Substitutsioon

Substitutsiooni definitsioon:

$$
\begin{aligned}
y[x \mapsto e] &= \begin{cases} e & \text{if } x = y \\ y & \text{otherwise} \end{cases} \\
(e_1\, e_2)[x \mapsto e] &= (e_1[x \mapsto e])\ (e_2[x \mapsto e]) \\
(\lambda y.\, e_1)[x \mapsto e] &= \begin{cases} \lambda y.\, e_1 & \text{if } x = y \\ \lambda y.\, e_1[x \mapsto e] & \text{if } y \notin \mathrm{FV}(e) \\ \lambda z.\, e_1[y \mapsto z][x \mapsto e] & \text{otherwise} \end{cases}
\end{aligned}
$$

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:

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

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:

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

- η-reduktsioon vastab funktsioonide ekstensionaalsuse omadusele

Uute tüüpide loomine

Uusi tüüpe saab luua kahel viisil:

data e. uue algebralise andmetüübi loomine,

Type e. avaldis, mille tüüp on `Type`

Uute tüüpide loomine

Uusi tüüpe saab luua kahel viisil:

data e. uue algebralise andmetüübi loomine,

Type e. avaldis, mille tüüp on `Type`

1. Saame defineerida tüübisünonüüme:

```
Pikkus : Type
Pikkus = Int
```

2. Tõeväärtuste tüüp `Bool` on defineeritud järgnevalt:

```
data Bool = True | False
```

Uute tüüpide loomine

Uusi tüüpe saab luua kahel viisil:

data e. uue algebralise andmetüübi loomine,

Type e. avaldis, mille tüüp on `Type`

1. Saame defineerida tüübisünonüüme:

```
Pikkus : Type
Pikkus = Int
```

2. Tõeväärtuste tüüp `Bool` on defineeritud järgnevalt:

```
data Bool = True | False
```

Sellest koodireast loeme välja järgnevat:

- defineeritakse uus andmetüüp `Bool`;

Uute tüüpide loomine

Uusi tüüpe saab luua kahel viisil:

data e. uue algebralise andmetüübi loomine,

Type e. avaldis, mille tüüp on `Type`

1. Saame defineerida tüübisünonüüme:

```
Pikkus : Type
Pikkus = Int
```

2. Tõeväärtuste tüüp `Bool` on defineeritud järgnevalt:

```
data Bool = True | False
```

Sellest koodireast loeme välja järgnevat:

- defineeritakse uus andmetüüp `Bool`;
- defineeritakse konstruktor `True : Bool`;

Uute tüüpide loomine

Uusi tüüpe saab luua kahel viisil:

data e. uue algebralise andmetüübi loomine,
Type e. avaldis, mille tüüp on `Type`

1. Saame defineerida tüübisünonüüme:

```
Pikkus : Type
Pikkus = Int
```

2. Tõeväärtuste tüüp `Bool` on defineeritud järgnevalt:

```
data Bool = True | False
```

Sellest koodireast loeme välja järgnevat:

- defineeritakse uus andmetüüp `Bool`;
- defineeritakse konstruktor `True : Bool`;
- defineeritakse konstruktor `False : Bool`.

Siis saab kasutada mustrisobitust:

```
f : Bool → ...
f True  = ...
f False = ...
```

Näide

```
Radius : Type
Radius = Double    -- s.t. Radius asendatakse Double-ga

Width : Type
Width = Double

Height : Type
Height = Double

data Shape = Circle Radius | Rect Width Height

area : Shape → Double
area (Circle r) = pi * r^2
area (Rect w h) = w * h

s1 : Shape
s1 = Circle 1.5

test : Double
test = area s1 + area (Rect 0.75 3.0)
```

Näide

```
data Tree a = Empty | Branch a (Tree a) (Tree a)

flatten : Tree a → List a
flatten Empty            = []
flatten (Branch x t1 t2) = flatten t1 ++ [x] ++ flatten t2

puu1 : Tree Char
puu1 = Branch 'b' (Branch 'a' Empty Empty) (Branch 'c' Empty Empty)

test2 : List Char
test2 = flatten puu1

fold : b → (a → b → b → b) → Tree a → b
fold em br Empty          = em
fold em br (Branch x y z) = br x (fold em br y) (fold em br y)

flatten' :  Tree a → List a
flatten' = fold [] (λx, xs, ys ⇒ xs ++ [x] ++ ys)
```

Näide: Listid

Listid on defineeritud järgmiselt:

```
data List a = Nil | (::) a (List a)
```

Näide: Listid

Listid on defineeritud järgmiselt:

```
data List a = Nil | (::) a (List a)
```

Ehk siis:

- Iga tüübi `a` jaoks eksisteerib list `List a`;
- tühja listi konstruktor `[] : List a`;
- mittetühja listi konstruktor `(::) : a → List a → List a`.

