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

β-reduktsioon

Ühesammuline β-reduktsioon:

$$\frac{}{(\lambda x.\ e_1)\ e_2 \to_\beta e_1[x \mapsto e_2]}$$

$$\frac{e_1 \to_\beta e_2}{e_1\ e_0 \to_\beta e_2\ e_0} \qquad \frac{e_1 \to_\beta e_2}{e_0\ e_1 \to_\beta e_0\ e_2}$$

$$\frac{e_1 \to_\beta e_2}{\lambda x.e_1 \to_\beta \lambda x.e_2}$$

β-reduktsioon

- Mitmesammuline β-reduktsioon:

$$\frac{e_1 \to_\beta e_2}{e_1 \twoheadrightarrow_\beta e_2} \qquad \frac{}{e \twoheadrightarrow_\beta e} \qquad \frac{e_1 \twoheadrightarrow_\beta e_2 \quad e_2 \twoheadrightarrow_\beta e_3}{e_1 \twoheadrightarrow_\beta e_3}$$

- NB! Antud definitsioon ei määra reduktsioonide järjekorda.
- Näide:

$$\begin{array}{lcl}
\underline{(\lambda f.\, f\,((\lambda x.\, x)\,3))\;(\lambda x.\, x)} & \to_\beta & \underline{(\lambda x.\, x)\,((\lambda x.\, x)\,3))} \\
 & \to_\beta & \underline{(\lambda x.\, x)\,3} \\
 & \to_\beta & 3 \\
(\lambda f.\, f\,\underline{((\lambda x.\, x)\,3)})\;(\lambda x.\, x) & \to_\beta & \underline{(\lambda f.\, f\,3)\;(\lambda x.\, x)} \\
 & \to_\beta & \underline{(\lambda x.\, x)\,3} \\
 & \to_\beta & 3
\end{array}$$

β-reduktsioon

- β-konversioon:

$$\frac{e_1 \twoheadrightarrow_\beta e_2}{e_1 =_\beta e_2} \qquad \frac{e_2 =_\beta e_1}{e_1 =_\beta e_2} \qquad \frac{e_1 =_\beta e_2 \quad e_2 =_\beta e_3}{e_1 =_\beta e_3}$$

- β-ekspansioon:

$$\frac{e_2 \twoheadrightarrow_\beta e_1}{e_1 \twoheadleftarrow_\beta e_2}$$

η-reduktsioon

- η-reduktsiooni reegel:

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

- η-reduktsioon vastab funktsioonide ekstensionaalsuse omadusele

Reduktsiooni järjekorrad

- Reduktsiooni järjekord ei oma tähtsust!
- Church-Rosseri teoreem: iga λ-termi e_0, e_1 ja e_2 korral, kui $e_0 \twoheadrightarrow_\beta e_1$ ja $e_0 \twoheadrightarrow_\beta e_2$, siis leidub e_3 selline et $e_1 \twoheadrightarrow_\beta e_3$ ja $e_2 \twoheadrightarrow_\beta e_3$.
- Järeldus: λ-termide normaalkujud on unikaalsed.
- NB! Normaalkuju leidumine ei ole garanteeritud!
- Näide:

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

Reduktsiooni järjekorrad

- Reduktsiooni järjekord on oluline!
- 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.

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

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

Reduktsiooni järjekorrad

- Normaaljärjekord võib olla ebaefektiivne!
- Normaljärjekord:

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

- Aplikatiivne järjekord:

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

- Laisk väärtustamine = normaaljärjekord + graafireduktsioon

Lahendus 1: `foldl`, kasutades `foldr`-i

Proovime leida `?h` ja `?j` et kehtiks: `foldl f a xs = foldr ?h ?j xs`

Vaatame juhtu `xs = []`:

```
foldl f a [] = foldr ?h ?j []
           a = ?j
```

ok, võtame `?j = a`

Vaatame juhtu `xs = [x]`:

```
foldl f a [x] = foldr ?h a [x]
        f a x = ?h x a
```

ok?, võtame `?h u v = f v u`

Vaatame juhtu `xs = [x,y]`:

```
foldl f a [x,y] = foldr ?h a [x,y]
    f (f a x) y = ?h x (?h y a)
```

fail! `?h u v = f v u` ei kehti: `a` läheb valesse kohta.

