Naturaalarvud

- Church'i numbrid

$$\begin{array}{lcl} \underline{n} & \equiv & \lambda f\ x.\ f^n\ x \\ \text{succ} & \equiv & \lambda n.\ \lambda f\ x.\ n\ f\ (f\ x) \\ \text{iszero} & \equiv & \lambda n.\ n\ (\lambda x.\ \text{false})\ \text{true} \\ \text{add} & \equiv & \lambda m\ n.\ \lambda f\ x.\ m\ f\ (n\ f\ x) \end{array}$$

- Näide

$$\begin{array}{lcl} \text{add}\ \underline{2}\ \underline{1} & \equiv & (\lambda m\ n.\ \lambda f\ x.\ m\ f\ (n\ f\ x))\ \underline{2}\ \underline{1} \\ & \twoheadrightarrow & \lambda f\ x.\ \underline{2}\ f\ (\underline{1}\ f\ x) \\ & \twoheadrightarrow & \lambda f\ x.\ f\ (f\ (\underline{1}\ f\ x)) \\ & \twoheadrightarrow & \lambda f\ x.\ f\ (f\ (f\ x)) \\ & \equiv & \underline{3} \end{array}$$

- Korrutamine ja astendamine

$$\begin{array}{lcl} \text{mul} & \equiv & \lambda m\ n.\ \lambda f\ x.\ m\ (n\ f)\ x \\ \text{exp} & \equiv & \lambda m\ n.\ \lambda f\ x.\ n\ m\ f\ x \end{array}$$

Laiskus

Meil on kaks standardset reduktsioonijärjekorda:

- normaaljärjekord ja
- aplikatiivjärjekord.

Mis on eelised ja puudused? Kumba eelistada?

Normaaljärjekord on parem!

- Normaaljärjekord leiab alati normaalkuju, kui see eksisteerib! Normaalkuju ei ole näiteks: $(\lambda\, x.\, x\ x)\ (\lambda\, x.\, x\ x)$
- Ei väärtusta argumente mida ei kasutata. Näiteks $(\lambda\, x.\, 0)$ (fact 100)
- Funktsiooni argumendiks võib anda avaldise, millel pole normaalkuju.
  - Praktikas näiteks lõpmatu list. Näiteks Haskellis:

```
sieve (p:xs) = filter (λx → x `mod` p ≠ 0) xs
primes = map head (iterate sieve [2..])

Main> take 3 primes
[2,3,5]
```

- Kui argumenti kasutatakse mitu korda, väärtustatakse seda mitu korda :(

Laisk väärtustamine on veel parem!

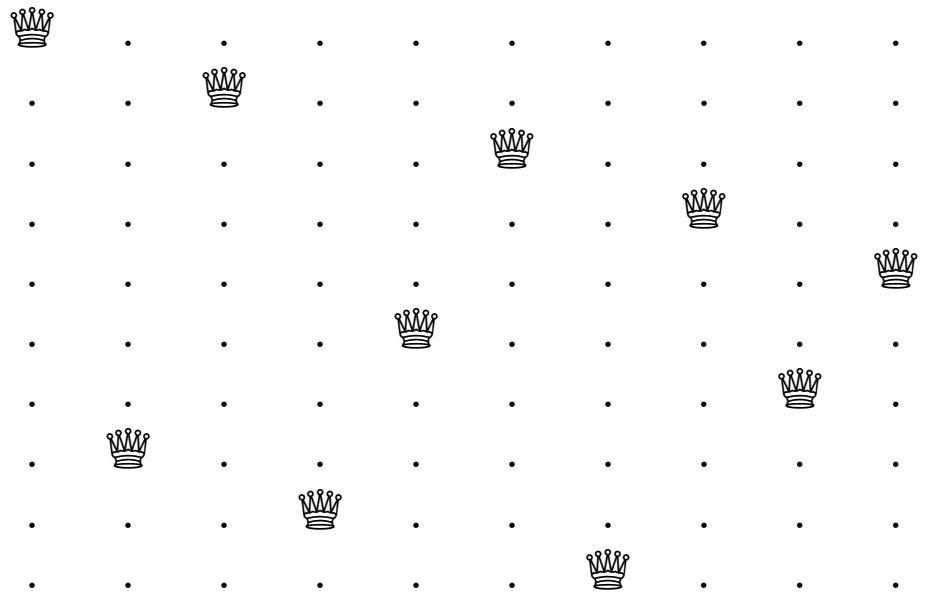
- Nagu normaaljärjekord aga argumenti väärtustatakse maksimaalselt üks kord.
- Graafireduktsioon, mitte puu-reduktsioon.
- (Laisal) väärtustamisel on ka kitsam tähendus, mis ei kasuta reduktsioonisammu vaid on lähedasem kompileerimisele. Sellest räägime mõnes teises loengus/kursusel.

Näide (Haskell)

```
type Queen = (Int, Int)

attacking :: Queen → Queen → Bool
attacking (x1, y1) (x2, y2) =
    (x1 == x2) || (y1 == y2) || abs (x1 - x2) == abs (y1 - y2)

placeable :: Queen → [Queen] → Bool
placeable p qs = and [ (not (attacking p q)) | q ← qs ]

nQueens :: Int → [[Queen]]
nQueens n = nQueens' n where
    nQueens' 0 = [[]]
    nQueens' k =
        [ q:qs  | qs ← nQueens' (k - 1)
                , q  ← [(k, y) | y ← [1..n]]
                , placeable q qs]
```

Näide (Haskell)

```
*Main> time $ queens 10
```



```
0.007199s
```