Näide:

```
len : List a → Nat    -- prelüüdis  length
len Nil       = 0
len (_ :: xs) = 1 + len xs
```

Näide: nurjumisvõimalusega funktsioonid

Tüübipere `Maybe` on defineeritud järgnevalt:

```
data Maybe a = Nothing | Just a
```

Näide: nurjumisvõimalusega funktsioonid

Tüübipere `Maybe` on defineeritud järgnevalt:

```
data Maybe a = Nothing | Just a
```

Listist otsimise funktsioon:

```
lookup : Int → List (Int,b) → Maybe b
lookup x []          = Nothing
lookup x ((y,z)::ys) = if x==y then Just z else lookup x ys
```

Näide: nurjumisvõimalusega funktsioonid

Tüübipere `Maybe` on defineeritud järgnevalt:

```
data Maybe a = Nothing | Just a
```

Listist otsimise funktsioon:

```
lookup : Int → List (Int,b) → Maybe b
lookup x []          = Nothing
lookup x ((y,z)::ys) = if x==y then Just z else lookup x ys
```

Näiteks:

- `lookup 4 [(3,'x'),(4,'y')] == Just 'y'`
- `lookup 2 [(3,'x'),(4,'y')] == Nothing`

Näide: nurjumisvõimalusega funktsioonid

Tüübipere `Maybe` on defineeritud järgnevalt:

```
data Maybe a = Nothing | Just a
```

Listist otsimise funktsioon:

```
lookup : Int → List (Int,b) → Maybe b
lookup x []          = Nothing
lookup x ((y,z)::ys) = if x==y then Just z else lookup x ys
```

Näiteks:

- `lookup 4 [(3,'x'),(4,'y')] == Just 'y'`
- `lookup 2 [(3,'x'),(4,'y')] == Nothing`

Tähelepanekud:

- `Maybe a` väärtusi on ühe võrra rohkem kui `a` väärtusi

Näide: Tüüpide summa

Tüübid `Either a b` on defineeritud järgnevalt:

```
data  Either a b = Left a | Right b
```

Näide: Tüüpide summa

Tüübid `Either a b` on defineeritud järgnevalt:

```
data  Either a b = Left a | Right b
```

Kasutatakse näiteks siis kui on vaja edastada veateadet:

```
lookup' : Int → List (Int,b) → Either String b
lookup' x []          = Left "Elementi ei leitud!"
lookup' x ((y,z)::ys) = if x==y then Right z else lookup' x ys
```

Näide: Tüüpide summa

Tüübid `Either a b` on defineeritud järgnevalt:

```
data  Either a b = Left a | Right b
```

Kasutatakse näiteks siis kui on vaja edastada veateadet:

```
lookup' : Int → List (Int,b) → Either String b
lookup' x []          = Left "Elementi ei leitud!"
lookup' x ((y,z)::ys) = if x==y then Right z else lookup' x ys
```

Näiteks:

- `lookup' 4 [(3,'x'),(4,'y')] == Right 'y'`
- `lookup' 2 [(3,'x'),(4,'y')] == Left "Elementi ei leitud!"`

Kirjed

Erisüntaks algebraliste andmetüübide jaoks.

- ```
  record Point where
      constructor MkPoint
      x, y, z : Double

  record Spehere where
      constructor MkSpehere
      center: Point
      radius: Double
  ```

- Saab kasutada konstruktorit

  ```
  pt1 : Point  -- "vana" süntaksiga
  pt1 = MkPoint 10 20 12
  ```

- ...või anda argumendid nimeliselt

  ```
  sp1 : Spehere  -- kirjete süntaks
  sp1 = MkSpehere { radius = 2, center = pt1 }
  ```

- Kirje välju saab projitseerida.

  ```
  Main> sp1.center.x
  10.0
  ```
  ```

Kirjed (jätk)

Erisüntaks algebraliste andmetüübide jaoks.

- Selliseid projektsioone saab ka ise defineerida:

```
(.dist) : Point → Double
(.dist) p = sqrt (p.x^2 + p.y^2)

Main> pt1.dist
22.360679774997898
```

Kirjed (jätk)

Erisüntaks algebraliste andmetüübide jaoks.

- Selliseid projektsioone saab ka ise defineerida:

```
(.dist) : Point → Double
(.dist) p = sqrt (p.x^2 + p.y^2)

Main> pt1.dist
22.360679774997898
```

- Uut kirjet saab luua ka vana kirje põhjal:

```
resize : Spehere → (dr: Double) → Spehere
resize c dr = record { radius = c.radius + dr } c
```

- Väljale saab rakendada teisendusfunktsiooni.

```
resize_alt : Spehere → (dr: Double)  → Spehere
resize_alt c dr = record { radius $= (+ dr) } c
```

- Seda ka läbi mitme taseme:

```
move : (dx: Double)→(dy: Double)→(dz: Double)→Spehere→Spehere
move dx dy dz = record { center.x $= (+ dx),
                         center.y $= (+ dy),
                         center.z $= (+ dz) }
```