“Sündmused” ja “käitumised”

- Sündmused on ümbritsevas keskonnas reaalajas “juhtuvad” tegevused
- Eesmärk on kombineerida käitumised sündmustega mingil ratsionaalsel viisil
- `Graphics` teek defineerib sündmuste primitiivtüübi:

```
data G.Event
  = Key       { char :: Char, isDown :: Bool }
  | Button    { pt :: Point, isLeft, isDown :: Bool }
  | MouseMove { pt :: Point }
  | Resize
  | Closed
deriving Show
```


“Sündmused” ja “käitumised”

- Lihtsate animatsioonide korral on käitumine ajast sõltuv väärtus
- Reaktiivsete animatsioonide korral sõltub käitumine lisaks ajale ka sündmustest
- Reaktiivne käitumine (1. katse)

```
type UserAction = G.Event

newtype Behavior1 a
  = Behavior1 ([(UserAction,Time)] -> Time -> a)
```

- NB! Sündmused on ajatembeldatud

“Sündmused” ja “käitumised”

- Toodud esitus on ebaefektiivne, kuna igal ajahetkel tuleb sündmuste jada uuesti algusest peale töödelda (samuti on probleemne mäluhalduse seisukohast)
- Analoogia: olgu antud kaks listi `xs` ja `ys` ning soovime kontrollida, kas listi `ys` kõik elemendid kuuluvad listi `xs`

```
inList :: [Int] -> Int -> Bool

result1 :: [Bool]
result1 = map (inList xs) ys
```

- Antud lahenduses läbitakse (halvimal juhul) listi `xs` nii mitu korda, kui palju on elemente listis `ys`

“Sündmused” ja “käitumised”

- Kui listid `xs` ja `ys` on monotoonselt järjestatud, siis on võimalik hakkama saada listi `xs` ühekordse läbivaatusega

```
result2 :: [Bool]
result2 = manyInList xs ys

manyInList :: [Int] -> [Int] -> [Bool]
manyInList (x:xs) (y:ys)
    | x < y     =          manyInList xs (y:ys)
    | otherwise = (x==y) : manyInList (x:xs) ys
manyInList _  _ = []
```

“Sündmused” ja “käitumised”

- Reaktiivne käitumine (2. katse) — kasutame eeltoodud analoogiat:

```
newtype Behavior2 a
  = Behavior2 ([(UserAction,Time)] -> [Time] -> [a])
```

- Reaktiivne käitumine (3. katse) — “võtame kaadreid” ainult sündmuste toimumiste hetkedel:

```
newtype Behavior3 a
  = Behavior3 ([UserAction] -> [Time] -> [a])
```

- Reaktiivne käitumine (4. katse) — lisame juurde (tagasi) “kaadrite võtmise” siis kui sündmusi ei toimu:

```
newtype Behavior4 a
  = Behavior4 ([Maybe UserAction] -> [Time] -> [a])
```


“Sündmused” ja “käitumised”

- Reaktiivne käitumine — lõppversioon:

```
newtype Behavior a
  = Behavior (([Maybe UserAction],[Time]) -> [a])
```

- Sündmuste esitamine:

```
newtype Event a
  = Event (([Maybe UserAction],[Time]) -> [Maybe a])
```

- Tüübid `Event a` ja `Behavior (Maybe a)` on *isomorfsed*!
- Intuitiivselt on sündmus käitumine, mis antud ajahetkel kas toimus või mitte

FAL'i realisatsioon

- Käitumiste baaskombinaatorid:

```
time :: Behavior Time
time = Behavior (\(_,ts) -> ts)

constB :: a -> Behavior a
constB x = Behavior (\_ -> repeat x)

($*) :: Behavior (a->b) -> Behavior a -> Behavior b
Behavior ff $* Behavior fb
  = Behavior (\uts -> zipWith ($) (ff uts) (fb uts))
```


FAL'i realisatsioon

- Väärtuste "kergitamine" käitumisteks:

```
lift0 :: a -> Behavior a
lift0 = constB

lift1 :: (a -> b) -> (Behavior a -> Behavior b)
lift1 f b1 = lift0 f $* b1

lift2 :: (a -> b -> c) ->
         (Behavior a -> Behavior b -> Behavior c)
lift2 f b1 b2 = lift1 f b1 $* b2

lift3 f b1 b2 b3 = lift2 f b1 b2 $* b3
```

FAL’i realisatsioon

- “Kergitatud” käitumisi:

```
pairB :: Behavior a -> Behavior b -> Behavior (a,b)
pairB = lift2 (,)

fstB :: Behavior (a,b) -> Behavior a
fstB  = lift1 fst
sndB :: Behavior (a,b) -> Behavior b
sndB  = lift1 snd
```

- Lisaks on “kergitatud” värvid, kujundid, regioonid, numbriklassid, jne

FAL’i realisatsioon

- Sündmuste ja käitumiste interaktsioon:

```
untilB :: Behavior a -> Event (Behavior a) -> Behavior a
Behavior fb `untilB` Event fe = memoB $
  Behavior (\uts@(us,ts) -> loop us ts (fe uts) (fb uts))
   where loop (_:us) (_:ts) ~(e:es) (b:bs) =
           b : case e of
                 Nothing             -> loop us ts es bs
                 Just (Behavior fb') -> fb' (us,ts)

memoB :: Behavior a -> Behavior a
memoB (Behavior fb) = Behavior (memo1 fb)
```

