Monaadid

Sissejuhatus

- Monaadid on abstraktsioon kõrvalefektidega arvutuste esitamiseks ”puhtas maailmas”.
- Pärinevad kategooriateooriast.
- Moggi (1988) kasutas monaade denotatsioonsemantika struktureerimiseks.
- Wadler (1990) näitas, et analoogselt on võimalik monaade kasutada efektidega funktsionaalprogrammide strutureerimiseks.
- Haskell 1.3 (1996) võttis monaadid kasutusele IO struktureerimiseks.
- ...

Kategooriateooria põhidefinitsioonid

Kategooriad

Kategooria $\mathcal{C}$ koosneb:

- kollektsioonist objektidest $\mathrm{Obj}(\mathcal{C})$;
- iga $A, B \in \mathrm{Obj}(\mathcal{C})$ jaoks, kollektsoonist morfismidest (ehk nooltest) $\mathrm{Mor}(A, B)$;
  - morfismi $f \in \mathrm{Mor}(A, B)$ tähistatakse $f : A \to B$
- iga $A \in \mathrm{Obj}(\mathcal{C})$, identsusmorfism $id_A : A \to A$;
- iga $f : A \to B$ ja $g : B \to C$ korral leidub morfism $g \cdot f : A \to C$ (f ja g kompositsioon);
- identsused ja kompositsioon rahuldavad võrdusi:

$$f = f \cdot id_A \quad f = id_B \cdot f \quad h \cdot (g \cdot f) = (h \cdot g) \cdot f$$

kus $f : A \to B$, $g : B \to C$ ja $h : C \to D$.

Kategooriateooria põhidefinitsioonid

Näited

Hulkade kategooria $\mathcal{S}et$:

- objektid on hulgad;
- morfismid on kujutised hulkade vahel.

Osaliste funktsioonide kategooria $\mathcal{P}fn$:

- objektid on hulgad;
- morfismid on osalised kujutised hulkade vahel.

Osaliste funktsioonide kategooria $\mathcal{R}el$:

- objektid on hulgad;
- morfismid on relatsioonid hulkade vahel.

Osaliste funktsioonide kategooria $\mathcal{M}on$:

- objektid on monoidid;
- morfismid on monoidide homomorfismid.

Kategooriateooria põhidefinitsioonid

Funktorid

Olgu $\mathcal{C}$, $\mathcal{D}$ kategooriad. Funktor $\mathsf{F} : \mathcal{C} \to \mathcal{D}$ koosneb kahest kujutisest:

- kujutis objektidel $\mathsf{F} : \mathrm{Obj}(\mathcal{A}) \to \mathrm{Obj}(\mathcal{B})$, ja
- kujutis morfismidel:

$$\begin{aligned} \mathsf{F}(f : A \to B) &: \mathsf{F}A \to \mathsf{F}B \\ \mathsf{F}(id_A) &= id_{\mathsf{F}A} \\ \mathsf{F}(g \cdot f) &= \mathsf{F}(g) \cdot \mathsf{F}(f) \end{aligned}$$

Funktorit $\mathsf{F} : \mathcal{C} \to \mathcal{C}$ (so. mille lähte- ning sihtkategooria on samad) nimetatakse endofunktoriks.

Kategooriateooria põhidefinitsioonid

Näited

- Identsus- ja konstantne funktor:

$$\begin{array}{rcl} \mathsf{Id} : \mathcal{C} \to \mathcal{C} \\ X & \mapsto & X \\ f & \mapsto & f \end{array} \qquad \begin{array}{rcl} \mathsf{K}_A : \mathcal{C} \to \mathcal{D} \\ X & \mapsto & A \\ f & \mapsto & id_A \end{array}$$

- Unustav- ja "powerset"-funktor:

$$\begin{array}{rcl} \mathsf{U} : \mathcal{M}on \to \mathcal{S}et \\ X & \mapsto & X \\ f & \mapsto & f \end{array} \qquad \begin{array}{rcl} \mathsf{P} : \mathcal{S}et \to \mathcal{S}et \\ \mathsf{P}(X) & = & \{Y \mid Y \subseteq X\} \\ \mathsf{P}(f) & = & X \mapsto \{\, f\,x \mid x \in X \,\} \end{array}$$

