

Tüübiklassid

Motivatsioon

- Aritmeetilised operaatorid:

$(+) :: Int \rightarrow Int \rightarrow Int$

$(+) :: Float \rightarrow Float \rightarrow Float$

- Tüüp $(+) :: a \rightarrow a \rightarrow a$ on liiga üldine!

- Polümorfne võrdus:

$(==) :: a \rightarrow a \rightarrow Bool$

- Kuidas kontrollida näiteks funktsioonide võrdust?

- Polümorfne väljastus:

$show :: a \rightarrow String$

- Funktsioon *show* on igal tüübil defineeritud erinevalt!

Tüübiklassid

Klasside defineerimine

class [*context* $\Rightarrow$] *classname* *tvar* **where** { *cbody* }

- Ülemääratud operaatoreid defineeritakse *tüübiklassidega*
- Klassidefinitsiooni keha koosneb meetodide tüübisignatuuridest ja *default*-definitioonidest

Klass *Eq* (definitsoon prelüüdist)

class *Eq* *a* **where**

$(==), (/=) :: a \rightarrow a \rightarrow Bool$

$x /= y = not (x == y)$

NB!

Meetodide tüüpides esinevad tüübimuutujad on kitsendatud vastava klassikontekstiga

$(==) :: Eq\ a \Rightarrow a \rightarrow a \rightarrow Bool$

Tüübiklassid

Klassikuuluvuse deklareerimine

instance [*context* $\Rightarrow$] *classname type* **where** {*ibody*}

- Deklareeritav tüüp peab olema kujul *tcon tvar₁ ... tvar_n*, kusjuures kõik tüübimuutujad peavad olema erinevad
- Deklaratsiooni keha koosneb meetodide definitsioonidest
- Default-definitsiooniga meetodi pole vaja (aga võib) uuesti defineerida

Tüübiklassid

Kahendlehtpuude võrdus

```
data IntBtree = Lf Int | Br IntBtree IntBtree
```

```
instance Eq IntBtree where
```

```
    Lf i1      == Lf i2      = i1 == i2
```

```
    Br t11 t12 == Br t21 t22 = t11 == t21 && t12 == t22
```

```
    _          == _          = False
```

Listide võrdus (defineeritud prelüüdis)

```
instance Eq a => Eq [a] where
```

```
    []      == []      = True
```

```
    (x : xs) == (y : ys) = x == y && xs == ys
```

```
    _       == _       = False
```

Tüübiklassid

Põõsaste võrdus

```
data Bush a = One a | Two (Bush a) (Bush a)
              | Many [Bush a]

instance Eq a => Eq (Bush a) where
    One x      == One y      = x == y
    Two x1 x2 == Two y1 y2 = x1 == y1 && x2 == y2
    Many xs   == Many ys   = xs == ys
    _          == _          = False
```

NB!

On kasutatud kolme erinevat võrdust!!!

```
(==) :: a → a → Bool
(==) :: Bush a → Bush a → Bool
(==) :: [Bush a] → [Bush a] → Bool
```

Tüübiklassid

Aksioomid

Võrdusoperaator (`==`) peaks olema refleksiive, sümmeetriline ja transitiivne; so. peaks täitma järgmisi aksioome:

$$x == x \quad = \quad \textit{True}$$

$$x == y \quad \Leftrightarrow \quad y == x$$

$$x == y \ \&\& \ y == z \quad \Rightarrow \quad x == z$$

NB!

Haskelli kompilaator ei kontrolli, kas defineeritav "isend" rahuldab antud aksioome või mitte, ning on puhtalt programmeerija ülesanne garanteerida, et ta seda tõepoolest teeb.

