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

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

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

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 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` ::= $\lambda c\ then\ else.c\ then\ else$ — 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

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

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

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
$\mathbf{if}\ false\ (\lambda x.x\ x)\ (\lambda x.x) \rightarrow_\beta^3$
$false\ (\lambda x.x\ x)\ (\lambda x.x) \rightarrow_\beta^2 \lambda x.x$

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

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

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

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** 3 5 $\rightarrow_{\beta}^{2} \lambda f.f\ 3\ 5$

perechea $(3, 5)$ reprezintă capabilitatea de a aplica o funcție de două argumente lui 3 si apoi lui 5.

Operații pe perechi

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

$$\mathbf{pair}\ xy \equiv_\beta f\ x\ y$$

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

$\mathbf{fst}\ (\mathbf{pair}\ 3\ 5) \to_\beta \mathbf{pair}\ 3\ 5\ \mathit{true} \to_\beta^3 \mathit{true}\ 3\ 5 \to_\beta^2 3$

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

$\mathbf{snd}\ (\mathbf{pair}\ 3\ 5) \to_\beta \mathbf{pair}\ 3\ 5\ \mathit{false} \to_\beta^3 \mathit{false}\ 3\ 5 \to_\beta^2 5$

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 = $ *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 $\lambda n\ s\ z.n\ s\ (sz)$

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

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

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

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

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

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

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

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

$\mathbf{exp} ::= \lambda m\ n.n\ (*\ m)\ 1$ sau $\lambda m\ n.n\ m$

$\mathbf{exp}\ 3\ 2$

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 $\lambda n\ s\ z.n\ s\ (sz)$

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

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

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

$\mathbf{exp} ::= \lambda m\ n.n\ (*\ m)\ 1$ sau $\lambda m\ n.n\ m$

$\mathbf{exp}\ 3\ 2 \rightarrow^2_\beta 2\ 3 \rightarrow^2_\beta \lambda z.3(3\ z) \rightarrow_\beta$
$\lambda z.\lambda z'.3\ z(3\ z(3\ z\ z')) \equiv_\alpha \lambda s\ z.3\ s(3\ s(3\ s\ z)) \rightarrow^6_\beta$
$\lambda s\ z.s(s(s(s(s(s(s(s(s\ z)))))))) = 9$

Scăderi și comparații

Presupunem că avem o funcție predecesor $\mathbf{P}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$

Scăderi și comparații

Presupunem că avem o funcție predecesor $\mathbf{P}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$

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

Scăderi și comparații

Presupunem că avem o funcție predecesor $\mathbf{P}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$

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

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

Scăderi și comparații

Presupunem că avem o funcție predecesor $\mathbf{P}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$

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

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

$$\texttt{<=} ::= \lambda m\ n.\ \texttt{?0=}\ (\texttt{-}\ m\ n)$$

Scăderi și comparații

Presupunem că avem o funcție predecesor $\mathbf{P}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$

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

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

$\texttt{<=} ::= \lambda m\ n.\ \texttt{?0=}\ (\texttt{-}\ m\ n)$

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

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

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

Scăderi și comparații

Presupunem că avem o funcție predecesor $\mathbf{P}\ x = \begin{cases} 0 & x = 0 \\ x-1 & x \neq 0 \end{cases}$

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

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

$$\texttt{<=} ::= \lambda m\ n.\ \texttt{?0=}\ (\texttt{-}\ m\ n)$$

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

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

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

$$\texttt{=} ::= \lambda m\ n.\ \texttt{and}\ (\texttt{<=}\ m\ n)\ (\texttt{>=}\ m\ n)$$

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

Problemă

Cum definim funcția $\mathbf{P}$?

