Lihtne funktsonaalkeel

Süntaks

$$
\begin{aligned}
\textbf{type}\ \mathit{Var} &= \mathit{String} \\
\textbf{data}\ \mathit{Term} &= \mathit{V}\ \mathit{Var} \\
&\mid \mathit{Term}\ \mathit{:@}\ \mathit{Term} \\
&\mid \mathit{L}\ \mathit{Var}\ \mathit{Term} \\
&\mid \mathit{N}\ \mathit{Integer} \\
&\mid \mathit{TT} \mid \mathit{FF} \\
&\mid \mathit{If}\ \mathit{Term}\ \mathit{Term}\ \mathit{Term} \\
&\mid \mathit{Rec}\ \mathit{Term}
\end{aligned}
$$

Lihtne funktsonaalkeel

Semantilised kategooriad

$$
\begin{aligned}
\textbf{data}\ \mathit{Val} &= \mathit{I}\ \mathit{Integer} \\
&\mid \mathit{B}\ \mathit{Bool} \\
&\mid \mathit{F}\ (\mathit{Val} \rightarrow \mathit{Val}) \\
\textbf{type}\ \mathit{Env} &= [(\mathit{Var}, \mathit{Val})]
\end{aligned}
$$

Lihtne funktsonaalkeel

Keskonnaga opereerimine

$$
\begin{aligned}
&empty :: [(a, b)] \\
&empty = [\,] \\
&update :: a \to b \to [(a, b)] \to [(a, b)] \\
&update\ a\ b\ abs = (a, b) : abs \\
&unsafeLookup :: Eq\ a \Rightarrow a \to [(a, b)] \to b \\
&unsafeLookup\ x\ ((a, b) : abs) \\
&\qquad \mid x \equiv a \qquad = b \\
&\qquad \mid otherwise = unsafeLookup\ x\ abs
\end{aligned}
$$

Lihtne interpretaator

Standardinterpretaator

$$
\begin{array}{lll}
ev :: Term \to Env \to Val & & \\
ev\ (V\ x) & env = unsafeLookup\ x\ env & \\
ev\ (e0\ :@\ e1) & env = \mathbf{case}\ ev\ e0\ env\ \mathbf{of} & \\
 & \quad F\ f \to f\ (ev\ e1\ env) & \\
ev\ (L\ x\ e) & env = F\ (\lambda v \to ev\ e\ (update\ x\ v\ env)) & \\
ev\ (N\ n) & env = I\ n & \\
ev\ TT & env = B\ True & \\
ev\ FF & env = B\ False & \\
ev\ (If\ e\ e0\ e1) & env = \mathbf{case}\ ev\ e\ env\ \mathbf{of} & \\
 & \quad B\ b \to \mathbf{if}\ b\ \mathbf{then}\ ev\ e0\ env & \\
 & \qquad\qquad \mathbf{else}\ \ ev\ e1\ env & \\
ev\ (Rec\ e) & env = \mathbf{case}\ ev\ e\ env\ \mathbf{of} & \\
 & \quad F\ f \to f\ (ev\ (Rec\ e)\ env) &
\end{array}
$$

Lihtne interpretaator

Algkeskkond

```
env0 = [("+",    mkArithOp2 (+))
        ...
        ,("==",  mkCompOp2 (≡))
        ...
        ,("||",  mkBoolOp2 (∨))
        ,("not", mkBoolOp1 (¬))
        ]
  where mkArithOp2 op  = F $ λ(I i1) →
                           F $ λ(I i2) → I (i1 `op` i2)
        mkCompOp2 op = F $ λ(I i1) →
                           F $ λ(I i2) → B (i1 `op` i2)
        mkBoolOp2 op   = F $ λ(B b1) →
                           F $ λ(B b2) → B (b1 `op` b2)
        mkBoolOp1 op   = F $ λ(B b) → B (op b)
eval t = ev t env0
```

Lihtne interpretaator

Näide

```
fact = Rec $ L "fact" $ L "x" $
         If (V "<=" :@ V "x" :@ N 1)
            (N 1)
            (V "*" :@ V "x"
                   :@ (V "fact" :@ (V "-" :@ V "x" :@ N 1)))

Main> eval $ fact :@ N 5
120
```

Monaadilised interpretaatorid

Semantilised kategooriad

$$
\begin{aligned}
\textbf{data}\ \mathit{Val}\ t\ &= \mathit{I}\ \mathit{Integer} \\
&\mid \mathit{B}\ \mathit{Bool} \\
&\mid \mathit{F}\ (\mathit{Val}\ t \to t\ (\mathit{Val}\ t)) \\
\textbf{type}\ \mathit{Env}\ t &= [(\mathit{Var}, \mathit{Val}\ t)]
\end{aligned}
$$

Väärtustusmonaad

$$
\begin{aligned}
&\textbf{class}\ \mathit{Monad}\ t \Rightarrow \mathit{MonadEv}\ t\ \textbf{where} \\
&\qquad \mathit{ev} :: \mathit{Term} \to \mathit{Env}\ t \to t\ (\mathit{Val}\ t) \\
&\mathit{eval}\ t = \mathit{ev}\ t\ \mathit{env0}
\end{aligned}
$$

Monaadilised interpretaatorid

Algkeskkond

```
env0 = [("+",    mkArithOp2 (+))
        ...
        ,("not", mkBoolOp1 (¬))
        ]
  where mkArithOp2 op  = F $ λ(I i1) → return $
                         F $ λ(I i2) → return $
                           I (i1 `op` i2)
        mkCompOp2 op = F $ λ(I i1) → return $
                         F $ λ(I i2) → return $
                           B (i1 `op` i2)
        mkBoolOp2 op   = F $ λ(B b1) → return $
                         F $ λ(B b2) → return $
                           B (b1 `op` b2)
        mkBoolOp1 op   = F $ λ(B b) → return $
                           B (op b)
```