- Listide funktor $\mathsf{List} : \mathcal{S}et \to \mathcal{M}on$ (aga ka $\mathsf{List} : \mathcal{S}et \to \mathcal{S}et$)

$$\begin{array}{rcl} \mathsf{List}(X) & = & X^* \\ \mathsf{List}(f) & = & [x_1, \ldots, x_n] \mapsto [f\,x_1, \ldots, f\,x_n] \end{array}$$

Kategooriateooria põhidefinitsioonid

Loomulikud teisendused

Olgu $\mathsf{F}, \mathsf{G} : \mathcal{C} \to \mathcal{D}$ funktorid. Loomulik teisendus $\tau : \mathsf{F} \to \mathsf{G}$ on morfismide pere $\tau_X : \mathsf{F}X \to \mathsf{G}X$ selline, et iga morfismi $f : X \to Y$ korral, järmine ruut kommuteerub:

$$\begin{array}{ccc} \mathsf{F}X & \xrightarrow{\tau_X} & \mathsf{G}X \\ {\scriptstyle \mathsf{F}f}\downarrow & & \downarrow{\scriptstyle \mathsf{G}f} \\ \mathsf{F}Y & \xrightarrow[\tau_Y]{} & \mathsf{G}Y \end{array}$$

Kategooriateooria põhidefinitsioonid

Näited

- Listide pööramine:

$$\begin{aligned} &\mathit{reverse} : \mathsf{List} \to \mathsf{List} \\ &\mathit{reverse}\,[x_1, \ldots, x_n] \;=\; [x_n, \ldots, x_1] \end{aligned}$$

$$\mathit{reverse} \cdot \mathsf{List}(f) \;=\; \mathsf{List}(f) \cdot \mathit{reverse}$$

- Listi pikkuse leidmine:

$$\begin{aligned} &\mathit{length} : \mathsf{List} \to \mathsf{K_N} \\ &\mathit{length}\,[x_1, \ldots, x_n] \;=\; n \end{aligned}$$

$$\begin{aligned} \mathit{length} \cdot \mathsf{List}(f) &= \mathsf{K_N}(f) \cdot \mathit{length} \\ &= \mathit{length} \end{aligned}$$

Kategooriateooria põhidefinitsioonid

Kategooriad ja Haskell

Haskelli võime mõelda kategooriana, kus:

- objektideks on (monomorfsed) tüübid;
- morfismideks on (monomorfsed) funktsioonid;
- (endo-)funktoriteks on tüübikonstruktorid (ehk polümorfsed tüübid);
- loomulikeks teisendusteks on (parameetriliselt) polümorfsed funktsioonid.

NB!

Tüübikonstruktorid teevad tüüpidest tüüpe, ehk siis on teisendus objektide vahel. Funktor peab teisendama ka morfismid morfismideks.

Kategooriad ja Haskell

Klass *Functor*

$$\textbf{class } \mathit{Functor}\ f \textbf{ where}$$
$$\quad \mathit{fmap} :: (a \to b) \to f\ a \to f\ b$$

NB!

- Vastavalt funktori definitsioonile, peab *fmap* lisaks veel säilitama identsust ja kompositsiooni.
- Seda Haskelli kompilaator ei kontrolli, ning see on puhtalt programmeerija vastutusel, et *fmap* tõepoolest seda teeb.

Kategooriad ja Haskell

Listide funktor

```
instance Functor [] where
    fmap = map
```

Maybe funktor

```
data Maybe a = Nothing | Just a

instance Functor Maybe where
    fmap f Nothing = Nothing
    fmap f (Just x) = Just (f x)
```

Aritmeetilste avaldiste väärtustamine

Aritmeetilste avaldised

$$
\begin{aligned}
\textbf{data}\ \mathit{Expr} = {} & \mathit{Num}\ \mathit{Int} \\
& \mid \mathit{Expr}\ \mathrel{:+:}\ \mathit{Expr} \\
& \mid \mathit{Expr}\ \mathrel{:-:}\ \mathit{Expr} \\
& \mid \mathit{Expr}\ \mathrel{:*:}\ \mathit{Expr} \\
& \mid \mathit{Expr}\ \mathrel{:/:}\ \mathit{Expr}
\end{aligned}
$$

Näited

$$
\begin{aligned}
\mathit{exp1} &= \mathit{Num}\ 1\ \mathrel{:+:}\ \mathit{Num}\ 2 \\
\mathit{exp2} &= \mathit{Num}\ 2\ \mathrel{:*:}\ (\mathit{Num}\ 4\ \mathrel{:-:}\ \mathit{Num}\ 1) \\
\mathit{exp3} &= \mathit{Num}\ 4\ \mathrel{:*:}\ \mathit{Num}\ 3\ \mathrel{:/:}\ \mathit{Num}\ 2
\end{aligned}
$$