Tüübiklassid

Intuitsioon

- Klassidest võib mõelda, kui predikaatidest üle tüüpide
- Tüübid, mis rahuldavad neid predikaate, omavad “lisa funktsionaalsust”
- Klasside deklaratsioonid defineerivad mis liiki “lisa funktsionaalsusega” on tegemist
- “Isendite” deklaratsioonid defineerivad selle “lisa funktsionaalsuse” konkreetse tüübi jaoks

Tüübiklassid

Klassikuuluvuse tuletamine

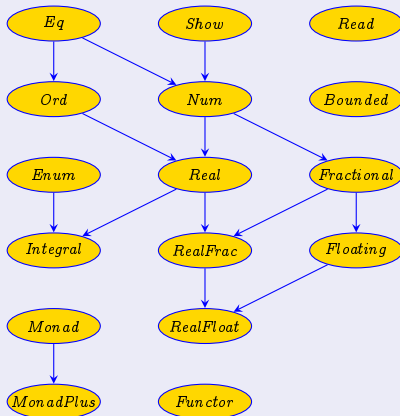
```
data Bool = False | True
  deriving (Eq, Ord, Ix, Enum, Read, Show, Bounded)
data Maybe a = Nothing | Just a
  deriving (Eq, Ord, Read, Show)
data Either a b = Left a | Right b
  deriving (Eq, Ord, Read, Show)
data Ordering = LT | EQ | GT
  deriving (Eq, Ord, Ix, Enum, Read, Show, Bounded)
```

NB!

`deriving` abil saab tuletada ainult eeldefineeritud klasse *Eq*, *Ord*, *Enum*, *Bounded*, *Show* ja *Read*.

Tüübiklassid

Eeldefineeritud klasside hierarhia



Tüübiklassid

Klass *Ord*

class (*Eq a*) $\Rightarrow$ *Ord a* **where**

compare :: *a* $\rightarrow$ *a* $\rightarrow$ *Ordering*

(*<*), (*<=*), (*>=*), (*>*) :: *a* $\rightarrow$ *a* $\rightarrow$ *Bool*

max, *min* :: *a* $\rightarrow$ *a* $\rightarrow$ *a*

compare *x y* | *x* == *y* = *EQ*

| *x* <= *y* = *LT*

| *otherwise* = *GT*

x <= *y* = *compare* *x y* /= *GT*

x < *y* = *compare* *x y* == *LT*

x >= *y* = *compare* *x y* /= *LT*

x > *y* = *compare* *x y* == *GT*

max *x y* = **if** *x* >= *y* **then** *x* **else** *y*

min *x y* = **if** *x* < *y* **then** *x* **else** *y*

Tüübiklassid

Klass *Enum*

class *Enum* *a* **where**

toEnum :: *Int* → *a*

fromEnum :: *a* → *Int*

enumFrom :: *a* → [*a*] -- [*n* ..]

enumFromThen :: *a* → *a* → [*a*] -- [*n*, *n'* ..]

enumFromTo :: *a* → *a* → [*a*] -- [*n* .. *m*]

enumFromThenTo :: *a* → *a* → *a* → [*a*] -- [*n*, *n'* .. *m*]

succ, *pred* :: *Enum* *a* ⇒ *a* → *a*

succ = *toEnum* . (+1) . *fromEnum*

pred = *toEnum* . (subtract 1) . *fromEnum*

Arvudega seotud klassid

Klass *Num*

```
class (Eq a, Show a)  $\Rightarrow$  Num a where  
  (+), (-), (*) :: a  $\rightarrow$  a  $\rightarrow$  a  
  negate      :: a  $\rightarrow$  a  
  abs, signum :: a  $\rightarrow$  a  
  fromInteger :: Integer  $\rightarrow$  a  
   $x - y = x + \text{negate } y$ 
```

Klass *Real*

```
class (Num a, Ord a)  $\Rightarrow$  Real a where  
  toRational :: a  $\rightarrow$  Rational
```

Arvudega seotud klassid

Klass *Integral*

```
class (Real a, Enum a)  $\Rightarrow$  Integral a where  
  quot, rem          :: a  $\rightarrow$  a  $\rightarrow$  a  
  div, mod           :: a  $\rightarrow$  a  $\rightarrow$  a  
  quotRem, divMod    :: a  $\rightarrow$  a  $\rightarrow$  (a, a)  
  toInteger          :: a  $\rightarrow$  Integer
```

Klass *Fractional*

```
class (Num a)  $\Rightarrow$  Fractional a where  
  (/)                :: a  $\rightarrow$  a  $\rightarrow$  a  
  recip              :: a  $\rightarrow$  a  
  fromRational       :: Rational  $\rightarrow$  a
```

NB!

