\texttt{<=} &::= \lambda m\ n.\ \mathbf{isZero}\ (\texttt{-}\ m\ n) \\
\texttt{>} &::= \lambda m\ n.\ \mathbf{not}\ (\texttt{<=}\ m\ n) \\
\texttt{>=} &::= \lambda m\ n.\ \texttt{<=}\ n\ m \\
\texttt{<} &::= \lambda m\ n.\ \texttt{>}\ n\ m \\
\texttt{eq} &::= \lambda m\ n.\ \mathbf{and}\ (\texttt{<=}\ m\ n)\ (\texttt{>=}\ m\ n) \\
\texttt{neq} &::= \lambda m\ n.\ \mathbf{not}\ (\mathbf{eq}\ m\ n)
\end{aligned}$$

Problemă

Cum definim funcția **pred**?

Definirea funcției **pred** folosind perechi

Vrem să definim:

$$\mathbf{pred}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$$

Definirea funcției **pred** folosind perechi

Vrem să definim:

$$\mathbf{pred}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$$

- definim o funcție care primind ca argument perechea $(n-1, n)$ în toarce perechea $(n, n+1)$

$$\mathbf{S''} = \lambda p.(\lambda x.\ \mathbf{pair}\ x(\mathbf{S}\ x))\ (\mathbf{snd}\ p)$$

- aplicăm funcția de mai sus de n ori plecând de la perechea **pair** 00

$$\mathbf{pred''} = \lambda n.n\ \mathbf{S''}\ (\mathbf{pair}\ 0\ 0)$$

Intitiv vom avea $(0,0) \mapsto (0,1) \mapsto (1,2) \mapsto \cdots \mapsto (n-1, n)$

Definirea funcției **pred** folosind perechi

Vrem să definim:

$$\mathbf{pred}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$$

- definim o funcție care primind ca argument perechea $(n-1, n)$ în toarce perechea $(n, n+1)$

$$\mathbf{S''} = \lambda p.(\lambda x.\, \mathbf{pair}\ x (\mathbf{S}\ x))\ (\mathbf{snd}\ p)$$

- aplicăm funcția de mai sus de n ori plecând de la perechea **pair** 00

$$\mathbf{pred''} = \lambda n.n\ \mathbf{S''}\ (\mathbf{pair}\ 0\ 0)$$

  Intitiv vom avea $(0,0) \mapsto (0,1) \mapsto (1,2) \mapsto \cdots \mapsto (n-1, n)$
- definim $\mathbf{pred} = \lambda n.\ \mathbf{fst}\ (\mathbf{pred''}\ n)$

Definirea funcției **pred** folosind perechi

$$\mathbf{pred}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$$

$$\mathtt{S}'' = \lambda p.(\lambda x.\ \mathbf{pair}\ x(\mathtt{S}\ x))\ (\mathbf{snd}\ p)$$

$$\mathbf{pred}'' = \lambda n.n\ \mathtt{S}''\ (\mathbf{pair}\ 0\ 0)$$

$$\mathbf{pred} = \lambda n.\ \mathbf{fst}\ (\mathbf{pred}''\ n)$$

Definirea funcției **pred** folosind perechi

$$\mathbf{pred}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$$

$$\mathbf{S}'' = \lambda p.(\lambda x.\ \mathbf{pair}\ x (\mathbf{S}\ x))\ (\mathbf{snd}\ p)$$

$$\mathbf{pred}'' = \lambda n.n\ \mathbf{S}''\ (\mathbf{pair}\ 0\ 0)$$

$$\mathbf{pred} = \lambda n.\ \mathbf{fst}\ (\mathbf{pred}''\ n)$$

$\mathbf{pred}\ 2 \to_\beta \mathbf{fst}\ (\mathbf{pred}''\ 2) \to_\beta \mathbf{fst}\ (2\ \mathbf{S}''\ (\mathbf{pair}\ 0\ 0)) \to_\beta^2$
$\mathbf{fst}\ (\mathbf{S}''\ (\mathbf{S}''\ (\mathbf{pair}\ 0\ 0))) \to_\beta \mathbf{fst}\ (\mathbf{S}''\ (\mathbf{S}''\ (\mathbf{pair}\ 0\ 0))) \to_\beta$
$\mathbf{fst}\ (\mathbf{S}''\ ((\lambda x.\ \mathbf{pair}\ x (\mathbf{S}\ x))\ (\mathbf{snd}\ (\mathbf{pair}\ 0\ 0)))) \to_\beta^6$
$\mathbf{fst}\ (\mathbf{S}''\ ((\lambda x.\ \mathbf{pair}\ x (\mathbf{S}\ x))\ 0)) \to_\beta \mathbf{fst}\ (\mathbf{S}''\ (\mathbf{pair}\ 0 (\mathbf{S}\ 0))) \to_\beta^8$
$\mathbf{fst}\ (\mathbf{pair}\ (\mathbf{S}\ 0)(\mathbf{S}\ (\mathbf{S}\ 0))) \to_\beta^6 \mathbf{S}\ 0 \to_\beta^3 1$