Aritmeetilste avaldiste väärtustamine

Lihtne väärtustaja

$$
\begin{aligned}
&eval :: Expr \to Int \\
&eval\ (Num\ i) \quad\ = i \\
&eval\ (e1\ :+:\ e2) = (eval\ e1) + (eval\ e2) \\
&eval\ (e1\ :-:\ e2) = (eval\ e1) - (eval\ e2) \\
&eval\ (e1\ :*:\ e2) = (eval\ e1) * (eval\ e2) \\
&eval\ (e1\ :/:\ e2) = (eval\ e1)\ `div`\ (eval\ e2)
\end{aligned}
$$

Näited

```
Main> eval exp2
6
Main> eval (Num 3 :/: Num 0)
Program error: divide by zero
```

Aritmeetilste avaldiste väärtustamine

Veatöötlusega väärtustaja

```
eval :: Expr → Maybe Int
eval (Num i)    = Just i
eval (e1 :+: e2) = case eval e1 of
                     Nothing → Nothing
                     Just v1  → case eval e2 of
                       Nothing → Nothing
                       Just v2  → Just (v1 + v2)
eval (e1 :-: e2) = ...
eval (e1 :*: e2) = ...
```

Aritmeetilste avaldiste väärtustamine

Veatöötlusega väärtustaja (järg)

$$
\begin{aligned}
\mathit{eval}\ (\mathit{e1}\ \mathit{:/:}\ \mathit{e2}) = \mathbf{case}\ \mathit{eval}\ \mathit{e1}\ \mathbf{of}& \\
\mathit{Nothing} &\rightarrow \mathit{Nothing} \\
\mathit{Just}\ \mathit{v1} &\rightarrow \mathbf{case}\ \mathit{eval}\ \mathit{e2}\ \mathbf{of} \\
\mathit{Nothing} &\rightarrow \mathit{Nothing} \\
\mathit{Just}\ \mathit{v2} &\rightarrow \mathbf{if}\ \mathit{v2} \equiv 0 \\
& \quad\ \mathbf{then}\ \mathit{Nothing} \\
& \quad\ \mathbf{else}\ \ \mathit{Just}\ (\mathit{v1}\ \text{`}\mathit{div}\text{`}\ \mathit{v2})
\end{aligned}
$$

Näited

```
Main> eval exp2
Just 6
Main> eval (Num 3 :/:  Num 0)
Nothing
```

Aritmeetilste avaldiste väärtustamine

NB!

- Suur hulk koodi duplitseerimist!
- Mis on ühtne muster mida välja faktoriseerida?
- Paneme tähele, et:
  - toimub alamavaldiste väärtustamiste järjestikustamine;
  - kui üks ebaõnnestub, siis ebaõnnestub ka kogu avaldise väärtustamine;
  - väärtustamise õnnestumisel tehakse resultaat kättesaadavaks järgnevatele väärtustustele.

Aritmeetilste avaldiste väärtustamine

Väärtustuste järjestikustamine

$$
\begin{aligned}
&evSeq :: Maybe\ Int \to (Int \to Maybe\ Int) \to Maybe\ Int \\
&evSeq\ ma\ f = \mathbf{case}\ ma\ \mathbf{of} \\
&\qquad\qquad\qquad Nothing \to Nothing \\
&\qquad\qquad\qquad Just\ a \quad \to f\ a
\end{aligned}
$$

Aritmeetilste avaldiste väärtustamine

Veatöötlusega väärtustaja

```
eval :: Expr → Maybe Int
eval (Num i)    = Just i
eval (e1 :+: e2) = eval e1 `evSeq` λv1 →
                   eval e2 `evSeq` λv2 →
                   Just (v1 + v2)
eval (e1 :-: e2) = ...
eval (e1 :*: e2) = ...
eval (e1 :/: e2) = eval e1 `evSeq` λv1 →
                   eval e2 `evSeq` λv2 →
                   if v2 ≡ 0
                      then Nothing
                      else Just (v1 `div` v2)
```

Ebaõnnestumistega arvutused

Üldistus

- Tüübist *Maybe a* võime mõelda kui tüüpi a väärtusi väljastavatest arvutustest, mis võivad ka ebaõnnestuda; so.
  - arvutus võib õnnestuda, ning väljastab tulemusena tüüpi a väärtuse;
  - arvutus võib ebaõnnestuda, ning tulemust ei väljastata;
  - arvutusi saab järjestikku komponeerida, so. üksteise järel täita.
- Järjestikkompositsiooni korral ühe alamarvutuse ebaõnnestumine põhjustab kogu arvutuse ebaõnnestumise.
- Seega, ebaõnnestumine on efekt, mõjutades kaudselt kõiki järgnevaid arvutusi.