Lisaks on eeldefineeritud *RealFrac*, *Floating*, *RealFloat*

Tüübiklassid

Klass *Show*

```
type ShowS = String → String
class Show a where
  showsPrec :: Int → a → ShowS
  showList  :: [a] → ShowS
  show      :: a → String
  show x    = showsPrec 0 x ""
  showsPrec _ x s = show x ++ s
```

Klass *Read*

```
class Read a where...
read :: (Read a) ⇒ String → a
```

Klass *Show*

Arimteetiliste avaldiste näitamine

```
showExp :: Expr → String
showExp (Num n)    = show n
showExp (Var v)     = v
showExp (e1 :+: e2) = showExp e1 ++ "+" ++
                        showExp e2
showExp (e1 **: e2) = showArg e1 ++ "*" ++
                        showArg e2
showArg (e1 :+: e2) = "(" ++
                        showExp (e1 :+: e2) ++
                        ")"
showArg e            = showExp e
```

NB!

Toodud definitsioon on ruutkeerukusega!

Klass *Show*

Eeldefineeritud abifunktsioon

shows :: *Show a* $\Rightarrow$ *a* $\rightarrow$ *ShowS*

shows = *showsPrec* 0

showChar :: *Char* $\rightarrow$ *ShowS*

showChar = (*:*)

showString :: *String* $\rightarrow$ *ShowS*

showString = (*++*)

showParen :: *Bool* $\rightarrow$ *ShowS* $\rightarrow$ *ShowS*

showParen *b p*

= if *b* then *showChar* '(' . *p* . *showChar* ')' else *p*

Klass *Show*

Arimteetiliste avaldiste näitamine

instance *Show Expr* where

showsPrec _ (*Num* *n*) = *shows* *n*

showsPrec _ (*Var* *v*) = *showString* *v*

showsPrec *d* (*e1* *:+:* *e2*) = *showParen* (*d* > 0)
 \$ *showsPrec* 0 *e1*

 . *showString* "+"

 . *showsPrec* 0 *e2*

showsPrec _ (*e1* *:::* *e2*) = *showsPrec* 1 *e1*
 . *showString* "*"

 . *showsPrec* 1 *e2*

Tüübiklassid

Klass *Functor*

```
class Functor f where
  fmap :: (a -> b) -> (f a -> f b)
```

NB!

Muutuja f tähistab unaarset tüübikonstruktorit (mitte tüüpi)!

Kahendpuude funktooriaalus

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

instance Functor Tree where
    fmap f Empty          = Empty
    fmap f (Branch x t1 t2) = Branch (f x) (fmap f t1)
                                     (fmap f t2)
```

Tüübiklassid

Listide funktoriaalus (defineeritud prelüüdis)

```
instance Functor [] where  
    fmap = map
```

Maybe funktoriaalus (defineeritud prelüüdis)

```
instance Functor Maybe where  
    fmap f Nothing = Nothing  
    fmap f (Just x) = Just (f x)
```

Aksioomid

$$\begin{aligned} \text{fmap } id &= id \\ \text{fmap } f . \text{fmap } g &= \text{fmap } (f . g) \end{aligned}$$

Tüübiklassid

Klass *Monad*

```
class Monad m where
  return :: a → m a
  (≫=)   :: m a → (a → m b) → m b
  (≫)    :: m a → m b → m b
  fail   :: String → m a
  p ≫ q  = p ≻= \_ → q
  fail s = error s
```

Listide monaad (defineeritud prelüüdis)