Monaadilised interpretaatorid

Baaskonstruktsioonide interpreteerimine

```
_ev :: MonadEv t => Term -> Env t -> t (Val t)
_ev (V x)      env = return $ unsafeLookup x env
_ev (e0 :@ e1) env = do F f <- ev e0 env
                        v   <- ev e1 env
                        f v
_ev (L x e)    env = return $ F (\v -> ev e (update x v env))
_ev (N n)      env = return $ I n
_ev TT         env = return $ B True
_ev FF         env = return $ B False
_ev (If e e0 e1) env = do B b <- ev e env
                          if b then ev e0 env
                               else ev e1 env
_ev (Rec e)    env = do F f <- ev e env
                        v   <- _ev (Rec e) env
                        f v
```

Monaadilised interpretaatorid

Standardinterpretaator

$$\textbf{newtype}\ Id\ a = Id\ a$$

$$\textbf{instance}\ Monad\ Id\ \textbf{where}$$
$$\quad return\ a\ = Id\ a$$
$$\quad Id\ a \gg\!= k = k\ a$$

$$\textbf{instance}\ MonadEv\ Id\ \textbf{where}$$
$$\quad ev\ e\ env\ = \_ev\ e\ env$$

Monaadilised interpretaatorid

Süntaks: erindid

```
data Term = ...
          | Raise
          | Term `Handle` Term
```

Erinditöötlusega interpretaator

```
instance MonadEv Maybe where
  ev Raise env              = mzero
  ev (e0 `Handle` e1) env = ev e0 env `mplus` ev e1 env
  ev e env                  = _ev e env
```

Monaadilised interpretaatorid

Süntaks: mittedeterminism

```
data Term = ...
          | Deadlock
          | Term `Choice` Term
```

Mittedetermineeritud interpretaator

```
instance MonadEv [] where
  ev Deadlock env          = []
  ev (e0 `Choice` e1) env = ev e0 env ++ ev e1 env
  ev e env                 = _ev e env
```

Monaadilised interpretaatorid

Näited

```
Main> eval $ V "+" :@ N 2 :@ N 3 :: Id (Val Id)
Id 5
Main> eval $ V "+" :@ N 2 :@ N 3 :: Maybe (Val Maybe)
Just 5
Main> eval $ N 2 `Handle` N 3 :: Maybe (Val Maybe)
Just 2
Main> eval $ Raise `Handle` N 3 :: Maybe (Val Maybe)
Just 3
Main> eval $ V "+" :@ N 2 :@ (N 3 `Choice` N 4) :: [Val []]
[5,6]
```

Monaadilised interpretaatorid

Näited

```
Main> eval $ fact :@ N 5 :: Id (Val Id)
Id 120
```

NB!

Rekursiooni väärtustamine toimub liiga agaralt!

Monaadilised interpretaatorid

Näited

```
Main> eval $ fact :@ N 5 :: Id (Val Id)
Id 120

Main> eval $ fact :@ N 5 :: Maybe (Val Maybe)
ERROR - Control stack overflow
```

NB!

Rekursiooni väärtustamine toimub liiga agaralt!

Monaadilised interpretaatorid

Näited

```
Main> eval $ fact :@ N 5 :: Id (Val Id)
Id 120

Main> eval $ fact :@ N 5 :: Maybe (Val Maybe)
ERROR - Control stack overflow
```

NB!

Rekursiooni väärtustamine toimub liiga agaralt!

Monaadilised interpretaatorid

Monaadiline püsipunktioperaator

```
class Monad t ⇒ MonadFix t where
  mfix :: (a → t a) → t a

fix :: (a → a) → a
fix f = f (fix f)

instance MonadFix Id where
  mfix f = fix (f ∘ unId)
           where unId (Id x) = x

instance MonadFix Maybe where
  mfix f = fix (f ∘ unJust)
           where unJust (Just x) = x

instance MonadFix [] where
  mfix f = fix (f ∘ unSglt)
           where unSglt [x] = x
```

Monaadilised interpretaatorid

Monaadiline püsipunktioperaator

$$
\begin{aligned}
&\textbf{instance } \mathit{MonadFix}\ ((\rightarrow)\ e)\ \textbf{where}\\
&\quad \mathit{mfix}\ f = \lambda e \rightarrow \textbf{let } a = f\ a\ e\\
&\qquad\qquad\qquad\qquad \textbf{in } a\\
&\textbf{instance } \mathit{MonadFix}\ (\mathit{StM}\ s)\ \textbf{where}\\
&\quad \mathit{mfix}\ f = \mathit{StM}\ \$\ \lambda s \rightarrow \textbf{let } (a, s') = \mathit{runStM}\ (f\ a)\ s\\
&\qquad\qquad\qquad\qquad\qquad\qquad \textbf{in } (a, s')
\end{aligned}
$$

Monaadilised interpretaatorid

Rekursiooni väärtustamine

$$
\begin{aligned}
&\textbf{class}\ MonadFix\ t \Rightarrow MonadEv\ t\ \textbf{where} \\
&\qquad ev :: Term \to Env\ t \to t\ (Val\ t) \\
&\_ev :: MonadEv\ t \Rightarrow Term \to Env\ t \to t\ (Val\ t) \\
&\ldots \\
&\_ev\ (Rec\ e)\ env = \textbf{do}\ F\ f \leftarrow ev\ e\ env \\
&\qquad\qquad\qquad\qquad\qquad\quad mfix\ f
\end{aligned}
$$

