

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.

$$f : a \rightarrow (b \rightarrow (c \rightarrow d))$$

$$f \quad x : \quad b \rightarrow (c \rightarrow d)$$

$$(f \quad x) \quad y : \quad c \rightarrow d$$

$$((f \quad x) \quad y) \quad z : \quad d$$

ehk

$$f : a \rightarrow b \rightarrow c \rightarrow d$$

$$f \quad x : \quad b \rightarrow c \rightarrow d$$

$$f \quad x \quad y : \quad c \rightarrow d$$

$$f \quad x \quad y \quad z : \quad d$$

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

```
f xs = sum (takeWhile ( $\neq 0$ ) xs)
```

asemel

```
f = sum  $\circ$  takeWhile ( $\neq 0$ )
```

- Funktsiooni $(a, b) \rightarrow c$ *karritud* kuju on $a \rightarrow b \rightarrow c$.
- Kirjuta funktsioone nii, et neid oleks võimalik osaliselt rakendada!

Kava

- * 1. loeng
 - Admin
 - K: Motivatsioon
 - K: Idris
 - K: paradigmad
- * 2. loeng
 - K: väärtused ja tüübid
 - K: listid
 - V: termid
- * 3. loeng
 - V: vabad muutujad
 - V: alfa-koversioon
 - V: substitutsioon
 - K: lambdad
 - K: kõrgemat-järku fun.
 - K: osaline rakendamine
- * 4. loeng
 - K: uute tüüpide loomine
 - K: kirjed
 - V: beta-reduktsioon
 - V: eeta-reduktsioon
 - V: reduktsiooni järjekorrad
- * 5. loeng
 - K: liidesed
- K: IO
- K: do-süntaks
- V: konstandid
- * 6. loeng
 - V: püsipunktid
 - K: laisk väärtustamine
- * 7. loeng
 - V: Lihtsalt tüübitud λ -arvutus
- * 8. loeng
 - V: Curry-Howard'i vastavus
- * 9. loeng
 - K: Sõltuvad tüübid
- * 10. loeng
 - V: Monaadidest maalähedaselt
- * 11. loeng
 - K: Monad ja Applicative Idrises
- * 12. loeng
 - K: Kvantitatiivne tüübiteooria
- * 13. loeng
 - V: Kombinaatorloogika
 - V: tõeväärtused
 - V: paarid ja ennikud
 - V: naturaalarvud
 - V: listid

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 z)

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

Näide: Naturaalarvud

Naturaalarvud on (standardteegis) defineeritud järgmiselt:

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

Ehk siis:

- `Nat` on naturaalarvude tüüp;
- `z` ehk null on naturaalarv;
- kui `n` on naturaalarv, 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

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)

- 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 = { radius := c.radius + dr } c
```

- Väljale saab rakendada teisendusfunktsiooni.

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

- Seda ka läbi mitme taseme:

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