Definirea funcției P folosind perechi

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

Definirea funcției $\mathbf{P}$ folosind perechi

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

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

Definirea funcției $\mathbf{P}$ folosind perechi

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

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

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

Definirea funcției $\mathtt{P}$ folosind perechi

$$\mathtt{P}\ x = \begin{cases} 0 & x = 0 \\ x - 1 & x \neq 0 \end{cases}$$

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

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

$$\mathtt{P} = \lambda n.\ \mathtt{fst}\ (\mathtt{P}''\ n)$$

Definirea funcției $\mathbf{P}$ folosind perechi

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

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

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

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

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

Funcția predecesor — definiție alternativă directă

Idee 1: $\mathbf{P} ::= \lambda n.\ \mathbf{?0=}\ n\ 0\ (\mathbf{P'}\ n)$ — am tratat primul caz. acum vrem o funcție $\mathbf{P'}$ care calculeaza $n - 1$ dacă $n \neq 0$

Idee 2: folosim iterația $\mathbf{P'} ::= \lambda n.n\ \mathbf{S'}\ Z'$, unde

- $\mathbf{S'}$ e un fel de succesor
- Z' e un fel de -1, i.e. $\mathbf{S'}\ Z' = 0$

$\mathbf{S'} ::= \lambda n.n\ \mathbf{S}\ 1$

$Z' ::= \lambda s\ z.0$ — Z' nu e codificarea unui număr
Dar se verifică că $\mathbf{S'}\ Z' \to_\beta Z'\ \mathbf{S}\ 1 \to_\beta^2 0$

Totul e OK deoarece $\mathbf{P'}$ e aplicată **doar** pe numere diferite de 0.

$\mathbf{P} ::= \lambda n.\ \mathbf{?0=}\ n\ 0\ (n\ (\lambda n.n\ \ \mathbf{S}\ 1)\ (\lambda s\ z.0))$

Liste

Intuiție: Capabilitatea de a agrega o listă

Codificare: O funcție cu 2 argumente
funcția de agregare și valoarea inițială

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.
Pentru aceste liste, operația de bază este `foldr`.

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

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

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

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

Operații pe liste

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

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

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

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

primul element al listei, sau d dacă lista e vidă

Operații pe liste

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

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

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

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

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

$\mathbf{foldr} ::= \lambda f\ i\ l.l\ f\ i$

$\mathbf{map} ::= \lambda f\ l.l\ (\lambda x\ t.\ \mathbf{cons}\ (f\ x)\ t)\ \ \mathbf{null}$

$\mathbf{filter} ::= \lambda p\ l.l\ (\lambda x\ t.p\ x\ (\mathbf{cons}\ x\ t)\ t)\ \ \mathbf{null}$

Factorial, Fibonacci, împărțire folosind perechi

fact ::= $\lambda n.\ \mathbf{snd}\ (n\ (\lambda p.\ \mathbf{pair}\ (\mathbf{S}\ (\mathbf{fst}\ p))(\mathbf{*}\ (\mathbf{fst}\ p)(\mathbf{snd}\ p)))\ (\mathbf{pair}\ 1\ 1))$

Factorial, Fibonacci, împărțire folosind perechi

$\texttt{fact} ::= \lambda n.\, \texttt{snd}\, (n\, (\lambda p.\, \texttt{pair}\, (\texttt{S}\, (\texttt{fst}\, p))(\texttt{*}\, (\texttt{fst}\, p)(\texttt{snd}\, p)))\, (\texttt{pair}\, 1\, 1))$

$\texttt{fib} ::= \lambda n.\, \texttt{fst}\, (n\, (\lambda p.\, \texttt{pair}\, (\texttt{snd}\, p)(+\, (\texttt{fst}\, p)\, (\texttt{snd}\, p)))\, (\texttt{pair}\, 0\, 1))$

Factorial, Fibonacci, împărțire folosind perechi

fact ::= $\lambda n.\ \mathbf{snd}\ (n\ (\lambda p.\ \mathbf{pair}\ (\mathbf{S}\ (\mathbf{fst}\ p))(*\ (\mathbf{fst}\ p)(\mathbf{snd}\ p)))\ (\mathbf{pair}\ 1\ 1))$

fib ::= $\lambda n.\ \mathbf{fst}\ (n\ (\lambda p.\ \mathbf{pair}\ (\mathbf{snd}\ p)(+\ (\mathbf{fst}\ p)\ (\mathbf{snd}\ p)))\ (\mathbf{pair}\ 0\ 1))$

divMod ::= $\lambda m\ n.m\ (\lambda p. >\ n\ (\mathbf{snd}\ p)\ p\ (\mathbf{pair}\ (\mathbf{S}\ (\mathbf{fst}\ p))\ (-\ (\mathbf{snd}\ p)\ n)))\ (\mathbf{pair}\ 0\ m)$