Monaadilised interpretaatorid

Näited

```
Main> eval $ fact :@ N 5 :: Id (Val Id)
Id 120
Main> eval $ fact :@ N 10 :: Maybe (Val Maybe)
Just 3628800
Main> eval $ fact :@ N 20 :: [Val []]
[2432902008176640000]
```

Monaadilised interpretaatorid

Väljundmonaad

```
newtype Output a = O (a, String)
instance Monad Output where
  m >>= f  = let O (x1, o1) = m
                 O (x2, o2) = f x1
             in O (x2, o1 ++ o2)
  return x = O (x, "")
output :: String -> Output ()
output s = O ((), s)
instance MonadFix Output where
  mfix f = fix (f . unO)
           where unO (O (x, o)) = x
```

Monaadilised interpretaatorid

Süntaks: väljund

```
data Term = ...
          | Out Term
```

Varjatud parameetriga interpretaator

```
instance MonadEv Output where
  ev (Out e) env = do v ← ev e env
                      output (show v)
                      return v
  ev e env       = _ev e env
```

Monaadilised interpretaatorid

Süntaks: varjatud parameeter

$$\begin{aligned}\textbf{data}\ \mathit{Term} = \quad & \ldots \\ \mid\ & \mathit{UseX}\end{aligned}$$

Varjatud parameetriga interpretaator

$$\begin{aligned}&\textbf{instance}\ \mathit{MonadEv}\ ((\rightarrow)\ \mathit{Integer})\ \textbf{where} \\ &\quad \mathit{ev}\ \mathit{UseX}\ \mathit{env} = \textbf{do}\ x \leftarrow \mathit{getEnv} \\ &\qquad\qquad\qquad\qquad\quad \mathit{return}\ (I\ x) \\ &\quad \mathit{ev}\ e\ \mathit{env} = \_\mathit{ev}\ e\ \mathit{env}\end{aligned}$$

NB!

Kasutab ülekattuvaid isendeid!

Monaadilised interpretaatorid

Süntaks: globaalne olek

$$
\begin{aligned}
&\textbf{data } \mathit{Term} = \ \ldots \\
&\qquad\qquad \mid \ \mathit{GetS} \\
&\qquad\qquad \mid \ \mathit{SetS}\ \mathit{Term}
\end{aligned}
$$

Varjatud parameetriga interpretaator

$$
\begin{aligned}
&\textbf{instance } \mathit{MonadEv}\ (\mathit{StM}\ \mathit{Integer})\ \textbf{where} \\
&\quad \mathit{ev}\ \mathit{GetS}\ \mathit{env} \quad\quad = \textbf{do } x \leftarrow \mathit{getStM} \\
&\qquad\qquad\qquad\qquad\quad\ \ \mathit{return}\ (I\ x) \\
&\quad \mathit{ev}\ (\mathit{SetS}\ e)\ \mathit{env} = \textbf{do } I\ x \leftarrow \mathit{ev}\ e\ \mathit{env} \\
&\qquad\qquad\qquad\qquad\quad\ \ \mathit{setStM}\ x \\
&\qquad\qquad\qquad\qquad\quad\ \ \mathit{return}\ (I\ x) \\
&\quad \mathit{ev}\ e\ \mathit{env} \quad\quad\quad\ = \_\mathit{ev}\ e\ \mathit{env}
\end{aligned}
$$

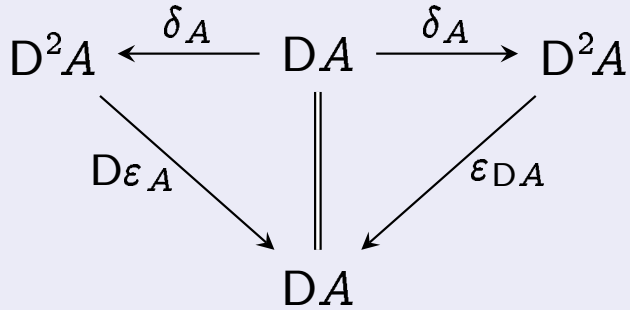
Komonaadid

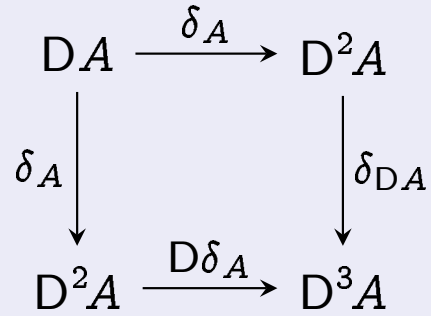
Definitsioon

Komonaad kategoorial $\mathcal{C}$ koosneb:

- endofunktorist $\mathrm{D} : \mathcal{C} \to \mathcal{C}$
- loomulikust teisendusest $\varepsilon_A : \mathrm{D}A \to A$
  - nn. komonaadi ühik
- loomulikust teisendusest $\delta_A : \mathrm{D}A \to \mathrm{DD}A$
  - nn. komonaadi komultiplikatsioon

selliselt, et järgnevad diagrammid kommuteeruvad:





Komonaadid

Definitsioon

Ko-Kleisli kolmik kategoorial $\mathcal{C}$ koosneb:

- kujutisest $\mathrm{D} : \mathrm{Obj}(\mathcal{C}) \to \mathrm{Obj}(\mathcal{C})$
- $\mathrm{Obj}(\mathcal{C})$-indekseeritud kujutiste perest $\varepsilon_A : \mathrm{D}A \to A$
- operatsioonist $-^\dagger$, mis igale morfismile $f : \mathrm{D}A \to B$ seab vastavusse morfismi $f^\dagger : \mathrm{D}A \to \mathrm{D}B$
  - nn. komonaadi ekstensioon

selliselt, et iga $f : \mathrm{D}A \to B$ ja $g : \mathrm{D}B \to C$ korral kehtivad järgmised võrdused:

$$\begin{aligned} \varepsilon_A{}^\dagger &= id_{\mathrm{D}A} \\ \varepsilon_B \cdot f^\dagger &= f \\ (g \cdot f^\dagger)^\dagger &= g^\dagger \cdot f^\dagger \end{aligned}$$

Komonaadid

Lemma

Komonaadid ja ko-Kleisli kolmikud on ekvivalentsed mõisted.