FAL’i realisatsioon

- Sündmuste ja käitumiste interaktsioon:

```
switch :: Behavior a -> Event (Behavior a) -> Behavior a

Behavior fb `switch` Event fe = memoB $
  Behavior (\uts@(us,ts) -> loop us ts (fe uts) (fb uts))
   where loop (_:us) (_:ts) ~(e:es) (b:bs) =
           b : case e of
                 Nothing  -> loop us ts es bs
                 Just (Behavior fb')
                          -> loop us ts es (fb' (us,ts))
```


FAL’i realisatsioon

- Sündmused — hiire “klõpsud”

```
lbp :: Event ()
lbp = Event (\(uas,_) -> map getlbp uas)
    where getlbp (Just (Button _ True True)) = Just ()
          getlbp _                           = Nothing
```

- Sündmused — klahvi vajutus

```
key :: Event Char
key = Event (\(uas,_) -> map getkey uas)
    where getkey (Just (Key ch True)) = Just ch
          getkey _                    = Nothing
```

FAL'i realisatsioon

- Sündmused — hiire liigutamine

```
mm :: Event Coordinate
mm = Event (\(uas,_) -> map getmm uas)
   where getmm (Just (MouseMove pt)) = Just (gPtToPt pt)
         getmm _                     = Nothing

mouse :: (Behavior Float, Behavior Float)
mouse = (fstB m, sndB m)
      where m = (0,0) `step` mm
```


FAL'i realisatsioon

- Sündmuste teisendamine:

```
(=>>) :: Event a -> (a->b) -> Event b
Event fe =>> f
  = Event (\uts -> map aux (fe uts))
    where aux (Just a) = Just (f a)
          aux Nothing  = Nothing

(->>) :: Event a -> b -> Event b
e ->> v = e =>> \_ -> v
```

FAL'i realisatsioon

- Sündmuste valik:

```
(.|.) :: Event a -> Event a -> Event a
Event fe1 .|. Event fe2
  = Event (\uts -> zipWith aux (fe1 uts) (fe2 uts))
      where aux Nothing  Nothing  = Nothing
            aux (Just x) _        = Just x
            aux _        (Just y) = Just y
```


FAL'i realisatsioon

- Sündmuste teisendamine:

```
withElem  :: Event a -> [b] -> Event (a,b)
withElem (Event fe) bs
  = Event (\uts -> loop (fe uts) bs)
    where loop (Just a  : evs) (b:bs)
                         = Just (a,b) : loop evs bs
          loop (Nothing : evs)  bs
                         = Nothing    : loop evs bs

withElem_ :: Event a -> [b] -> Event b
withElem_ e bs = e `withElem` bs =>> snd
```

FAL'i realisatsioon

- Virtuaalsed sündmused:

```
while, when :: Behavior Bool -> Event ()
while (Behavior fb) = Event (\uts -> map aux (fb uts))
    where aux True  = Just ()
          aux False = Nothing

when = unique . while

unique :: Eq a => Event a -> Event a
unique (Event fe) = Event (\uts -> aux (fe uts))
     where aux xs = zipWith remdup (Nothing:xs) xs
           remdup x y | x==y      = Nothing
                      | otherwise = y
```


FAL'i realisatsioon

- Sündmuste ja käitumiste interaktsioon:

```
snapshot :: Event a -> Behavior b -> Event (a,b)
Event fe `snapshot` Behavior fb
  = Event (\uts -> zipWith aux (fe uts) (fb uts))
      where aux (Just x) y = Just (x,y)
            aux Nothing  _ = Nothing

snapshot_ :: Event a -> Behavior b -> Event b
snapshot_ e b = e `snapshot` b =>> snd
```

FAL'i realisatsioon

- Sündmuste ja käitumiste interaktsioon:

```
step :: a -> Event a -> Behavior a
a `step` e = constB a `switch` e =>> constB

stepAccum :: a -> Event (a->a) -> Behavior a
a `stepAccum` e = b
   where b = a `step` (e `snapshot` b =>> uncurry ($))
```


FAL'i realisatsioon

- Arvkäitumiste integreerimine:

$$\int_{t=0}^{T} f(t)dt = \lim_{\Delta t \to 0} \sum_{n=0}^{T/\Delta t} f(\Delta t \cdot n) \cdot \Delta t$$

```
integral :: Behavior Float -> Behavior Float
integral (Behavior fb)
  = Behavior (\uts@(us,t:ts)
                   -> 0 : loop t 0 ts (fb uts))
      where loop t0 acc (t1:ts) (a:as)
                 = let acc' = acc + (t1-t0)*a
                   in acc' : loop t1 acc' ts as
```

Näide — “paddleball”

```
paddleball vel = walls `over` paddle `over` pball vel

walls = upper `over` left `over` right
  where blueRec coord sizeX sizeY
              = paint blue (translate coord)
                                  (rec sizeX sizeY)
        upper = blueRec ( 0,1.7) 4.4  0.05
        left  = blueRec (-2.2,0) 0.05 3.4
        right = blueRec ( 2.2,0) 0.05 3.4

paddle = paint red (translate (fst mouse, -1.7)
                                  (rec 0.5 0.05))
```


Näide — “paddleball”

```
pball vel =
 let xvel    = vel `stepAccum` xbounce ->> negate
     xpos    = integral xvel
     xbounce = when (xpos >*  2 ||* xpos <* -2)
     yvel    = vel `stepAccum` ybounce ->> negate
     ypos    = integral yvel
     ybounce = when (ypos >* 1.5
             ||* ypos      `between` (-2.0,-1.5) &&*
                 fst mouse `between` (xpos-0.25,xpos+0.25))
 in paint yellow (translate (xpos, ypos) (ell 0.2 0.2))

x `between` (a,b) = x >* a &&* x <* b
```