

Reaktiivsed pildid

- Piltide definitsioon:

```
data Picture = Region Color Region
             | Picture 'Over' Picture
             | EmptyPic
             deriving Show
```

- Eesmärk on kirjutada programm, mis hiire vasaku nupu kliki korral toob viidatava regiooni pealmiseks; kui klikkida taustale, siis programm lõpetab töö.

Reaktiivsed pildid

- Piltide teisendamise regioonide listiks:

```
pictToList :: Picture -> [(Color,Region)]  
pictToList EmptyPic      = []  
pictToList (Region c r)   = [(c,r)]  
pictToList (p1 'Over' p2) = pictToList p1 ++ pictToList p2
```

Reaktiivsed pildid

- Positsioonile vastava regiooni leidmine:

```
adjust :: [(Color,Region)] -> Coordinate ->
          (Maybe (Color,Region), [(Color,Region)])
adjust [] p = (Nothing, [])
adjust ((c,r):regs) p
  | r 'containsR' p = (Just (c,r), regs)
  | otherwise      = (hit, (c,r) : rs)
  where (hit, rs) = adjust regs p
```

Reaktiivsed pildid

- Mitterekursiivne definitsioon:

```
adjust regs p
  = case (break (\(_,r) -> r 'containsR' p) regs) of
      (top,hit:rest) -> (Just hit, top++rest)
      (_,[])         -> (Nothing, regs)
```

- Funktsioon break on eeldefineeritud:

```
break :: (a -> Bool) -> [a] -> ([a],[a])
```

```
break (>3) [1..6] ==> ([1,2,3],[4,5,6])
```

Reaktiivsed pildid

- Põhistsükkel:

```
loop :: Window -> [(Color,Region)] -> IO ()
loop w regs =
  do clearWindow w
     sequence_ [ drawRegionInWindow w c r
                 | (c,r) <- reverse regs ]
  (x,y) <- getLBP w
  case (adjust regs (pixelToInch (x - xWin2),
                        pixelToInch (yWin2 - y))) of
    (Nothing, _      ) -> closeWindow w
    (Just hit, newRegs) -> loop w (hit : newRegs)
```

Reaktiivsed pildid

- Pildi joonistamine:

```
draw2 :: String -> Picture -> IO ()  
draw2 s p = runGraphics $  
    do w <- openWindow s (xWin,yWin)  
    loop w (pictToList p)
```

Animatsioonid

- Animatsioon on “liikuv”, ehk ajast sõltuv, graafika
- “Liikumise” illusioon saavutatakse sellega, et teatud ajavahemike järel (optimaalseks peetakse 30 freimi sekundis) joonistatakse erineva pildiga uus freim
- Freimide joonistamiseks kasutatakse puhverdatud graafikat, kus on paralleelselt kasutusel kaks videomälu puhvrit
 - kui ühes puhvris olevat pilti kuvatakse ekraanile, toimub teise puhvrisse uue pildi joonistamine; ning kui on aeg uus freim ekraanile kuvada, siis vahetatakse puhvrite rollid ära

Animatsioonid

- Puhverdatud graafika interfeis Haskellis:

```
openWindowEx :: Title -> Maybe Point -> Maybe Size  
              -> RedrawMode -> Maybe Time  
              -> IO Window
```

```
drawGraphic      :: RedrawMode
```

```
drawBufferedGraphic :: RedrawMode
```

```
getWindowTick    :: Window -> IO ()
```

```
timeGetTime       :: IO Word32
```

```
setGraphic        :: Window -> Graphic -> IO ()
```

Animatsioonid

- Animatsioonid Haskellis:

```
type Animation a = Time -> a
```

```
type Time = Float
```

```
rubberBall :: Animation Shape
```

```
rubberBall t = Ellipse (sin t) (cos t)
```

```
animate :: String -> Animation Graphic -> IO ()
```

```
main1 :: IO ()
```

```
main1 = animate "Animated Shape"
```

```
    (withColor Blue . shapeToGraphic . rubberBall)
```

Animatsioonid

```
animate :: String -> Animation Graphic -> IO ()
animate title anim = runGraphics $
  do w <- openWindowEx title (Just(0,0)) (Just(xWin,yWin))
      drawBufferedGraphic (Just 30)

  t0 <- timeGetTime
  let loop =
    do t <- timeGetTime
      let ft = intToFloat(word32ToInt(t-t0)) / 1000
      setGraphic w (anim ft)
      getWindowTick w
      loop
  loop
```