Tõestus

Olgu $f : \mathsf{D}A \to B$ ja $g : A \to B$

- $(\mathsf{D}, \varepsilon, \delta) \Longrightarrow (\mathsf{D}, \varepsilon, -^\dagger)$

$$f^\dagger = \mathsf{D}f \cdot \delta_A$$

- $(\mathsf{D}, \varepsilon, -^\dagger) \Longrightarrow (\mathsf{D}, \varepsilon, \delta)$

$$\begin{aligned} \mathsf{D}g &= (g \cdot \varepsilon_A)^\dagger \\ \delta_A &= id_{\mathsf{D}A}{}^\dagger \end{aligned}$$

Komonaadid

Intuitsioon

- Komonaadid modellerivad kontekstist sõltuvaid arvutusi.
- Kui A on väärtusi esitav tüüp, siis $\mathrm{D}A$ on kontekstis sõltuv väärtus.
- Kontekstist sõltuv funktsioon tüübist A tüüpi B on funktsioon tüüpi $\mathrm{D}A \to B$.
- Komonaadi ühik $\varepsilon_A : \mathrm{D}A \to A$ ignoreerib konteksti ning väljastab puhta väärtuse.
- Komonaadi ekstensioonioon $f^\dagger : \mathrm{D}A \to \mathrm{D}B$ arvutab etteantud kontekstist sõltuva funktsiooni $f : \mathrm{D}A \to B$ põhjal uue väärtuse säilitades konteksti.
- Konteksti muutvad arvutused on komonaadi spetsiifilised.

Komonaadid Haskellis

Komonaadide klass

$$
\begin{aligned}
&\textbf{class } Comonad\ d \textbf{ where}\\
&\quad counit\ \ :: d\ a \to a\\
&\quad cobind\ \ :: (d\ a \to b) \to d\ a \to d\ b\\
&cmap\ \ \ \ :: Comonad\ d \Rightarrow (a \to b) \to d\ a \to d\ b\\
&cmap\ f\ x = cobind\ (f \circ counit)\ x\\
&cdup\ \ \ \ :: Comonad\ d \Rightarrow d\ a \to d\ (d\ a)\\
&cdup\ \ \ \ = cobind\ id
\end{aligned}
$$

NB!

Komonaadi defineerimisel peab kontrollima, et ta rahuldaks komonaadi seadusi.

Komonaadid Haskellis

Identsuskomonaad

$$
\begin{aligned}
&\textbf{instance } \mathit{Comonad}\ \mathit{Id} \textbf{ where} \\
&\quad \mathit{counit}\ (\mathit{Id}\ a) = a \\
&\quad \mathit{cobind}\ k\ d \quad = \mathit{Id}\ (k\ d)
\end{aligned}
$$

NB!

Identsuskomonaad esitab ”puhtaid” (so. kontekstist sõltumatuid) arvutusi.

Komonaadid Haskellis

Korrutiskomonaad

```
instance Comonad ((,) e) where
  counit (_, x)         = x
  cobind k d@(e, _) = (e, k d)
askP                :: (e, a) → e
askP (e, _)         = e
localP              :: (e → e) → (e, a) → (e, a)
localP f (e, x) = (f e, x)
```

NB!

Korrutiskomonaad esitab ”tavakeskkonnast” sõltuvaid arvutusi.

Komonaadid Haskellis

Striimide komonaad

$$\textbf{data}\ \mathit{Stream}\ a = a :< \mathit{Stream}\ a$$

$$\begin{aligned}&\textbf{instance}\ \mathit{Comonad}\ \mathit{Stream}\ \textbf{where}\\ &\quad \mathit{counit}\ (x :< xs) = x\\ &\quad \mathit{cobind}\ k\ d@(x :< xs) = k\ d :< \mathit{cobind}\ k\ xs\end{aligned}$$

$$\begin{aligned}&\mathit{fbyS} :: a \to \mathit{Stream}\ a \to \mathit{Stream}\ a\\ &\mathit{fbyS}\ x\ xs = x :< xs\end{aligned}$$

$$\begin{aligned}&\mathit{nextS} :: \mathit{Stream}\ a \to \mathit{Stream}\ a\\ &\mathit{nextS}\ (x :< xs) = xs\end{aligned}$$

NB!

Mis liiki arvutusi esitab striimide komonaad?

Andmevooarvutused

Striimid ja signaalid

- Striimid esitavad diskreetse ajaga signaale.
- Striimid (lõpmatud jadad) A^ω on isomorfsed funktsioonidega $\mathbf{N} \to A$.

Isomorfismi realistatsioon Haskellis

$$
\begin{aligned}
&\mathit{str2fun} :: \mathit{Stream}\ a \to \mathit{Int} \to a \\
&\mathit{str2fun}\ (x :< xs)\ 0 = x \\
&\mathit{str2fun}\ (x :< xs)\ n = \mathit{str2fun}\ xs\ (n-1) \\
&\mathit{fun2str} :: (\mathit{Int} \to a) \to \mathit{Stream}\ a \\
&\mathit{fun2str}\ f = \mathit{fun2str'}\ f\ 0 \\
&\quad \textbf{where}\ \mathit{fun2str'}\ f\ n = f\ n :< \mathit{fun2str'}\ f\ (n+1)
\end{aligned}
$$

Andmevooarvutused

Striimifunktsioonid ja andmevooarvutused

- Andmevooarvutused on (diskreetse ajaga) signaalide teisendajad.
- Ehk siis striimifunktsioonid $A^\omega \to B^\omega$.