Factorial, Fibonacci, împărțire folosind perechi

fact ::= $\lambda n.\, \mathbf{snd}\ (n\ (\lambda p.\, \mathbf{pair}\ (\mathbf{S}\ (\mathbf{fst}\ p))(*\ (\mathbf{fst}\ p)(\mathbf{snd}\ p)))\ (\mathbf{pair}\ 1\ 1))$

fib ::= $\lambda n.\, \mathbf{fst}\ (n\ (\lambda p.\, \mathbf{pair}\ (\mathbf{snd}\ p)(+\ (\mathbf{fst}\ p)\ (\mathbf{snd}\ p)))\ (\mathbf{pair}\ 0\ 1))$

divMod ::= $\lambda m\ n.m\ (\lambda p. >\ n\ (\mathbf{snd}\ p)\ p\ (\mathbf{pair}\ (\mathbf{S}\ (\mathbf{fst}\ p))\ (-\ (\mathbf{snd}\ p)\ n)))\ (\mathbf{pair}\ 0\ m)$

Observații

- Nu toate funcțiile pot fi definite prin iterare fixată
- Pentru **divMod** obținem răspunsul (de obicei) din mult mai puțin de *m* iterații

Recursie

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

Recursie

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

Recursie

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

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 $x = 0$

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

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

  are proprietatea că $Y\ f =_\beta f\ (Y\ f)$, deci $Y\ f$ 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

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

  are proprietatea că $Y\ f =_\beta f\ (Y\ f)$, deci $Y\ f$ 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

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

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

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

  Avem $Y\ f =_\beta f\ (Y\ f) =_\beta f\ (f\ (Y\ f)) =_\beta \ldots$

  Putem folosi Y pentru a obține apeluri recursive!

Puncte fixe - funcția factorial

$\texttt{fact} ::= \lambda n.\, \texttt{if}\ (\texttt{?0=}\ n)\ 1\ (\texttt{*}\ n\ (fact\ (\texttt{P}\ \ n)))$

Puncte fixe - funcția factorial

$$\mathtt{fact} ::= \lambda n.\, \mathbf{if}\ (\mathbf{?0=}\ n)\ 1\ (\mathbf{*}\ n\ (\mathit{fact}\ (\mathbf{P}\ \ n)))$$

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

Puncte fixe - funcția factorial

$$\texttt{fact} ::= \lambda n.\ \texttt{if}\ (\texttt{?0=}\ n)\ 1\ (\texttt{*}\ n\ (\mathit{fact}\ (\texttt{P}\ \ n)))$$

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

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

  $$\texttt{factA} ::= \lambda f.\lambda n.\ \texttt{if}\ (\texttt{?0=}\ n)\ 1\ (\texttt{*}\ n\ (f\ (\texttt{P}\ \ n)))$$

Puncte fixe - funcția factorial

$$\texttt{fact} ::= \lambda n.\,\mathbf{if}\,(\mathbf{?0=}\ n)\ 1\ (\mathbf{*}\ n\ (\mathit{fact}\ (\mathbf{P}\ \ n)))$$

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

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

  $$\texttt{factA} ::= \lambda f.\lambda n.\,\mathbf{if}\,(\mathbf{?0=}\ n)\ 1\ (\mathbf{*}\ n\ (f\ (\mathbf{P}\ \ n)))$$

- Pasul 2: aplicăm combinatorul de punct fix

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

Puncte fixe - funcția factorial

$$\texttt{fact} ::= \lambda n.\,\texttt{if}\,(\texttt{?0=}\ n)\ 1\ (\texttt{*}\ n\ (\mathit{fact}\ (\mathbf{P}\ \ n)))$$

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

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

$$\texttt{factA} ::= \lambda f.\lambda n.\,\texttt{if}\,(\texttt{?0=}\ n)\ 1\ (\texttt{*}\ n\ (f\ (\mathbf{P}\ \ n)))$$

- Pasul 2: aplicăm combinatorul de punct fix

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

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

$$\texttt{fact} =_\beta \lambda n.\,\texttt{if}\,(\texttt{?0=}\ n)\ 1\ (\texttt{*}\ n\ (\ \mathit{fact}\ (\mathbf{P}\ \ n))$$

divMod — definiție recursivă

Definiția primitiv recursivă (fără Y)

