Regioonid

- Andmetüüp `Shape` võimaldab esitada lihtsaid geomeetrilisi kujundeid, mida me saame ka lihtsasti graafiliselt joonistada
- Kuidas . . .
  - konstrueerida lihtsamatest kujunditest keerulisemaid?
  - lahendada "probleemi", et osad kujundid (ellips, ristkülik ja täisnurkne kolmnurk) on tasandi koordinaatteljestiku suhtes tsentreeritud?

Regioonid

- Regiooni definitsioon:

```
data Region = Shape Shape
            | Translate Vector Region
            | Scale     Vector Region
            | Complement Region
            | Region `Union` Region
            | Region `Intersect` Region
            | Empty
     deriving Show

type Vector = (Float,Float)
```

Regioonide “tähendus”

- Regioon on tasandil asetsevate punktide hulk
- Tihti (peaaegu alati?) lõpmatu
- Regioonidele formaalse “tähenduse” andmiseks vaja efektiivset moodust lõpmatute hulkade esitamiseks
- Kasutame *karakteristlikke funktsioone*

  ```
  type Set a = a -> Bool
  ```

- Näide:

  ```
  evenInts :: Set Int
  evenInts x = (x `mod` 2) == 0
  ```


Operatsioone hulkadel

- Ülesanne — kuidas leida:
  - kahe hulga ühendit?

    ```
    union :: Set a -> Set a -> Set a
    x `union` y =
    ```

  - kahe hulga ühisosa?

    ```
    intersect :: Set a -> Set a -> Set a
    x `intersect` y =
    ```

  - vastandhulka?

    ```
    complement :: Set a -> Set a
    complement x =
    ```

Regioonide karakteristlikud funktsioonid

- Etteantud koordinaadi kuulumine regiooni:

```
type Coordinate = (Float,Float)
containsR :: Region -> Coordinate -> Bool
```

- Paneme tähele, et `containsR r` on karakteristlik funktsioon; ehk

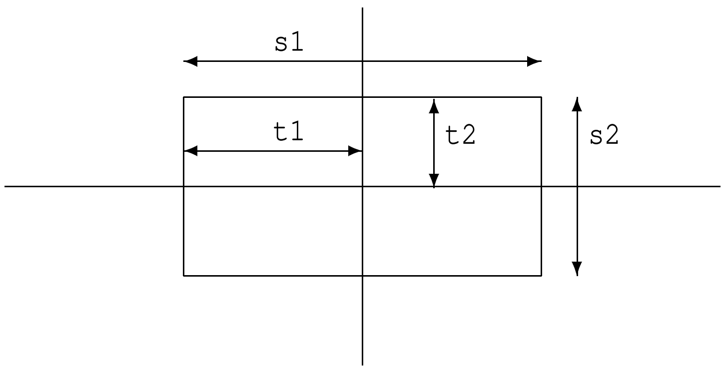
```
containsR :: Region -> Set Coordinate
```

- Defineerime rekursiooniga üle `Region` andmetüübi.
- Rekursiooni baasjuhuks on kujundite andmetüüp `Shape`, mille jaoks defineerime eraldi funktsiooni:

```
containsS :: Region -> Coordinate -> Bool
```

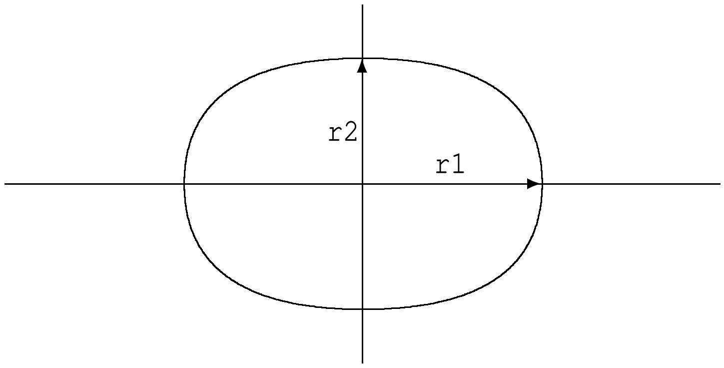

Ristkülik

```
(Rectangle s1 s2) `containsS` (x,y)
   = let t1 = s1/2
         t2 = s2/2
     in -t1<=x && x<=t1 && -t2<=y && y<=t2
```



Ellips

```
(Ellipse r1 r2) `containsS` (x,y)
                 = (x/r1)^2 + (y/r2)^2 <= 1
```