Näiteid andmevooprogrammidest

$$
\begin{aligned}
pos &= 0 \; fby \; (pos + 1) \\
sum\ x &= x + (0 \; fby \; (sum\ x)) \\
fact &= 1 \; fby \; (fact * (pos + 1)) \\
fibo &= 0 \; fby \; (fibo + (1 \; fby \; fibo))
\end{aligned}
$$

pos	0	1	2	3	4	5	6	...
sum pos	0	1	3	6	10	15	21	...
fact	1	1	2	6	24	120	720	...
fibo	0	1	1	2	3	5	8	...

Andmevooarvutused

Striimifunktsioonid

Striimifunktsioonid on loomulikult isomorfsed funktsioonidega:

$$\begin{aligned} A^\omega \to B^\omega &\cong A^\omega \to (\mathbf{N} \Rightarrow B) \\ &\cong A^\omega \times \mathbf{N} \to B \end{aligned}$$

Positsiooniga striimide komonaad

Striimid koos positsiooniga moodustavad komonaadi:

$$\begin{aligned} \mathrm{D}A &= A^\omega \times \mathbf{N} \\ \varepsilon_A(\alpha, n) &= \alpha_n \\ f^\dagger(\alpha, n) &= (\beta, n) \\ &\quad \text{kus } \forall i.\ \beta_i = f(\alpha, i) \end{aligned}$$

NB!

Striimifunktsioonid vs. olekuteisendajad.

Andmevooarvutused

Positsiooniga striimid

- Positsiooniga striimis (α, n) vastab striim α striimifunktsiooni sisendile ning positsioon n on väljundstriimi meid hetkel huvitava elemendi indeks.
- Sisendstriimi saame jaotada positsiooni suhtes kolmeks:

$$a_0, a_1, \ldots, a_{n-1}, \boxed{a_n}, a_{n+1}, a_{n+2}, \ldots$$

  - ajalugu — antud positsioonist väiksemate indeksitega elemendid (neid on lõplik hulk);
  - olevik — antud positsioonis olev element (täpselt üks);
  - tulevik — antud positsioonist suuremate indeksitega elemendid (neid on lõpmatu hulk).

Andmevooarvutused

Positsiooniga striimide alternatiivseid esitusi

$$
\begin{aligned}
\mathrm{D}A &= A^{\omega} \times \mathbf{N} \\
&\cong (\mathbf{N} \Rightarrow A) \times \mathbf{N} \\
&\cong A^{\star} \times A \times A^{\omega} \\
&\cong A^{+} \times A^{\omega} \\
&\cong A^{\star} \times A^{\omega}
\end{aligned}
$$

Andmevooarvutused

Kausaalsed striimifunktsioonid

Sünkroonsed andmevooarvutused esitavad mitte üldisi vaid ainult kausaalseid striimifunktsioone

- s.o. hetke väljundväärtus võib sõltuda ainult hetke sisendist ning varasematest sisenditest, aga mitte tulevikust;
- seega, kausaalsed striimifunktsioonid on funktsioonid kujul $A^+ \to B$.

Mittetühjade listide komonaad

$$\begin{aligned} \mathsf{D}A &= A^+ \\ \varepsilon_A[\alpha_1, \ldots, \alpha_n] &= \alpha_n \\ f^\dagger[\alpha_1, \ldots, \alpha_n] &= [f[\alpha_1],\ f[\alpha_1, \alpha_2], \ldots, f[\alpha_1, \ldots, \alpha_n]] \end{aligned}$$

Andmevooarvutused

Antikausaalsed striimifunktsioonid

- Atikausaalsed on striimifunktsioonid, millede hetke väljundväärtus võib sõltuda ainult hetke sisendist ning tulevikus saabuvatest sisenditest, aga mitte minevikust.
- Pole just väga praktiline, aga ...
- kausaalsed striimifunktsioonid on funktsioonid kujul $A^\omega \to B$.

NB!

Striimide komonaad esitab antikausaalseid striimifunktsioone!

Komonaadid Haskellis

FunArg komonaad

$$\textbf{data}\ \mathit{FunArg}\ a = (\mathit{Int} \to a) :\# \mathit{Int}$$

$$\begin{aligned}
&\textbf{instance}\ \mathit{Comonad}\ \mathit{FunArg}\ \textbf{where}\\
&\quad \mathit{counit}\ (f :\# i) \quad = f\ i\\
&\quad \mathit{cobind}\ k\ (f :\# i) = (\lambda i' \to k\ (f :\# i')) :\# i
\end{aligned}$$

$$\begin{aligned}
&\mathit{fbyFA} :: a \to \mathit{FunArg}\ a \to a\\
&\mathit{fbyFA}\ a\ (f :\# 0) \qquad = a\\
&\mathit{fbyFA}\ _\ (f :\# (i+1)) = f\ i
\end{aligned}$$

$$\begin{aligned}
&\mathit{nextFA} :: \mathit{FunArg}\ a \to a\\
&\mathit{nextFA}\ (f :\# i) = f\ (i+1)
\end{aligned}$$

$$\begin{aligned}
&\mathit{runFA} :: (\mathit{FunArg}\ a \to b) \to \mathit{Stream}\ a \to \mathit{Stream}\ b\\
&\mathit{runFA}\ k\ as = \mathit{runFA'}\ k\ (\mathit{str2fun}\ as :\# 0)\\
&\quad \textbf{where}\ \mathit{runFA'}\ k\ d@(f :\# i) = k\ d :< \mathit{runFA'}\ k\ (f :\# (i+1))
\end{aligned}$$