$\texttt{divMod} ::= \lambda m\, n.m\ (\lambda p. > n\ (\textbf{snd}\ p)\ p\ (\textbf{pair}\ (\textbf{S}\ (\textbf{fst}\ p))\ (\textbf{-}\ (\textbf{snd}\ p)\ n)))\ (\textbf{pair}\ 0\ m)$

Definiția recursivă (incorectă)

$\texttt{divMod} ::= \lambda m\, n.\ \textbf{?0=}\ n\ (\textbf{pair}\ 0\ m)\ (\textbf{divMod}'\ \ 0\ m)$ where

$\texttt{divMod'} ::= \lambda q\, r. > n\ r\ (\textbf{pair}\ q\ r)\ (\textbf{divMod}'\ \ (\textbf{S}\ q)(\textbf{-}\ m\ n))$

Definiția recursivă (corectă, folosind Y)

$\texttt{divMod} ::= \lambda m\, n.\ \textbf{?0=}\ n\ \ (\textbf{pair}\ 0\ m)$
$(\textbf{Y}\ (\lambda f.\lambda q\, r. > n\ r\ (\textbf{pair}\ q\ r)\ (f\ (\textbf{S}\ q)\ (\textbf{-}\ m\ n)))$
$0\ m)$

Apel prin valoare (Call by Value)

Pentru a reduce $e_1 \; e_2$:

- Reducem e_1 până la o funcție $\lambda x.e$
- Apoi reducem e_2 până la o valoare v
- Apoi reducem $(\lambda x.e) \; v$ la $[v/x]e$

Nu simplificăm sub λ

Apel prin valoare (Call by Value)

Pentru a reduce $e_1 \;\; e_2$:

- Reducem e_1 până la o funcție $\lambda x.e$
- Apoi reducem e_2 până la o valoare v
- Apoi reducem $(\lambda x.e) \;\; v$ la $[v/x]e$

Nu simplificăm sub λ

$$\texttt{?0=}\; 0\ (+\ 2\ 1)\ (+\ 3\ 4)$$
$$\to_{\beta(V)}\ 0\ (\lambda x.false)\ true\ (+\ 2\ 1)\ (+\ 3\ 4)$$
$$\to^{2}_{\beta(V)}\ true\ (+\ 2\ 1)\ (+\ 3\ 4)$$
$$\to^{2}_{\beta(V)}\ true\ (\lambda s\ z.2\ s\ (1\ s\ z))\ (+\ 3\ 4)$$
$$\to_{\beta(V)}\ (\lambda f.\lambda s\ z.2\ s\ (1\ s\ z))\ (+\ 3\ 4)$$
$$\to^{2}_{\beta(V)}\ (\lambda f.\lambda s\ z.2\ s\ (1\ s\ z))\ (\lambda s\ z.3\ s\ (4\ s\ z))$$
$$\to_{\beta(V)}\ \lambda s\ z.2\ s\ (1\ s\ z)$$
$$\not\to_{\beta(V)}$$

$$\texttt{?0=} ::= (\lambda n.n\ (\lambda x.false)\ true)$$
$$0 ::= (\lambda s\ z.z)$$
$$+ ::= (\lambda m\ n\ s\ z.m\ s\ (n\ s\ z))$$
$$true ::= \lambda t\ f.t$$

Apel prin nume (Limbaje pur funcționale)

Pentru a reduce $e_1 \; e_2$:

- Reducem e_1 până la o funcție $\lambda x.e$
- Apoi reducem $(\lambda x.e) \; e_2$ la $[e_2/x]e$

Nu simplificăm sub λ nici în dreapta aplicației

Apel prin nume (Limbaje pur funcționale)

Pentru a reduce $e_1 \;\; e_2$:

- Reducem e_1 până la o funcție $\lambda x.e$
- Apoi reducem $(\lambda x.e) \;\; e_2$ la $[e_2/x]e$