Animatsioonid

- Keerulisemaid animatsioone:

```
revolvingBall :: Animation Region
revolvingBall t = let ball = Shape (Ellipse 0.2 0.2)
                  in Translate (sin t, cos t) ball
```

```
planets :: Animation Picture
planets t
  = let p1 = Region Red (Shape (rubberBall t))
      p2 = Region Yellow (revolvingBall t)
      in p1 'Over' p2
```

```
tellTime :: Animation String
tellTime t = "The time is: " ++ show t
```

Animatsioonid

- Keerulisemaid animatsioone (järg):

```
main2 :: IO ()
main2 = animate "Animated Text"
          (text (100,200) . tellTime)

regionToGraphic :: Region -> Graphic
regionToGraphic = drawRegion . regionToGRegion

main3 :: IO ()
main3 = animate "Animated Region"
          (withColor Yellow . regionToGraphic
            . revolvingBall)
```

Animatsioonid

- Piltide animeerimine:

```
picToGraphic :: Picture -> Graphic
picToGraphic (Region c r) = withColor c (regionToGraphic r)
picToGraphic (p1 'Over' p2)
    = picToGraphic p1 'overGraphic' picToGraphic p2
picToGraphic EmptyPic      = emptyGraphic

main4 :: IO ()
main4 = animate "Animated Picture"
        (picToGraphic . planets)
```

Animatsioonid

- Animeerivate piltide konstrueerimine:

```
emptyA :: Animation Picture
```

```
emptyA t = EmptyPic
```

```
overA :: Animation Picture -> Animation Picture
```

```
      -> Animation Picture
```

```
overA a1 a2 t = a1 t 'Over' a2 t
```

```
overManyA :: [Animation Picture] -> Animation Picture
```

```
overManyA = foldr overA emptyA
```

- Operatsioonid animatsioonidel on analoogsed baastüübi operatsioonidega

Animatsioonid

- Operatsioonide sarnasuse baastüüpidel ja animatsioonidel saame abstraherida tüübiklasside abil
- Tehnilistel põhjustel ei luba Haskell tüübisünonüümi `Animate` tüübiklassi esindajaks deklareerida; seetõttu defineerime uue tüübi, mida lubab

- “Käitumised”:

```
newtype Behavior a = Beh (Time -> a)
```

```
animateB :: String -> Behavior Picture -> IO ()
```

```
animateB s (Beh pf) = animate s (picToGraphic . pf)
```

Animatsioonid

- Tavaliste funktsioonide “kergitamine” käitumistele:

```
lift0 :: a -> Behavior a
```

```
lift0 x = Beh (\t -> x)
```

```
lift1 :: (a -> b) -> (Behavior a -> Behavior b)
```

```
lift1 f (Beh a) = Beh (\t -> f (a t))
```

```
lift2 :: (a -> b -> c) -> (Behavior a -> Behavior b -> Behavior c)
```

```
lift2 g (Beh a) (Beh b) = Beh (\t -> g (a t) (b t))
```

```
lift3 :: (a -> b -> c -> d)
```

```
    -> (Behavior a -> Behavior b -> Behavior c -> Behavior d)
```

```
lift3 g (Beh a) (Beh b) (Beh c) = Beh (\t -> g (a t) (b t) (c t))
```

Animatsioonid

- Käitumiste isendidefinitsioonid:

```
instance Eq (Behavior a) where
  a1 == a2 = error "Can't compare behaviors."
```

```
instance Show (Behavior a) where
  showsPrec n a1 = error "<< Behavior >>"
```

```
instance Num a => Num (Behavior a) where
  (+) = lift2 (+);    (*) = lift2 (*)
  negate = lift1 negate
  abs     = lift1 abs
  signum  = lift1 signum
  fromInteger = lift0 . fromInteger
```

Animatsioonid

- Käitumiste isendidefinitatsioonid (järg):

```
instance Fractional a => Fractional (Behavior a) where
  (/) = lift2 (/)
  fromRational = lift0 . fromRational
```

```
instance Floating a => Floating (Behavior a) where
  pi      = lift0 pi;      sqrt  = lift1 sqrt
  exp     = lift1 exp;    log    = lift1 log
  sin     = lift1 sin;    cos    = lift1 cos
  tan     = lift1 tan;    asin   = lift1 asin
  acos    = lift1 acos;   atan   = lift1 atan
  sinh    = lift1 sinh;   cosh   = lift1 cosh
  tanh    = lift1 tanh;   asinh  = lift1 asinh
  acosh   = lift1 acosh;  atanh  = lift1 atanh
```

Animatsioonid

- Nüüd saame kasutada “arvulisi” käitumisi!!