Komonaadid Haskellis

LVS komonaad

```
data List a = Nil | List a :> a
data LV a   = List a := a
data LVS a = LV a :| Stream a

instance Comonad LVS where
  counit (az := a :| as) = a
  cobind k d = cobindL d := k d :| cobindS d
    where cobindL (Nil       := a :| as)  = Nil
          cobindL (az' :> a' := a :| as)  = cobindL d' :> k d'
            where d' = az' := a' :| (a :< as)
          cobindS (az := a :| (a' :< as')) = k d' :< cobindS d'
            where d' = az :> a := a' :| as'
```

Komonaadid Haskellis

LVS komonaadi operatsioonid

$$
\begin{aligned}
&fbyLVS :: a \to LVS\ a \to a \\
&fbyLVS\ a0\ (Nil\ \ \ \ := \_ :|\ \_) = a0 \\
&fbyLVS\ \_\ ((\_ :> a') := \_ :|\ \_) = a' \\
&nextLVS :: LVS\ a \to a \\
&nextLVS\ (\_ := \_ :|\ (a :< \_)) = a \\
&runLVS :: (LVS\ a \to b) \to Stream\ a \to Stream\ b \\
&runLVS\ k\ (a' :< as') = runLVS'\ k\ (Nil := a' :|\ as') \\
&\quad \textbf{where}\ runLVS'\ k\ d@(az := a :|\ (a' :< as')) \\
&\qquad\qquad\qquad\qquad = k\ d :< runLVS'\ k\ (az :> a := a' :|\ as')
\end{aligned}
$$

Komonaadid Haskellis

LV komonaad

```
instance Comonad LV where
  counit (_ := a)          = a
  cobind k d@(az := _) = cobindL k az := k d
    where cobindL k Nil        = Nil
          cobindL k (az :> a) = cobindL k az :> k (az := a)

fbyLV :: a → LV a → a
fbyLV a0 (Nil    := _) = a0
fbyLV _ ((_ :> a') := _) = a'

runLV :: (LV a → b) → Stream a → Stream b
runLV k (a' :< as') = runLV' k (Nil := a') as'
  where runLV' k d@(az := a) (a' :< as')
                    = k d :< runLV' k (az :> a := a') as'
```

Komonaadilised interpretaatorid

Semantilised kategooriad

$$
\begin{aligned}
\textbf{data}\ \mathit{Val}\ d\ &= \mathit{I}\ \mathit{Integer} \\
&\mid \mathit{B}\ \mathit{Bool} \\
&\mid \mathit{F}\ (d\ (\mathit{Val}\ d) \to \mathit{Val}\ d) \\
\textbf{type}\ \mathit{Env}\ d &= [(\mathit{Var}, \mathit{Val}\ d)]
\end{aligned}
$$

Väärtustuskomonaad

$$
\begin{aligned}
&\textbf{class}\ \mathit{Comonad}\ d \Rightarrow \mathit{ComonadEv}\ d\ \textbf{where} \\
&\qquad \mathit{ev} :: \mathit{Term} \to d\ (\mathit{Env}\ d) \to \mathit{Val}\ d
\end{aligned}
$$

Komonaadilised interpretaatorid

Baaskonstruktsioonide interpreteerimine

$$
\begin{array}{lll}
\_ev :: ComonadEv\ d \Rightarrow Term \to d\ (Env\ d) \to Val\ d & & \\
\_ev\ (V\ x) & denv = unsafeLookup\ x\ (counit\ denv) & \\
\_ev\ (e0\ :@\ e1) & denv = \mathbf{case}\ ev\ e0\ denv\ \mathbf{of} & \\
 & \quad F\ f \to f\ (cobind\ (ev\ e1)\ denv) & \\
\_ev\ (L\ x\ e) & denv = \ldots & \\
\_ev\ (N\ n) & denv = I\ n & \\
\_ev\ TT & denv = B\ True & \\
\_ev\ FF & denv = B\ False & \\
\_ev\ (If\ e\ e0\ e1) & denv = \mathbf{case}\ ev\ e\ denv\ \mathbf{of} & \\
 & \quad B\ b \to \mathbf{if}\ b\ \mathbf{then}\ ev\ e0\ denv & \\
 & \qquad\qquad \mathbf{else}\ \ ev\ e1\ denv & \\
\_ev\ (Rec\ e) & denv = \ldots &
\end{array}
$$

Komonaadilised interpretaatorid

Rekursiooni väärtustamine

$$\_ev\ (Rec\ e)\ denv = \mathbf{case}\ ev\ e\ denv\ \mathbf{of}$$
$$F\ f \to f\ (cobind\ (\_ev\ (Rec\ e))\ denv)$$

NB!

Komonaadide korral "naiivne" rekursiooni realisatsioon tööab; sh. rekursioonioperaator on uniformne kõikidele komonaadidele.

Komonaadiline rekursioonioperaator

$$cfix :: Comonad\ d \Rightarrow d\ (d\ a \to a) \to a$$
$$cfix\ f = counit\ f\ (cobind\ cfix\ f)$$

Komonaadilised interpretaatorid

Lambda-abstraktsiooni väärtustamine

$$\_ev\ (L\ x\ e)\ denv = F\ (\lambda d \to ev\ e\ (extend\ x\ d\ denv))$$

NB!

- Funktsioon *extend* tüüp peaks olema:

$$extend :: Comonad\ d \Rightarrow Var \to d\ (Val\ d) \to d\ (Env\ d) \to d\ (Env\ d)$$

- Ta peab laiendama keskkonda selliselt, et muutuja x oleks seotud muutujas d "oleva" väärtusega.
- Seejuures peab väljundkontekst ühendama mõlemad sisendkontekstid.