Recursie

- există termeni care **nu** pot fi reduși la o β-formă normală, de exemplu
  $(\lambda x.xx)(\lambda x.xx) \to_\beta (\lambda x.xx)(\lambda x.xx)$
  În λ-calcul putem defini calcule infinite!

Recursie

- există termeni care **nu** pot fi reduși la o β-formă normală, de exemplu
  $(\lambda x.xx)(\lambda x.xx) \rightarrow_\beta (\lambda x.xx)(\lambda x.xx)$
  În λ-calcul putem defini calcule infinite!
  Dacă notăm $Af ::= \lambda x.f(xx)$ atunci
  $(Af)(Af) =_\beta (\lambda x.f(xx))(Af) =_\beta f((Af)(Af))$
  Dacă notăm $Yf ::= (Af)(Af)$ atunci $Yf =_\beta f(Yf)$.

Puncte fixe

- pentru o funcție $f : X \to X$ un **punct fix** este un element $x_0 \in X$ cu $f(x_0) = x_0$.
  - $f : \mathbb{N} \to \mathbb{N} f(x) = x + 1$ nu are puncte fixe
  - $f : \mathbb{N} \to \mathbb{N} f(x) = 2x$ are punctul fix $x = 0$

Puncte fixe

- pentru o funcție $f : X \to X$ un **punct fix** este un element $x_0 \in X$ cu $f(x_0) = x_0$.
  - $f : \mathbb{N} \to \mathbb{N} f(x) = x + 1$ nu are puncte fixe
  - $f : \mathbb{N} \to \mathbb{N} f(x) = 2x$ are punctul fix $x = 0$
- În λ-calcul

$$\mathbf{Y} ::= \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

  are proprietatea că $Yf =_\beta f(Yf)$, deci Yf este un **punct fix** pentru f.

Puncte fixe

- pentru o funcție $f : X \to X$ un **punct fix** este un element $x_0 \in X$ cu $f(x_0) = x_0$.
  - $f : \mathbb{N} \to \mathbb{N} f(x) = x + 1$ nu are puncte fixe
  - $f : \mathbb{N} \to \mathbb{N} f(x) = 2x$ are punctul fix $x = 0$
- În λ-calcul

  $$\mathbf{Y} ::= \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

  are proprietatea că $Yf =_\beta f(Yf)$, deci Yf este un **punct fix** pentru f.

  Y se numește **combinator de punct fix**.

Puncte fixe

- pentru o funcție $f : X \to X$ un **punct fix** este un element $x_0 \in X$ cu $f(x_0) = x_0$.
  - $f : \mathbb{N} \to \mathbb{N} f(x) = x + 1$ nu are puncte fixe
  - $f : \mathbb{N} \to \mathbb{N} f(x) = 2x$ are punctul fix $x = 0$
- În λ-calcul

$$\mathbf{Y} ::= \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

are proprietatea că $Yf =_\beta f(Yf)$, deci Yf este un **punct fix** pentru f.

Y se numește **combinator de punct fix.**

Avem $Yf =_\beta f(Yf) =_\beta f(f(YF)) =_\beta \ldots$

Putem folosi Y pentru a obține apeluri recursive!

Puncte fixe - funcția factorial

$\mathtt{fact} ::= \lambda n.\, \mathit{if}\ (\mathtt{isZero}\ \ n)\ \ \mathtt{one}\ \ (*\, n\, \mathit{fact}(\mathtt{pred}\ \ n))$

Puncte fixe - funcția factorial

$$\texttt{fact} ::= \lambda n.\ \mathit{if}\ (\texttt{isZero}\ \ n)\ \ \texttt{one}\ \ (*\, n\, \mathit{fact}(\texttt{pred}\ \ n))$$

Această definiție nu este corectă conform regulilor noastre, cum procedăm?

Puncte fixe - funcția factorial

$$\texttt{fact} ::= \lambda n.\ if\ (\mathbf{isZero}\ \ n)\ \ \mathbf{one}\ \ (*\, n\, fact(\mathbf{pred}\ \ n))$$

Această definiție nu este corectă conform regulilor noastre, cum procedăm?

