β-reduktsioon

- λ-termide väärtustamine toimub reduktsiooni reeglite korduva rakendamise kaudu.
- β-reduktsiooni reegel:

$$(\lambda x.\, e_1)\, e_2 \quad \to_\beta \quad e_1[x \mapsto e_2]$$

- Alamavaldist kujul $(\lambda x.\, e_1)\, e_2$ nimetatakse (β-)reedeksiks.
- NB! λ-termis võib olla palju reedekseid.
- Ilma (β-)reedeksiteta λ-term on (β-)normaalkujul.
- Näited:

$\lambda x.\, x\,(\lambda y.\, x)$	reedekseid ei ole (so. normaalkuju)
$\lambda x.\, \underline{(\lambda y.\, y)\, 3}$	üks reedeks
$\lambda f.\, f\, \underline{((\lambda x.\, x)\, 3)}\, \underline{((\lambda x.\, x)\, 4)}$	kaks reedeksit
$\underline{(\lambda f.\, f\, \underline{((\lambda x.\, x)\, 3)})\, (\lambda x.\, x)}$	kaks (kattuvat) reedeksit

δ-reduktsioon

Konstandid sisaldavad protsessori poolt toetatud

- täisarve ($\mathsf{Int} \subseteq \mathrm{C}$)
  - Näiteks: $0, 1, -1, 4, 8,$ jne.
- aritmetilisi operatsioone add, sub, mul
  - sub 9 (add 3 1) $\rightarrow_\delta$ sub 9 4 $\rightarrow_\delta$ 5
  - mul 0 9 $\rightarrow_\delta$ 0
- võrdlusoperatsiooni iszero
  - iszero 0 $\rightarrow_\delta$ true
  - iszero 8 $\rightarrow_\delta$ false

δ-reduktsioon

- $\{\mathsf{true}, \mathsf{false}, \mathsf{cond}\} \in C$

$$\mathsf{cond}\ \mathsf{true}\ e_1\ e_2 \to_\delta e_1$$
$$\mathsf{cond}\ \mathsf{false}\ e_1\ e_2 \to_\delta e_2$$

- $\{\mathsf{pair}, \mathsf{fst}, \mathsf{snd}\} \in C$

$$\mathsf{fst}\ (\mathsf{pair}\ e_1\ e_2) \to_\delta e_1$$
$$\mathsf{snd}\ (\mathsf{pair}\ e_1\ e_2) \to_\delta e_2$$

- $\{\mathsf{nil}, \mathsf{cons}, \mathsf{null}, \mathsf{hd}, \mathsf{tl}\} \in C$

$$\mathsf{null}\ \mathsf{nil} \to_\delta \mathsf{true}$$
$$\mathsf{null}\ (\mathsf{cons}\ e_1\ e_2) \to_\delta \mathsf{false}$$
$$\mathsf{hd}\ (\mathsf{cons}\ e_1\ e_2) \to_\delta e_1$$
$$\mathsf{tl}\ \mathsf{nil} \to_\delta \mathsf{nil}$$
$$\mathsf{tl}\ (\mathsf{cons}\ e_1\ e_2) \to_\delta e_2$$

η-reduktsioon

- η-reduktsiooni reegel:

$$\lambda x.\, e\, x \quad \rightarrow_\eta \quad e \qquad\qquad x \notin \mathrm{FV}(e)$$

- η-reduktsioon vastab funktsioonide ekstensionaalsuse omadusele s.o. funktsioonid f ja g on võrdsed, kui nad annavad võrdse tulemuse iga argumendi puhul.
- Lihtsamatel juhtudel väärtustamiseks η-reduktsiooni vaja ei lähe.

