Uute tüüpide defineerimine

- Tüübisünonüümid:

$$\texttt{type}\ tcon\ tv_1 \ldots tv_n \quad = \quad texpr \qquad\qquad n{\geq}0$$

- Algebralised andmetüübid:

$$\begin{array}{rcl} \texttt{data}\ tcon\ tv_1 \ldots tv_n & = & dcon_1\ texpr_{11} \ldots texpr_{1m_1} \\ & \mid & \ldots \qquad\qquad n,m_1,\ldots,m_n{\geq}0 \\ & \mid & dcon_n\ texpr_{n1} \ldots texpr_{nm_n} \end{array}$$

- "Uustüübid":

$$\texttt{newtype}\ tcon\ tv_1 \ldots tv_n \quad = \quad dcon\ texpr \qquad n{\geq}0$$

Tüübisünonüümid

- Annab tüübiavaldisele nime, mis on esialgese tüübiavaldisega täiesti ekvivalentne
- Tüüp võib olla polümorfne, kuid mitte rekursiivne ja tüübikonstruktorit ei saa osaliselt rakendada
- Näited:

```
type String  = [Char]    -- eeldefineeritud
type AssocList a b = [(a,b)]
type Predicate a   = a -> Bool
```

Algebralised andmetüübid

- Loenditüübid

```
data Day = Mon | Tue | Wed | Thu
               | Fri | Sat | Sun
         deriving Show
```

- Nimed `Mon` – `Sun` on konstruktor-konstandid (kuna neil ei ole argumente) ja on tüübi `Day` ainsad elemendid.

- Näide:

```
valday :: Int -> Day
valday n = days !! (n-1)
   where days = [Mon,Tue,Wed,Thu,Fri,Sat,Sun]
```


Algebralised andmetüübid

- Funtsioonid defineeritakse näidiste sobitamise abil

```
workday :: Day -> Bool
workday Mon  = True
workday Tue  = True
workday Wed  = True
workday Thu  = True
workday Fri  = True
workday Sat  = False
workday Sun  = False
```

Algebralised andmetüübid

- Variant-kirjed

```
data Tagger = Tagn Integer | Tagb Bool
```

- Konstruktorid `Tagn` ja `Tagb` on funktsioonid

```
Tagn :: Integer -> Tagger
Tagb :: Bool    -> Tagger
```

- Erinevalt (tavalistest) funktsioonidest
  - on konstruktor-aplikatsioonid (näit. `Tagn 7`) kanoonilisel kujul (so. neid ei saa edasi lihtsustada);
  - saab konstruktoreid kasutada näidiste sobitamisel

Algebralised andmetüübid

- Näiteid:

```
number  (Tagn i) = i
boolean (Tagb b) = b

isNum (Tagn _) = True
isNum (Tagb _) = False
isBool x       = not (isNum x)

Hugs> :t number
number :: Tagger -> Integer
Hugs> number (Tagn 3)
3
```

Algebralised andmetüübid

- Polümorfsed andmetüübid

```
data Maybe a = Just a | Nothing
               deriving Show
```

- Konstruktorid `Just` ja `Nothing` on polümorfset tüüpi

```
Just    :: a -> Maybe a
Nothing :: Maybe a
```

- Polümorfsete konstantide "näitamine" nõuab tüübi ilmutatud deklareerimist (analoogselt tühja listiga)!

Ebaõnnestumise modelleerimine

- Otsida tabelist võtmele vastav väärtus

```
type Table = [(String,Int)]

lookup :: String -> Table -> Int
lookup key ((k,v):xs) | key == k  = v
                      | otherwise = lookup key xs
```

- Kui võtmele vastavat elementi pole, siis on viga!

```
lookup :: String -> Table -> Maybe Int
lookup key []                     = Nothing
lookup key ((k,v):xs) | key == k  = Just v
                      | otherwise = lookup key xs
```

Algebralised andmetüübid

- Rekursiivsed andemtüübid

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

- Näide — puu "lamendamine"

```
flatten :: Tree a -> [a]
flatten Empty = []
flatten (Branch x t1 t2)
              = flatten t1 ++ [x] ++ flatten t2
```

- Konstruktori argumendid võivad olla "märgendatud"

```
data Tree a = Empty
     | Branch {val :: a, left, right :: Tree a}
```


"Uustüübid"

- Ühe unaarse konstruktoriga tüüp

```
newtype Table a b = mkTable [(a,b)]
```

- Erinevalt tüübisünonüümist võib olla rekursiivne ja saab osaliselt rakendada
- Erinevalt ühe unaarse konstruktoriga andmetüübist on konstruktor "virtuaalne"
- Konstruktori argument võib olla "märgendatud"

Geomeetrilised kujundid

- Kujundite esitamine

```
data Shape = Rectangle Side Side
           | Ellipse Radius Radius
           | RtTriangle Side Side
           | Polygon [Vertex]
     deriving Show

type Radius = Float
type Side   = Float
type Vertex  = (Float,Float)
```


Geomeetrilised kujundid

- Uute kujundite defineerimine:

```
square s = Rectangle s s
circle r = Ellipse r r
```

- Kujundite pindala:

```
area :: Shape -> Float
area (Rectangle  s1 s2) = s1*s2
area (RtTriangle s1 s2) = s1*s2/2
area (Ellipse r1 r2)    = pi*r1*r2
```

Geomeetrilised kujundid

- Kolmnurga pindala:

```
triArea :: Vertex -> Vertex -> Vertex -> Float
triArea v1 v2 v3 = let a = distBetween v1 v2
                       b = distBetween v2 v3
                       c = distBetween v3 v1
                       s = 0.5*(a+b+c)
                   in sqrt (s*(s-a)*(s-b)*(s-c))
```

- Lõigu pikkus

```
distBetween :: Vertex -> Vertex -> Float
distBetween (x1,y1) (x2,y2)
           = sqrt ((x1-x2)^2 + (y1-y2)^2)
```


Geomeetrilised kujundid

- Polügoni pindala:

```
area (Polygon (v1:vs))  = polyArea vs
   where polyArea :: [Vertex] -> Float
         polyArea (v2:v3:vs')
                        = triArea v1 v2 v3
                        + polyArea (v3:vs')
         polyArea _     = 0
```