Sisend-väljund Idrises

- Idrise puhtad funktsioonid ei võimalda teha mittepuhtaid arvutusi.
  - Puhas — funktsiooni tulemus sõltub ainult argumentide väärtusest.
  - Ei saa teha näiteks juhuarvude funktsiooni `random : () → Int`
- Lahendus: `IO` liides
  - '`IO a`' tüüpi väärtus — „masin mis arvutab `a` tüüpi väärtuse“
  - `pure : a → IO a` — masin tagastab esimese argumendi väärtuse
  - `(>>=) : IO a → (a → IO b) → IO b` — masin käivitab esimese argumendi ja rakendab tulemuse teisele
  - ... lisaks baasfunktsioonid nagu `putStrLn : String → IO ()` ja `getLine : IO String`.
- Nii saab kombineerida olemasolevaid `IO` „masinaid“. Näiteks:

```
main : IO ()
main = randomRIO (1, 10) >>= classify >>= putStrLn
   where classify : Int → IO String
         classify x = if x `mod` 2 == 1 then
                          pure "paaritu"
                      else
                          pure "paaris"
```

do-süntaks I

Eelneval slaidil olnud koodi on keeruline lugeda ja kirjutada:

```
main : IO ()
main = randomRIO (1, 10) >>= classify >>= putStrLn
   where classify : Int → IO String
         classify x = if x `mod` 2 == 1 then
                          pure "paaritu"
                      else
                          pure "paaris"
```

Sama saab saavutada järgnevalt

```
main : IO ()
main = do
   r ← randomRIO (1, 10)
   c ← classify r
   putStrLn c
      where classify : Int → IO String
            classify x = ...
```

või

```
main : IO ()
main = do
   r ← randomRIO (1, 10)
   if r `mod` 2 == 1
    then putStrLn "paaritu"
    else putStrLn "paaris"
```

do-süntaks II

Näide

```
proc : IO ()
proc = do
   s ← getLine
   let n : Int
       n = read s
       n2 : Int
       n2 = 2*n
   putStrLn ("Kaks␣korda␣" ++ s ++ "␣on␣" ++ show n2)
```

Do-süntaks algab `do`-võtmesõnaga, millele järgnevad *järjest töödeldavad* laused.

- Laused mustriga $x \leftarrow p$, kus p : `IO` a siis $x : a$,
- `let` laused ning
- avaldised e, mille tüüp on `IO a`.

Mitme `do` kasutamine

- `do` seob kokku IO-avaldised, kuid ei saa vaadata konstruktsioonide sisse
- S.t. ühe avaldise jaoks pole do-d vaja
  - `main = putStrLn "Hello World!"`
- Hargenmise puhul võib olla vaja kasutada mitut do-d:

```
main : IO ()
main = do
   putStrLn "Kirjuta midagi!"
   xs ← getLine
   if (xs=="")
    then putStr "Sõnakuulmatu!"
    else do
       putStrLn "Tänan!"
       putStrLn ("Kirjutasid: " ++ xs)
```

`do` tähendus

- `a >>= f` on sama mis

```
do x ← a
   f x
```

- `a >> b` on sama mis

```
do a
   b
```

```
do a                    do do a                      do a
   b    on sama mis           b    on sama mis          do b
   c                       c                             c
```

- Fixity:

```
infixl 1 >>
infixl 1 >>=
```

Näide

```
main : IO ()
main = do
 putStrLn "Kirjuta␣midagi!"
 xs ← getLine
 if (xs=="")
   then putStr "Sõnakuulmatu!"
   else do
     putStrLn "Tänan!"
     putStrLn ("Kirjutasid:␣" ++ xs)
```

on sama mis

```
main : IO ()
main =
 putStrLn "Kirjuta␣midagi!" >>
 getLine  >>= (λ xs ⇒
 if (xs=="")
   then putStr "Sõnakuulmatu!"
   else
     putStrLn "Tänan!" >>
     putStrLn ("Kirjutasid:␣" ++ xs)
 )
```

Sisend-väljund

+ `IO` on hea näide, kuidas mittepuhtaid arvutusi saab modelleerida puhaste funktsioonide abil.
+ Selline modelleerimine on teoreetiliselt huvitav, kuna võimaldab katsetada erinevaid võimalusi ja kitsendusi. Näiteks erindid ja jätkud.

- Tekitab palju segadust, kui mõisted pole (veel) selged.
- Praktiline kasu pole ilmne: lihtsam kasutada mittepuhast programmeerimiskeelt.

Seetõttu: Selles kursuses harjutame `IO`-d aga ei lähe seda rada kaugemale.

Paarid ja ennikud

- Paarid

$$\begin{array}{lcl} \text{fst} & \equiv & \lambda p.\ p\ \text{true} \\ \text{snd} & \equiv & \lambda p.\ p\ \text{false} \\ (E_1, E_2) & \equiv & \lambda f.\ f\ E_1\ E_2 \end{array}$$

- Ennikud

$$\begin{array}{lcl} (E_1, \ldots, E_n) & \equiv & (E_1, (\ldots (E_{n-1}, E_n) \ldots)) \\ E \overset{n}{\downarrow} 1 & \equiv & \text{fst}\ E \\ E \overset{n}{\downarrow} 2 & \equiv & \text{fst}\ (\text{snd}\ E) \\ & \ldots & \\ E \overset{n}{\downarrow} i & \equiv & \text{fst}\ (\text{snd}^{i-1} E) \\ & \ldots & \\ E \overset{n}{\downarrow} n & \equiv & \text{snd}^{n-1}\ E \end{array}$$

Naturaalarvud

- Standardnumbrid

$$
\begin{array}{lcll}
\ulcorner 0 \urcorner & \equiv & \lambda x.\ x & (\equiv \ \mathrm{I}) \\
\ulcorner n{+}1 \urcorner & \equiv & (\mathrm{false}, \ulcorner n \urcorner) & \\
\mathrm{succ} & \equiv & \lambda n.\ (\mathrm{false}, n) & \\
\mathrm{pred} & \equiv & \lambda n.\ n\ \mathrm{false} & (\equiv \ \mathrm{snd}) \\
\mathrm{iszero} & \equiv & \lambda n.\ n\ \mathrm{true} & (\equiv \ \mathrm{fst})
\end{array}
$$

- Liitmine (?!)

$$\mathrm{add} \quad = \quad \lambda x\ y.\ \mathrm{cond}(\mathrm{iszero}\ x)\ y\ (\mathrm{add}(\mathrm{pred}\ x)(\mathrm{succ}\ y))$$

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

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