Reduktsiooni järjekorrad

- λ-termide normaalkujud on unikaalsed.
- Normaalkuju leidumine ei ole garanteeritud!
- Näide:

$$
\begin{aligned}
(\lambda x.\, x\, x)\,(\lambda x.\, x\, x) \quad &\to_\beta \quad (\lambda x.\, x\, x)\,(\lambda x.\, x\, x) \\
&\to_\beta \quad (\lambda x.\, x\, x)\,(\lambda x.\, x\, x) \\
&\to_\beta \quad (\lambda x.\, x\, x)\,(\lambda x.\, x\, x) \\
&\to_\beta \quad (\lambda x.\, x\, x)\,(\lambda x.\, x\, x) \\
&\ldots
\end{aligned}
$$

Reduktsiooni järjekorrad

- Normaaljärjekord: alati redutseerida vasakpoolne väline reedeks.

$$\underline{(\lambda x.\, y)((\lambda x.\, x\, x)(\lambda x.\, x\, x))} \;\to_\beta\; y$$

- Aplikatiivne järjekord: alati redutseerida vasakpoolne sisemine reedeks.

$$\begin{aligned} & (\lambda x.\, y)(\underline{(\lambda x.\, x\, x)(\lambda x.\, x\, x)}) \\ \to_\beta\; & (\lambda x.\, y)(\underline{(\lambda x.\, x\, x)(\lambda x.\, x\, x)}) \\ \ldots & \end{aligned}$$

- Normaliseerimisteoreem: kui λ-termil leidub normaalkuju, siis on see saavutatav normaaljärjekorraga.

Reduktsiooni järjekorrad

- Normaaljärjekord võib olla ebaefektiivne!
- Normaljärjekord:

$$\begin{aligned}
&\underline{(\lambda x.\, x\, x)\,((\lambda x.\, x)\,(\lambda x.\, x))} && \to_\beta && ((\underline{(\lambda x.\, x)\,(\lambda x.\, x)})\,((\lambda x.\, x)\,(\lambda x.\, x)) \\
& && \to_\beta && \underline{(\lambda x.\, x)\,((\lambda x.\, x)\,(\lambda x.\, x))} \\
& && \to_\beta && \underline{(\lambda x.\, x)\,(\lambda x.\, x)} \\
& && \to_\beta && \lambda x.\, x
\end{aligned}$$

- Aplikatiivne järjekord:

$$\begin{aligned}
&(\lambda x.\, x\, x)\,(\underline{(\lambda x.\, x)\,(\lambda x.\, x)}) && \to_\beta && \underline{(\lambda x.\, x\, x)\,(\lambda x.\, x)} \\
& && \to_\beta && \underline{(\lambda x.\, x)\,(\lambda x.\, x)} \\
& && \to_\beta && \lambda x.\, x
\end{aligned}$$

Identifitseeri reedeksid

$$(\lambda x.\ x)\ (\lambda y.\ \text{add}\ 1\ y)\ (\text{mul}\ 3\ 2)$$

Redutseeri normaalkujule

$$(\lambda x.\ x)\ (\lambda y.\ \text{add}\ 1\ y)\ (\text{mul}\ 3\ 2)$$

Redutseeri normaalkujule :

1. aplikatiivjärjekorras
2. normaaljärjekorras

$$(\lambda f\ x.\ f\ (\text{add}\ 1\ 2))\ \text{mul}\ (\text{add}\ 1\ 1)\ 3$$

Anonüümsed funktsioonid

- λ-arvutus: $\lambda\ x\ y\ z.\ e$
- Idrises: `\ a, b, c => e`
  (slaididel \ ja => asemel kirjutame λ ja $\Rightarrow$)

Näiteks:

```
liida : Int → Int → Int
liida = λ x, y ⇒ x+y
```

ja

```
viis : Int
viis = (λ x, y ⇒ x+y) 2 3
```

Kõrgemat järku funktsioon

... on funktsioon, mis võtab argumendiks või tagastab funktsiooni.

Näiteks map:

```
map : (a → b) → List a → List b
map f []     = []
map f (x::xs) = f x :: map f xs
```

Funktsiooni map illustreerib järgmine võrdus:

$$\texttt{map}\ f\ [x_1, x_2, \ldots, x_n] = [f\ x_1, f\ x_2, \ldots, f\ x_n]$$

Näide:

```
Main> map (λ x ⇒ 1.0 / x) [1,2,4,8]
[1.0, 0.5, 0.25, 0.125]
```

Kõrgemat järku funktsioon: `filter`

... näiteks `filter`:

```
filter : (a → Bool) → List a → List a
filter p []     = []
filter p (x::xs) = if p x then x :: filter p xs else filter p xs
```

Funktsiooni `filter` illustreerib paremini alternatiivne definitsioon:

```
filter : (a → Bool) → List a → List a
filter p xs = [x | x←xs, p x]
```

