Näide

Binaararvud ja liitmine

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`.

Ülesanne

Täida lüngad, et implementeerida and'' funktsiooniga foldr.

```
and' : List Bool → Bool
and' [] = True
and' (x :: xs) = x && and' xs

-- Main> and' [4<5, 5<6, 7≠8]
-- True

and'' : List Bool → Bool
and'' = foldr ?f ?a
```

Kas saab ka foldl-ga?

Ülesanne

Täida lüngad, et implementeerida filter” funktsiooniga foldr .

```
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

-- Main> filter' (<10) [3,7,12,13,4]
-- [3, 7, 4]

filter'' : (a → Bool) → List a → List a
filter'' p = foldl ?f2 ?a2
```

Kas saab ka foldl -ga?

Ülesanne

Kirjuta funktsioon maximum, mis leiab täisarvude listist maksimaalse elemendi.

Uusi tüüpe saab luua kahel viisil:

data e. uue algebralise andmetüübi loomine,

Type e. avaldis, mille tüüp on `Type`

1. Saame defineerida tüübisünonüüme ja tüübifunktsioone:

```
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: Naturaalarvud

Naturaalarvud on (standardteegis) defineeritud järgmiselt:

```
data Nat = Z | S Nat
```

Ehk siis:

- `Nat` on naturaalarvude tüüp;
- `Z : Nat` ehk null on naturaalarv;
- `S : Nat → Nat` ehk kui `n : Nat`, siis ühe võrra suurem arv `S n` on naturaalarv .

Näide:

```
add : Nat → Nat → Nat
add 0 y = y
add (S x) y = S (add x y)
```

Näide: Listid

Listid on (standardteegis) defineeritud järgmiselt:

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

Ehk siis:

- Iga tüübi a jaoks eksisteerib list `List a`;
- tühja listi konstruktor `Nil : 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
```

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

Kirjed

Erisüntaks algebraliste andmetüübide jaoks.

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

  record Sphere where
      constructor MkSphere
      center: Point
      radius: Double
  ```

- Saab kasutada konstruktorit

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

- ...või anda argumendid nimeliselt

  ```
  sp1 : Sphere  -- kirjete süntaks
  sp1 = MkSphere { radius = 2, center = pt1 }
  ```

- Kirje välju saab projitseerida.

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

Ülesanne

Implementeeri div2 funktsiooniga naturaalarvude kahega jagamine.

```
div2 : Nat → Nat
div2 n = ?div2_rhs

-- Main> div2 10
-- 5
-- Main> div2 11
-- 5
-- Main> div2 12
-- 6
```

Ülesanne

Implementeeri sub funktsiooniga naturaalarvude lahutamine.

```
sub : Nat → Nat → Nat
sub x y = ?sub_rhs

-- Main> 3 `sub` 2
-- 1
-- Main> 3 `sub` 5
-- 0
```