- Näide:

```
time :: Behavior Time
```

```
time = Beh (\t -> t)
```

```
time + 5
```

```
==> (lift2 (+)) (Beh (\t -> t)) (Beh (\t -> 5))
```

```
==> (\ (Beh a) (Beh b) -> Beh (\t -> a t + b t))  
      (Beh (\t -> t)) (Beh (\t -> 5))
```

```
==> Beh (\t -> (\t -> t) t + (\t -> 5) t )
```

```
==> Beh (\t -> t + 5)
```

Animatsioonid

- Käitumiste konstrueerimine:

```
class Combine a where  
  empty :: a  
  over  :: a -> a -> a
```

```
instance Combine [a] where  
  empty = []  
  over  = (++)
```

```
newtype Fun a = Fun (a->a)  
instance Combine (Fun a) where  
  empty = Fun id  
  Fun a 'over' Fun b = Fun (a . b)
```

Animatsioonid

- Käitumiste konstrueerimine — näide:

```
m :: Behavior Picture
m = let a = lift0 (empty 'over' p)
      in a 'over' empty
```

```
p :: Picture
p = empty
```

- **NB!** Siin on tüübideklaratsioonid kohustuslikud, kuna vastasel korral ei ole võimalik ülelaadimist lahendada

Animatsioonid

- Käitumiste konstrueerimine:

```
reg    = lift2 Region
shape  = lift1 Shape
ell    = lift2 Ellipse
red    = lift0 Red
yellow = lift0 Yellow
translate (Beh a1, Beh a2) (Beh r)
        = Beh (\t -> Translate (a1 t, a2 t) (r t))
```

- Siin pole vaja tüüpe deklareerida, kuna tüübid on üheselt määratud

Animatsioonid

- Käitumiste konstrueerimine:

```
revolvingBallB :: Behavior Picture
```

```
revolvingBallB
```

```
  = let ball = shape (ell 0.2 0.2)
```

```
    in reg red (translate (sin time, cos time) ball)
```

```
main5 :: IO ()
```

```
main5 = animateB "Revolving Ball Behavior"
```

```
          revolvingBallB
```

Animatsioonid

- Käitumiste konstrueerimine:

```
(>*) :: Ord a => Behavior a -> Behavior a -> Behavior Bool
```

```
(>*) = lift2 (>)
```

```
ifFun :: Bool -> a -> a -> a
```

```
ifFun p c a = if p then c else a
```

```
cond :: Behavior Bool -> Behavior a -> Behavior a -> Behavior a
```

```
cond = lift3 ifFun
```

```
flash :: Behavior Color
```

```
flash = cond (sin time >* 0) red yellow
```

Animatsioonid

- Aja teisendused:

`timeTrans :: Behavior Time -> Behavior a -> Behavior a`

`timeTrans (Beh f) (Beh a) = Beh (a . f)`

`timeTrans (2*time) anim`

`timeTrans (5+time) anim 'over' anim`

Animatsioonid

- Aja teisendused:

```
flashingBall :: Behavior Picture
```

```
flashingBall
```

```
  = let ball = shape (ell 0.2 0.2)
```

```
    in reg (timeTrans (8*time) flash)
```

```
        (translate (sin time, cos time) ball)
```

```
main6 :: IO ()
```

```
main6 = animateB "Flashing Ball" flashingBall
```

Animatsioonid

- Aja teisendused:

```
revolvingBalls :: Behavior Picture
```

```
revolvingBalls
```

```
  = overMany [ timeTrans (lift0 (t*pi/4) + time) flashingBall  
              | t <- [0..7]]
```

```
main7 :: IO ()
```

```
main7 = animateB "Lots of Flashing Balls" revolvingBalls
```

Animatsioonid

- Objektide pööramine:

```
class Turnable a where  
  turn :: Float -> a -> a
```

```
instance Turnable Picture where  
  turn theta (Region c r)    = Region c (turn theta r)  
  turn theta (p1 'Over' p2) =      turn theta p1  
                                'Over' turn theta p2  
  turn theta EmptyPic       = EmptyPic
```

```
instance Turnable a => Turnable (Behavior a) where  
  turn theta (Beh b) = Beh (turn theta . b)
```

Animatsioonid

- Objektide pööramine:

```
rotate :: Float -> Coordinate -> Coordinate
```

```
rotate theta (x,y) = (x*c+y*s, y*c-x*s)
```

```
  where (s,c) = (sin theta, cos theta)
```

```
instance Turnable Shape where
```

```
  turn theta (Polygon ps) = Polygon (map (rotate theta) ps)
```

```
  -- ülejäänud variandid puudu
```

```
instance Turnable Region where
```

```
  turn theta (Shape sh) = Shape (turn theta sh)
```

```
  -- ülejäänud variandid puudu
```