Komonaadilised interpretaatorid

Klass *ComonadZip*

$$\textbf{class } \mathit{Comonad}\ d \Rightarrow \mathit{ComonadZip}\ d \textbf{ where}$$
$$\quad \mathit{czip} :: d\ a \to d\ b \to d\ (a, b)$$

NB!

Funktsioon *czip* peab rahuldama järgmist võrdust:

$$\mathit{czip} \circ (\lambda d \to (d, d)) = \mathit{cmap}\ (\lambda x \to (x, x))$$

Komonaadilised interpretaatorid

Kontekstide ühendamine identsuskomonaadis

$$\textbf{instance } \mathit{ComonadZip}\ \mathit{Id} \textbf{ where}$$
$$\quad \mathit{czip}\ (\mathit{Id}\ a)\ (\mathit{Id}\ b) = \mathit{Id}\ (a, b)$$

Kontekstide ühendamine striimide komonaadis

$$\textbf{instance } \mathit{ComonadZip}\ \mathit{Stream} \textbf{ where}$$
$$\quad \mathit{czip}\ (x :< xs)\ (y :< ys) = (x, y) :< \mathit{czip}\ xs\ ys$$

Kontekstide ühendamine *FunArg* komonaadis

$$\textbf{instance } \mathit{ComonadZip}\ \mathit{FunArg} \textbf{ where}$$
$$\quad \mathit{czip}\ (f :\# i)\ (g :\# j) \mid i \equiv j = (\lambda n \to (f\ n, g\ n)) :\# i$$

Komonaadilised interpretaatorid

Kontekstide ühendamine LV komonaadis

$$
\begin{aligned}
&zipL :: List\ a \to List\ b \to List\ (a, b) \\
&zipL\ (xz :> x)\ (yz :> y) = zipL\ xz\ yz :> (x, y) \\
&zipL\ \_\ \ \_\ \ = Nil
\end{aligned}
$$

instance *ComonadZip LV* **where**

$$
czip\ (xz := x)\ (yz := y) = zipL\ xz\ yz := (x, y)
$$

Kontekstide ühendamine LVS komonaadis

instance *ComonadZip LVS* **where**

$$
\begin{aligned}
&czip\ (xz := x :|\ xs)\ (yz := y :|\ ys) \\
&\qquad = zipL\ xz\ yz := (x, y) :|\ czip\ xs\ ys
\end{aligned}
$$

Komonaadilised interpretaatorid

Lambda-abstraktsiooni väärtustamine

$$
\begin{aligned}
&\textbf{class}\ ComonadZip\ d \Rightarrow ComonadEv\ d\ \textbf{where}\\
&\qquad ev :: Term \to d\ (Env\ d) \to Val\ d\\
&\_ev :: ComonadEv\ d \Rightarrow Term \to d\ (Env\ d) \to Val\ d\\
&\ldots\\
&\_ev\ (L\ x\ e)\ denv = F\ (\lambda d \to ev\ e\ (cmap\ repair\ (czip\ d\ denv)))\\
&\qquad\qquad\qquad\qquad \textbf{where}\ repair\ (a, env) = update\ x\ a\ env\\
&\ldots
\end{aligned}
$$

Komonaadilised interpretaatorid

Algkeskkond

```
env0 = [ ("+",    mkArithOp2 (+))
       , ...
       , ("not", mkBoolOp1 ¬)
       ]
  where mkArithOp2 op  = F $ λdi1 → let I i1 = counit di1
                             in F $ λdi2 → let I i2 = counit di2
                             in I (i1 `op` i2)
        mkCompOp2 op = F $ λdi1 → let I i1 = counit di1
                             in F $ λdi2 → let I i2 = counit di2
                             in B (i1 `op` i2)
        mkBoolOp2 op   = F $ λdb1 → let B b1 = counit db1
                             in F $ λdb2 → let B b2 = counit db2
                             in B (b1 `op` b2)
        mkBoolOp1 op   = F $ λdb → let B b = counit db
                             in B (op b)
```

Komonaadilised interpretaatorid

Standardinterpretaator

```
instance MonadEv Id where
        ev e env = _ev e env
evalI   :: Term → Val Id
evalI e = ev e (Id env0)
```

Näide

```
Main> evalI $ fact :@ N 5
120
```

Komonaadilised interpretaatorid

Süntaks: sünkroonsed andmevookeeled

```
data Term = ...
          | Term `Fby` Term
```

Kausaalsete striimifunktsioonidega interpretaator

```
instance ComonadEv LV where
  ev (e0 `Fby` e1) denv = ev e0 denv
                            `fbyLV` cobind (ev e1) denv
  ev e             denv = _ev e denv
```

Komonaadilised interpretaatorid

Kausaalsete striimifunktsioonidega interpretaator

$$
\begin{aligned}
&denv0L &&:: ComonadEv\ d \Rightarrow Int \to List\ (Env\ d)\\
&denv0L\ 0 &&= Nil\\
&denv0L\ n &&= denv0L\ (n-1) :> env0\\
&evalLV &&:: Term \to Int \to Val\ LV\\
&evalLV\ e\ i &&= ev\ e\ (denv0L\ i := env0)\\
&denv0S &&:: ComonadEv\ d \Rightarrow Stream\ (Env\ d)\\
&denv0S &&= env0 :< denv0S\\
&evRunLV &&:: Term \to Stream\ (Val\ LV)\\
&evRunLV\ e &&= runLV\ (ev\ e)\ denv0S
\end{aligned}
$$