Nu simplificăm sub λ nici în dreapta aplicației

$$
\begin{aligned}
&\texttt{?0=}\ 0\ (+\ 2\ 1)\ (+\ 3\ 4)\\
\to_{\beta(N)}\ &0\ (\lambda x.\mathit{false})\ \mathit{true}\ (+\ 2\ 1)\ (+\ 3\ 4)\\
\to^2_{\beta(N)}\ &\mathit{true}\ (+\ 2\ 1)\ (+\ 3\ 4)\\
\to^2_{\beta(N)}\ &(+\ 2\ 1)\\
\to^2_{\beta(N)}\ &(\lambda s\ z.2\ s\ (1\ s\ z))\\
\not\to_{\beta(N)}\ &
\end{aligned}
$$

$$
\begin{aligned}
\texttt{?0=} &::= (\lambda n.n\ (\lambda x.\mathit{false})\ \mathit{true})\\
0 &::= (\lambda s\ z.z)\\
\mathit{true} &::= \lambda t\ f.t\\
+ &::= (\lambda m\ n\ s\ z.m\ s\ (n\ s\ z))
\end{aligned}
$$

Evaluare leneșă

Pentru a reduce $e_1 \; e_2$:

- Reduc e_1 până la o funcție $\lambda x.e$
- Apoi reduc corpul funcției e până la un e' care are nevoie de x
- Apoi reduc e_2 până la o valoare v
- Apoi reduc $(\lambda x.e') \; v$ la $[v/x]e'$

Evaluare nerestricționată

Pentru a reduce $e_1 \; e_2$

- Reducem fie e_1 fie e_2
- Putem reduce corpurile funcțiilor
- Oricând avem $(\lambda x.e')e''$, o putem (sau nu) reduce la $[e''/x]e'$
- Reduce până la o formă normală

Nu garantează găsirea unei forme normale

Evaluare „normală“

Pentru a reduce $e_1 \ \ e_2$

- Reducem mereu cel mai din stânga redex din cele de mai sus
- Reducem $(\lambda x.e')e''$ la $[e''/x]e'$
- Reducem e_1 (putem reduce și corpurile funcțiilor)
- Dacă $e_1 \not\rightarrow$, reducem e_2
- Reduce până la o formă normală
- Garantează găsirea unei forme normale

Apel prin valoare

Pentru a reduce $e_1 \; e_2$:

- Reducem e_1 până la o funcție $\lambda x.e$
- Apoi reducem e_2 până la o valoare v
- Apoi reducem $(\lambda x.e) \; v$ la $[v/x]e$

Reguli

$$\text{(V@S)} \quad \frac{e_1 \rightarrow_\beta e_1'}{e_1 \; e_2 \rightarrow_\beta e_1' \; e_2}$$

$$\text{(V@D)} \quad \frac{e_2 \rightarrow_\beta e_2'}{(\lambda x.e_1) \; e_2 \rightarrow_\beta (\lambda x.e_1) \; e_2'}$$

$$\text{(V@)} \quad (\lambda x.e_1) \; v_2 \rightarrow_\beta e \quad \textit{dacă } e = [v_2/x]e_1$$

Apel prin nume

Pentru a reduce $e_1 \;\; e_2$:

- Reducem e_1 până la o funcție $\lambda x.e$
- Apoi reducem $(\lambda x.e) \;\; e_2$ la $[e_2/x]e$

Reguli

$$\text{(N@S)} \quad \frac{e_1 \to_\beta e_1'}{e_1 \;\; e_2 \to_\beta e_1' \;\; e_2}$$

$$\text{(N@)} \quad (\lambda x.e_1) \;\; e_2 \to_\beta e \quad \textit{dacă } e = [e_2/x]e_1$$

Evaluare nerestricționată

Pentru a reduce $e_1 \ \ e_2$

- Reducem fie e_1 fie e_2
- Putem reduce corpurile funcțiilor
- Oricând avem $(\lambda x.e')e''$, o putem (sau nu) reduce la $[e''/x]e'$

Reguli

$$\text{(NR@S)} \quad \frac{e_1 \rightarrow e_1'}{e_1 \ \ e_2 \rightarrow e_1' \ \ e_2}$$

$$\text{(NR@D)} \quad \frac{e_2 \rightarrow e_2'}{e_1 \ \ e_2 \rightarrow e_1 \ \ e_2'}$$

$$\text{(NR}\textsc{Fun}\text{D)} \quad \frac{e \rightarrow e'}{\lambda x.e \rightarrow \lambda x.e'}$$

$$\text{(NR@)} \quad (\lambda x.e_1) \ \ e_2 \rightarrow e \quad \textit{dacă } e = [e_2/x]e_1$$