- Pasul 1: abstractizăm, astfel încât construcția să fie corectă

$$\texttt{factA} ::= \lambda f \lambda n.\ if\ (\mathbf{isZero}\ \ n)\ \ \mathbf{one}\ \ (*\, n\, (f(\mathbf{pred}\ \ n)))$$

Puncte fixe - funcția factorial

$$\texttt{fact} ::= \lambda n.\ if\ (\texttt{isZero}\ \ n)\ \ \texttt{one}\ \ (*\, n\, fact(\texttt{pred}\ \ n))$$

Această definiție nu este corectă conform regulilor noastre, cum procedăm?

- Pasul 1: abstractizăm, astfel încât construcția să fie corectă

  $$\texttt{factA} ::= \lambda f \lambda n.\ if\ (\texttt{isZero}\ \ n)\ \ \texttt{one}\ \ (*\, n\, (f(\texttt{pred}\ \ n)))$$

- Pasul 2: aplicăm combinatorul de punct fix

  $$\texttt{fact} ::= Y\ \ \texttt{factA}$$

Puncte fixe - funcția factorial

$$\mathtt{fact} ::= \lambda n.\ if\ (\mathtt{isZero}\ \ n)\ \ \mathtt{one}\ \ (*\, n\, fact(\mathtt{pred}\ \ n))$$

Această definiție nu este corectă conform regulilor noastre, cum procedăm?

- Pasul 1: abstractizăm, astfel încât construcția să fie corectă

  $$\mathtt{factA} ::= \lambda f \lambda n.\ if\ (\mathtt{isZero}\ \ n)\ \ \mathtt{one}\ \ (*\, n\, (f(\mathtt{pred}\ \ n)))$$

- Pasul 2: aplicăm combinatorul de punct fix

  $$\mathtt{fact} ::= Y\ \mathtt{factA}$$

  Deoarece $Y\ \mathtt{factA} =_\beta \mathtt{factA}\, (Y\ \mathtt{factA})$ obținem

$$\mathtt{fact} =_\beta \lambda n.\ if\ (\mathtt{isZero}\ \ n)\ \ \mathtt{one}\ \ (*\, n\, (\mathtt{fact}\, (\mathtt{pred}\ \ n)))$$

Puncte fixe - funcția factorial

$\texttt{fact zero} =_\beta (Y\ \texttt{factA})\ \texttt{zero}$

$=_\beta \texttt{factA}\ (Y\ \texttt{factA})\ \texttt{zero}$

$=_\beta\ \textit{if}\ (\texttt{isZero zero})\ \texttt{one}\ (*\ \texttt{zero}\ ((Y\ \texttt{factA})(\texttt{pred zero})))$

$=_\beta \texttt{one}$

Puncte fixe - funcția factorial

$\mathbf{fact}\ \mathbf{zero} =_\beta (Y\ \mathbf{factA})\ \mathbf{zero}$
$=_\beta \mathbf{factA}\ (Y\ \mathbf{factA})\ \mathbf{zero}$
$=_\beta$ *if* $(\mathbf{isZero}\ \mathbf{zero})\ \mathbf{one}\ (*\ \mathbf{zero}\ ((Y\ \mathbf{factA})(\mathbf{pred}\ \mathbf{zero})))$
$=_\beta \mathbf{one}$

$\mathbf{fact}\ \mathbf{one} =_\beta (Y\ \mathbf{factA})\ \mathbf{one}$
$=_\beta \mathbf{factA}\ (Y\ \mathbf{factA})\ \mathbf{one}$
$=_\beta$ *if* $(\mathbf{isZero}\ \mathbf{one})\ \mathbf{one}\ (*\ \mathbf{one}\ ((Y\ \mathbf{factA})(\mathbf{pred}\ \mathbf{one})))$
$=_\beta *\ \mathbf{one}\ ((Y\ \mathbf{factA})(\mathbf{pred}\ \mathbf{one}))$
$=_\beta \ldots$

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

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

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

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

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

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

$$\texttt{tail} ::= \lambda l.\ \mathbf{fst}\ (l\ (\lambda x\ p.\ \mathbf{pair}\ (\mathbf{snd}\ p)\ (\mathbf{cons}\ x\ (\mathbf{snd}\ p)))\ (\mathbf{pair}\ \mathbf{null}\ \mathbf{null}))$$

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

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

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

$\mathbf{cons} ::= \lambda x\, l.\, \mathbf{pair}\ \ \mathit{false}\ (\mathbf{pair}\ \ x\ l)$

Liste ca perechi

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

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

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

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

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

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

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

Pe săptămâna viitoare!