Komonaadilised interpretaatorid

Näited

$$
\begin{aligned}
&-- \; pos = 0 \; `fby` \; (pos + 1) \\
&pos \quad = Rec \; (L \; \texttt{"pos"} \; (N \; 0 \; `Fby` \; (V \; \texttt{"pos"} \; \#+ \; N \; 1))) \\
&-- \; summ \; x = x + (0 \; `fby` \; summ \; x) \\
&summ = L \; \texttt{"x"} \; (Rec \; (L \; \texttt{"sumx"} \\
&\qquad\qquad\qquad\qquad (V \; \texttt{"x"} \; \#+ \; (N \; 0 \; `Fby` \; V \; \texttt{"sumx"})))) \\
&-- \; diff \; x = x - (0 \; `fby` \; diff \; x) \\
&diff \quad = L \; \texttt{"x"} \; (V \; \texttt{"x"} \; \#- \; (N \; 0 \; `Fby` \; V \; \texttt{"x"})) \\
&-- \; ini \; x = x \; `fby` \; ini \; x \\
&ini \quad = L \; \texttt{"x"} \; (Rec \; (L \; \texttt{"inix"} \; (V \; \texttt{"x"} \; `Fby` \; V \; \texttt{"inix"})))
\end{aligned}
$$

Komonaadilised interpretaatorid

Näited

$$
\begin{aligned}
-- \mathit{fakt} &= 1\ `\mathit{fby}`\ (\mathit{fact} * (\mathit{pos} + 1)) \\
\mathit{fakt} &= \mathit{Rec}\ (L\ \texttt{"fakt"}\ (N\ 1\ `\mathit{Fby}`\ (V\ \texttt{"fakt"}\ \#*\ (\mathit{pos}\ \#+\ N\ 1)))) \\
-- \mathit{fibo} &= 0\ `\mathit{fby}`\ (\mathit{fibo} + (1\ `\mathit{fby}`\ \mathit{fibo})) \\
\mathit{fibo} &= \mathit{Rec}\ (L\ \texttt{"fibo"}\ (N\ 0\ `\mathit{Fby}`\ (V\ \texttt{"fibo"}\ \#+\ (N\ 1\ `\mathit{Fby}`\ V\ \texttt{"fibo"}))))
\end{aligned}
$$

Komonaadilised interpretaatorid

Näitetestid

```
Main> evalLV pos 10
10
Main> evRunLV pos
0 :< (1 :< (2 :< (3 :< (4 :< (5 :< (6 :< (7 :< (8 :< ({Interrupted!}
Main> evRunLV $ summ :@ pos
0 :< (1 :< (3 :< (6 :< (10 :< (15 :< (21 :< (28 :< ({Interrupted!}
Main> evRunLV $ fakt
1 :< (1 :< (2 :< (6 :< (24 :< (120 :< (720 :< (5040 :< ({Interrupted!}
Main> evRunLV $ fibo
0 :< (1 :< (1 :< (2 :< (3 :< (5 :< (8 :< (13 :< (21 :< ({Interrupted!}
```

Komonaadilised interpretaatorid

Süntaks: üldised andmevookeeled

```
data Term = ...
          | Term `Fby` Term
          | Next Term
```

Üldiste striimifunktsioonidega interpretaator

```
instance ComonadEv FunArg where
  ev (e0 `Fby` e1) denv = ev e0 denv
                            `fbyFA` cobind (ev e1) denv
  ev (Next e)      denv = nextFA (cobind (ev e) denv)
  ev e             denv = _ev e denv
```

Komonaadilised interpretaatorid

Näide: filtreerimine

$$
\begin{aligned}
&-- \; x \; `wvr` \; y = \textbf{if} \; y \; \textbf{then} \; x \; `fby` \; (next \; x \; `wvr` \; next \; y) \\
&-- \qquad\qquad\qquad\qquad \textbf{else} \; (next \; x \; `wvr` \; next \; y) \\
&wvr = Rec \; (L \; \texttt{"wvr"} \; (L \; \texttt{"x"} \; (L \; \texttt{"y"} \; (\\
&\qquad\qquad If \; (V \; \texttt{"y"}) \\
&\qquad\qquad\quad (V \; \texttt{"x"} \; `Fby` \; (V \; \texttt{"wvr"} \; :@ \; (Next \; (V \; \texttt{"x"})) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad :@ \; (Next \; (V \; \texttt{"y"})))) \\
&\qquad\qquad\quad (V \; \texttt{"wvr"} \; :@ \; (Next \; (V \; \texttt{"x"})) \\
&\qquad\qquad\qquad\qquad\qquad :@ \; (Next \; (V \; \texttt{"y"})))))))
\end{aligned}
$$

Komonaadilised interpretaatorid

Näide: algarvud

$$
\begin{aligned}
&-- \mathit{sieve}\ x = x\ `\mathit{fby}`\ \mathit{sieve}\ (x\ `\mathit{wvr}`\ (x\ `\mathit{mod}`\ \mathit{ini}\ x \not\equiv 0))\\
&\mathit{sieve} = \mathit{Rec}\ (L\ \texttt{"sieve"}\ (L\ \texttt{"x"}\ (\\
&\qquad\qquad (V\ \texttt{"x"})\ `\mathit{Fby}`\\
&\qquad\qquad (V\ \texttt{"sieve"}\ :@\\
&\qquad\qquad\quad (\mathit{wvr}\ :@\ V\ \texttt{"x"}\ :@\\
&\qquad\qquad\qquad (V\ \texttt{"x"}\ \#\%\ (\mathit{ini}\ :@\ (V\ \texttt{"x"}))\ \#/{=}\ N\ 0))))))\\
&\mathit{eratosthenes} = \mathit{sieve}\ :@\ (\mathit{pos}\ \#{+}\ N\ 2)
\end{aligned}
$$