```
instance Monad [] where
  (x : xs) ≻= f = f x ++ (xs ≻= f)
  []        ≻= f = []
  return x  = [x]
  fail s    = []
```

Tüübiklassid

Maybe monaad (defineeritud prelüüdis)

instance Monad Maybe where

Just x $\gg=$ *k* = *k x*

Nothing $\gg=$ *k* = *Nothing*

return = *Just*

fail s = *Nothing*

Aksioomid

return x $\gg=$ *f* = *f x*

m $\gg=$ *return* = *m*

$(m \gg= \backslash x \rightarrow k) \gg= f$ = $m \gg= \backslash x \rightarrow (k \gg= f)$

Aritmeetilste avaldiste väärtustamine

Aritmeetilste avaldised

```
data Expr = Num Int
          | Expr :+: Expr
          | Expr :-: Expr
          | Expr *: Expr
          | Expr /\: Expr
```

Näited

```
exp1 = Num 1 :+: Num 2
exp2 = Num 2 *: (Num 4 :-: Num 1)
exp3 = Num 4 *: Num 3 /\: Num 2
```

Aritmeetilste avaldiste väärtustamine

Lihtne väärtustaja

```
eval :: Expr → Int  
eval (Num i)      = i  
eval (e1 :+: e2) = (eval e1) + (eval e2)  
eval (e1 :-: e2) = (eval e1) - (eval e2)  
eval (e1 **: e2) = (eval e1) * (eval e2)  
eval (e1 :/: e2) = (eval e1) 'div' (eval e2)
```

Näited

```
Main> eval exp2
```

```
6
```

```
Main> eval (Num 3 :/: Num 0)
```

```
Program error: divide by zero
```

Aritmeetilste avaldiste väärtustamine

Veatöõtlusega väärtustaja

$eval :: Expr \rightarrow Maybe Int$

$eval (Num\ i) = Just\ i$

$eval (e1\ :+:\ e2) = \text{case } eval\ e1 \text{ of}$
 $Nothing \rightarrow Nothing$
 $Just\ v1 \rightarrow \text{case } eval\ e2 \text{ of}$
 $Nothing \rightarrow Nothing$
 $Just\ v2 \rightarrow Just\ (v1 + v2)$

$eval (e1\ :-:\ e2) = \dots$

$eval (e1\ :*\ e2) = \dots$

Aritmeetiliste avaldiste väärtustamine

Veatöötusega väärtustaja (järg)

```
eval (e1 :/: e2) = case eval e1 of
    Nothing → Nothing
    Just v1 → case eval e2 of
        Nothing → Nothing
        Just v2 → if v2 == 0
                    then Nothing
                    else Just (v1 'div' v2)
```

Näited

```
Main> eval exp2
Just 6
Main> eval (Num 3 :/: Num 0)
Nothing
```

Aritmeetilste avaldiste väärtustamine

NB!

- Suur hulk koodi duplitseerimist!
- Mis on ühtne muster mida välja faktoriseerida?
- Paneme tähele, et:
 - toimub alamavaldiste väärtustamiste järjestikustamine;
 - kui üks ebaõnnestub, siis ebaõnnestub ka kogu avaldise väärtustamine;
 - väärtustamise õnnestumisel tehakse resultaat kättesaadavaks järgnevatele väärtustustele;
 - so. täpselt see mida ($\gg$) operaator teeb!!

Aritmeetiliste avaldiste väärtustamine

Veatöötusega väärtustaja (monaadiline)

```
eval :: Expr → Maybe Int
eval (Num i)    = return i
eval (e1 :+: e2) = eval e1 >>= \v1 →
                    eval e2 >>= \v2 →
                    return (v1 + v2)
eval (e1 :-: e2) = ...
eval (e1 :*: e2) = ...
eval (e1 :/: e2) = eval e1 >>= \v1 →
                    eval e2 >>= \v2 →
                    if v2 == 0
                      then fail "Divide by 0"
                      else return (v1 'div' v2)
```