Evaluare „normală“

Pentru a reduce $e_1 \ \ e_2$

- Reducem mereu cel mai din stânga redex din cele de mai sus
- Reducem $(\lambda x.e')e''$ la $[e''/x]e'$
- Reducem e_1 (putem reduce și corpurile funcțiilor)
- Dacă $e_1 \not\twoheadrightarrow$, reducem e_2

Reguli

$$(\textsc{Nor@}) \quad (\lambda x.e_1) \ \ e_2 \to e \quad \textit{dacă } e = [e_2/x]e_1$$

$$(\textsc{Nor@S}) \quad \frac{e_1 \to e_1'}{e_1 \ \ e_2 \to e_1' \ \ e_2} \quad \textit{dacă } e_1 \text{ nu e încă funcție}$$

$$(\textsc{NorFunD}) \quad \frac{e \to e'}{\lambda x.e \to \lambda x.e'}$$

$$(\textsc{Nor@D}) \quad \frac{e_1 \not\twoheadrightarrow \qquad e_2 \to e_2'}{e_1 \ \ e_2 \to e_1 \ \ e_2'}$$

Evaluare leneșă

Pentru a reduce $e_1 \;\; e_2$:

- Reduc e_1 până la o funcție $\mathbf{fun}\ (x : T) \to e$
- Apoi reduc corpul funcției e până la un e' care are nevoie de x
- Apoi reduc e_2 până la o valoare v
- Apo reduc $(\lambda x.e') \;\; v$ la $[v/x]e'$

Reguli?

E mai complicat decât pare, deoarece trebuie să ne dăm seama că e' are nevoie de x.

Contexte de evaluare

- Găsirea redex-ului prin analiză sintactică
- Putem înlocui regulile structurale cu reguli gramaticale

Sintaxă: $e ::= x \mid \lambda x.e \mid e\ e$

Reguli structurale

$$\frac{e_1 \longrightarrow_\beta e_1'}{e_1\ e_2 \longrightarrow_\beta e_1'\ e_2}$$

$$\frac{e_2 \longrightarrow_\beta e_2'}{\langle(\lambda x.e_1)\ e_2 \longrightarrow_\beta (\lambda x.e_1)\ e_2'}$$

Contexte de evaluare

$$\begin{array}{rcl} c & ::= & \blacksquare \\ & \mid & c\ e \\ & \mid & (\lambda x.e)\ c \end{array}$$

Instanțierea unui context c cu expresia e

$$c[e] = [e/\blacksquare]c$$

Contexte de evaluare

Sintaxă: $e ::= x \mid \lambda x.e \mid e\ e$

Contexte: $c ::= \blacksquare \mid c\ e \mid (\lambda x.e)\ c$

Exemple de contexte

Corecte	Greșite
$\blacksquare$	5
$3\ \blacksquare$	$true\ x\ \blacksquare$

Exemple de contexte instanțiate

- $\blacksquare[x\ 1] = x\ 1$
- $(9\ (\blacksquare\ 7))[x] = 9\ (x\ 7)$
- $(9\ (\blacksquare\ 7))[5] = 9\ (5\ 7)$

Semantica Operațională Contextuală

Un pas de execuție folosind contexte de evaluare

- Descompune expresia în contextul c și redex-ul r
- Aplică o regulă operațională asupra lui r obținând e
- Pune e în contextul inițial, obținând $c[e]$

$$\frac{r \longrightarrow_\beta e}{c[r] \longrightarrow_\beta c[e]}$$

Evaluare leneșă folosind Semantica Contextuală

Contexte de evaluare pentru aplicație

$$
\begin{array}{rcl}
c & ::= & \blacksquare \\
& | & c\ e \\
& | & (\lambda x.c)\ e \\
& | & (\lambda x.c[x])\ c
\end{array}
$$

Regulă de evaluare pentru aplicație

$$(\lambda x.c[x])\ v \rightarrow (\lambda x.c[v])\ v$$