`filter p xs` jätab listi alles tingimusele `p` vastavad elementid.

```
Main> filter (λx ⇒ x `mod` 2 == 0) [1..10]
[2, 4, 6, 8, 10]
```

Kõrgemat järku funktsioon: foldr

```
foldr : (a → b → b) → b → List a → b
foldr f b []      = b
foldr f b (x::xs) = f x (foldr f b xs)
```

Funktsiooni foldr illustreerib järgmine võrdus:

$$\texttt{foldr}\ (+)\ b\ [x_1, x_2, \ldots, x_n] = x_1 + (x_2 + (\ldots + (x_n + b)))$$

Funktsiooni map saab defineerida läbi foldr-i

```
map : (a → b) → List a → List b
map f xs = foldr g [] xs
    where g : a → List b → List b
          g x ys = f x :: ys
```

ehk

$$\begin{aligned}\texttt{foldr}\ g\ []\ [x_1, x_2, \ldots, x_n] &= x_1\texttt{`}g\texttt{`}(x_2\texttt{`}g\texttt{`}(\ldots\texttt{`}g\texttt{`}(x_n\texttt{`}g\texttt{`}[]))) = \\ &= f\ x_1 :: f\ x_2 :: \ldots :: f\ x_n :: [] = \\ &= [f\ x_1, f\ x_2, \ldots, f\ x_n]\end{aligned}$$

Kõrgemat järku funktsioon: foldl

```
foldl : (b → a → b) → b → List a → b
foldl f b []     = b
foldl f b (x::xs) = foldl f (f b x) xs
```

Funktsiooni foldl illustreerib järgmine võrdus:

$$\texttt{foldl}\ (+)\ b\ [x_1, x_2, \ldots, x_n] = (((b + x_1) + x_2) + \ldots) + x_n$$

Funktsiooni reverse saab defineerida läbi foldl-i

```
reverse : List a → List a
reverse xs = foldl g [] xs
   where g : List a → a → List a
         g x y = y :: x
```

ehk

$$\begin{aligned}\texttt{foldl}\ g\ []\ [x_1, x_2, \ldots, x_n] &= ((([]\ `g`\ x_1)\ `g`\ x_2)\ `g`\ \ldots)\ `g`\ x_n = \\ &= x_n :: \ldots :: x_2 :: x_1 :: [] = \\ &= [x_n, \ldots, x_2, x_1]\end{aligned}$$

Funktsiooni osaline rakendamine

- Selle asemel, et siduda kõik argumendid muutujatega

```
uusFunktsioon x y z = olemasolevFunktsioon (x+1) y z
```

  saame Idrises võtta osa argumente ja tagastada funktsiooni

```
uusFunktsioon x = olemasolevFunktsioon (x+1)
```

- Pane tähele, kuidas töötavad koos aplikatsiooni vasakassotsiatiivsus ja funktsiooni tüübi paremassotsiatiivsus.

```
f : a → (b → (c → d))
f   x :   b → (c → d)
(f  x)    y :   c → d
((f x)    y)    z :  d
```

ehk

```
f : a → b → c → d
f   x :  b → c → d
f   x    y :  c → d
f   x    y    z :  d
```

- funktsioone saame `(.)`-ga komponeerida, ehk:

```
f xs = sum (takeWhile (≠0) xs)
```

  asemel

```
f = sum . takeWhile (≠0)
```

- Funktsiooni `(a,b) → c` *karritud* kuju on `a → b → c`.

Defineeri funktsioon

```
takeWhile : (a→Bool) → List a → List a
takeWhile p = foldr ?f []
```

Näiteks:

```
Main> takeWhile (<10) [4,7,9,10,14,3]
[4, 7, 9]
Main> takeWhile (≠'x') ['a','b','x','c']
['a', 'b']
```

Mis on tulemus?

```
map (λ f ⇒ f 2) [ (*2), (+3), div 6]
```

funktsioon

```
(.) : (a → b) → (c → a) → c → b
(f . g) x = f (g x)

g : List Int → List Int
g xs = map (+1) (map (+2) xs)
```

Kirjuta funktsioon g ümber nii, et see ei kasutakse formaalset parameetrit vaid funktsioonide kompositsiooni.

Programmeerime midagi

- binaararvud listidega
- aasta kalender