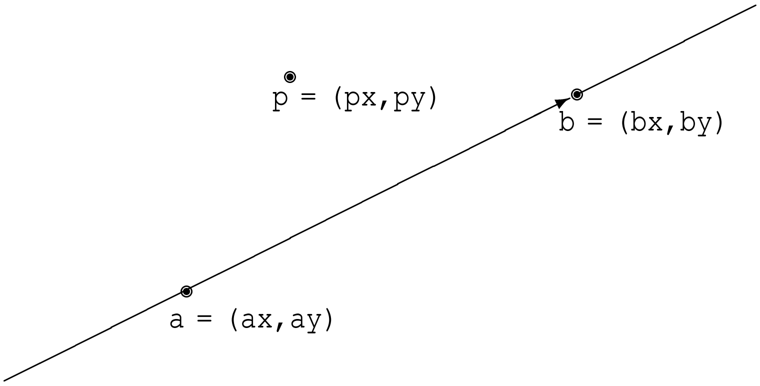
Polügon

Punktist p asub (kumera) polügoni sees, kui ta asub “vasakul pool” igast vastupäeva orienteeritud polügoni küljest

```
(Polygon pts) `containsS` p
   = let shiftpts = tail pts ++ [head pts]
         leftOfList = map isLeftOfp
                          (zip pts shiftpts)
         isLeftOfp p' = isLeftOf p p'
     in and leftOfList
```

Joone “vasak pool”

Punktist `a` punkti `b` mineva vektori poolt tekitatud joonest vasakul asumine



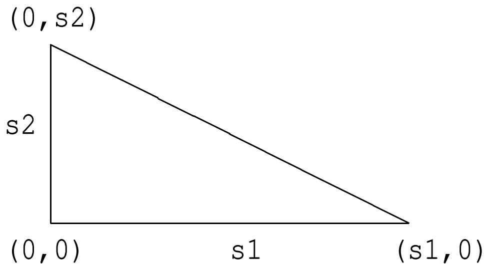
Joone “vasak pool”

```
type Ray = (Coordinate, Coordinate)

isLeftOf :: Coordinate -> Ray -> Bool
(px,py) `isLeftOf` ((ax,ay),(bx,by))
       = let (s,t) = (px-ax, py-ay)
             (u,v) = (px-bx, py-by)
         in  s*v >= t*u
```

Täisnurkne kolmnurk

```
(RtTriangle s1 s2) `containsS` p
   = (Polygon [(0,0),(s1,0),(0,s2)]) `containsS` p
```



Koordinaadi kuulumine regiooni

```
(Shape s) `containsR` p = s `containsS` p
(Translate (u,v) r) `containsR` (x,y)
   = let p = (x-u,y-v) in r `containsR` p
(Scale (u,v) r) `containsR` (x,y)
   = let p = (x/u,y/v) in r `containsR` p
(Complement r) `containsR` p
   = not (r `containsR` p)
(r1 `Union` r2)     `containsR` p
   = r1 `containsR` p || r2 `containsR` p
(r1 `Intersect` r2) `containsR` p
   = r1 `containsR` p && r2 `containsR` p
Empty `containsR` p     = False
```

Regioonide algebra

- Iga `r1`, `r2` ja `r3` korral

  ```
  (r1 `Union` (r2 `Union` r3)) `containsR` p
  ```

  siis ja ainult siis kui

  ```
  ((r1 `Union` r2) `Union` r3) `containsR` p
  ```

- ..., ehk `Union` on *assotsiatiivne*:

  ```
  r1 `Union` (r2 `Union` r3)
               === (r1 `Union` r2) `Union` r3
  ```

- Tõestus on lihtne arvutus ...

Assotsiatiivsuse tõestus

```
(r1 `Union` (r2 `Union` r3)) `containsR` p
==> (r1 `containsR` p) ||
        ((r2 `Union` r3) `containsR` p)
==> (r1 `containsR` p) ||
        ((r2 `containsR` p) || (r3 `containsR` p))
==> ((r1 `containsR` p) || (r2 `containsR` p)) ||
         (r3 `containsR` p)
==> ((r1 `Union` r2) `containsR` p) ||
         (r3 `containsR` p)
==> ((r1 `Union` r2) `Union` r3) `containsR` p
```

NB! Tõestus sõltub `(||)` assotsiatiivsusest (mille saab ka tõstada analoogselt)

Teisi omadusi

- `Union` ja `Intersect` on *assotsiatiivsed*
- `Union` ja `Intersect` on *kommutatiivsed*
- `Union` ja `Intersect` on *distributiivsed*
- Iga `r` korral

```
r `Union` Empty === r
r `Intersect` Complement Empty === r
r `Union` Complement r === Complement Empty
r `Intersect` Complement r === r
```