Teine katse: `foldl f a xs = foldr ?h ?j xs a`

Vaatame juhtu `xs = []`:

```
foldl f a [] = foldr ?h ?j [] a
           a = ?j a
```

ok, võtame `?j = (λ x ⇒ x) = id`

Vaatame juhtu `xs = [x]`:

```
foldl f a [x] = foldr ?h id [x] a
        f a x = ?h x id a
```

ok?, võtame `?h u g v = f v u`

Vaatame juhtu `xs = [x,y]`:

```
foldl f a [x,y] = foldr ?h id [x,y] a
    f (f a x) y = ?h x (?h y id) a
```

ok, võtame `?h u g v = g (f v u)`

Vaatame juhtu `xs = [x,y,z]`:

```
foldl f a [x,y,z] = foldr ?h2 id [x,y,z] a
f (f (f a x) y) z = ?h2 x (?h2 y (?h2 z id)) a
f (f (f a x) y) z = ?h2 y (?h2 z id) (f a x)
f (f (f a x) y) z = ?h2 z id (f (f a x) y)
f (f (f a x) y) z = id (f (f (f a x) y) z)
```

Lahendus 2: Kasutame mustkunsti!

- Võtame `foldl`-i definitsiooni ja alustame transmogrifitseerimist!!

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

- Kasutame rekursiooni jaoks abifunktsiooni nimega `fold`!

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

- Kasutame rekursiooni jaoks abifunktsiooni nimega `fold`!

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold f a xs    where
    fold : (a → e → a) → a → List e → a
    fold f a []       = a
    fold f a (x::xs) = fold f (f a x) xs
```

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold f a xs    where
    fold : (a → e → a) → a → List e → a
    fold f a []       = a
    fold f a (x::xs) = fold f (f a x) xs
```

- `fold`-s võimne `f`-d kustutada – selle väärtuse töö käigus ei muutu!

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold a xs    where
    fold : a → List e → a
    fold a []       = a
    fold a (x::xs) = fold (f a x) xs
```

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold a xs    where
    fold : a → List e → a
    fold a []       = a
    fold a (x::xs) = fold (f a x) xs
```

- Vahetame `fold`-i argumentide järjekorra!

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a    where
    fold : List e → a → a
    fold [] a       = a
    fold (x::xs) a = fold xs (f a x)
```

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold [] a      = a
    fold (x::xs) a = fold xs (f a x)
```

- Identsusfunktsioon `id` on tore! Kasutame seda ka.

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold [] a      = id a
    fold (x::xs) a = fold xs (f a x)
```

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold [] a      = id a
    fold (x::xs) a = fold xs (f a x)
```

- Tekitame segadust ja toome rekursiooni `fold xs` lambda argumendiks!

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold [] a      = id a
    fold (x::xs) a = (λ g ⇒ g (f a x)) (fold xs)
```

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold [] a      = id a
    fold (x::xs) a = (λ g ⇒ g (f a x)) (fold xs)
```

- Lambda on igavalt kirja pandud! Üldistame seda!

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold [] a      = id a
    fold (x::xs) a = (λ x,g,a⇒ g (f a x)) x (fold xs) a
```

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold [] a      = id a
    fold (x::xs) a = (λ x,g,a⇒ g (f a x)) x (fold xs) a
```

- Voh! Nüüd on a-d võrduste paremal pool – eemaldame need!

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold []      = id
    fold (x::xs) = (λ x,g,a⇒ g (f a x)) x (fold xs)
```

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold xs a   where
    fold : List e → a → a
    fold []       = id
    fold (x::xs) = (λ x,g,a⇒ g (f a x)) x (fold xs)
```

- Super! Nüüd saame `id` ja lambda tuua fold-i argumentideks!

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold (λ x,g,a⇒ g (f a x)) id xs a where
    fold : (e→(a→a)→a→a)→(a→a)→List e→a→a
    fold g h []       = h
    fold g h (x::xs) = g x (fold g h xs)
```

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = fold (λ x,g,a⇒ g (f a x)) id xs a where
    fold : (e→(a→a)→a→a)→(a→a)→List e→a→a
    fold g h []       = h
    fold g h (x::xs) = g x (fold g h xs)
```

- Oih! See `fold` on ju `foldr`.

```
foldl : (a → e → a) → a → List e → a
foldl f a xs = foldr (λ x,g,a⇒ g (f a x)) id xs a
```

- Tadaa!