Ebaõnnestumistega arvutused

Ebaõnnestumistega arvutuste monaad

$$
\begin{aligned}
&mReturn &&:: a \to Maybe\ a \\
&mReturn\ a &&= Just\ a \\
&mFail &&:: Maybe\ a \\
&mFail &&= Nothing \\
&mBind &&:: Maybe\ a \to (a \to Maybe\ b) \to Maybe\ b \\
&mBind\ ma\ f &&= \mathbf{case}\ ma\ \mathbf{of} \\
&&&\qquad Nothing \to Nothing \\
&&&\qquad Just\ a \to f\ a
\end{aligned}
$$

Aritmeetilste avaldiste väärtustamine

Veatöötlusega väärtustaja

$$
\begin{aligned}
&eval :: Expr \to Maybe\ Int \\
&eval\ (Num\ i) \quad = mReturn\ i \\
&eval\ (e1\ :+:\ e2) = eval\ e1\ `mBind`\ \lambda v1 \to \\
&\qquad\qquad\qquad\quad eval\ e2\ `mBind`\ \lambda v2 \to \\
&\qquad\qquad\qquad\quad mReturn\ (v1 + v2) \\
&eval\ (e1\ :-:\ e2) = \ \ldots \\
&eval\ (e1\ :*:\ e2) = \ \ldots \\
&eval\ (e1\ :/:\ e2) = eval\ e1\ `mBind`\ \lambda v1 \to \\
&\qquad\qquad\qquad\quad eval\ e2\ `mBind`\ \lambda v2 \to \\
&\qquad\qquad\qquad\quad \textbf{if}\ v2 \equiv 0 \\
&\qquad\qquad\qquad\qquad\ \ \textbf{then}\ mFail \\
&\qquad\qquad\qquad\qquad\ \ \textbf{else}\ \ mReturn\ (v1\ `div`\ v2)
\end{aligned}
$$

Puu lehtede nummerdamine

Binaarsed "lehtpuud"

$$\begin{aligned}\textbf{data}\ \mathit{Tree}\ a = {} & \mathit{Leaf}\ a \\ & \mid \mathit{Branch}\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)\end{aligned}$$

Näited

$$\mathit{tree1} = \mathit{Branch}\ (\mathit{Leaf}\ 7)\ (\mathit{Leaf}\ 3)$$

$$\begin{aligned}\mathit{tree2} = \mathit{Branch}\ & (\mathit{Branch}\ (\mathit{Leaf}\ \texttt{'A'}) \\ & \qquad (\mathit{Branch}\ (\mathit{Leaf}\ \texttt{'B'}) \\ & \qquad\qquad (\mathit{Leaf}\ \texttt{'C'}))) \\ & \quad (\mathit{Branch}\ (\mathit{Leaf}\ \texttt{'D'}) \\ & \qquad (\mathit{Leaf}\ \texttt{'E'}))\end{aligned}$$

Puu lehtede nummerdamine

Lehtede nummerdamine

$$
\begin{aligned}
&labelTree \;\; :: \; Tree\ a \to Tree\ Int \\
&labelTree\ t = fst\ (labAux\ t\ 0) \\
&labAux :: Tree\ a \to Int \to (Tree\ Int, Int) \\
&labAux\ (Leaf\ x) \qquad\qquad\quad c = (Leaf\ c, c+1) \\
&labAux\ (Branch\ t1\ t2)\ c = \mathbf{let}\ (t1', c1) = labAux\ t1\ c \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad (t2', c2) = labAux\ t2\ c1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{in}\ (Branch\ t1'\ t2', c2)
\end{aligned}
$$

Puu lehtede nummerdamine

NB!

- Loendur algul intsialiseeritakse ning lehtedes kasutatakse tema hetkeväärtust.
- Alampuude korrektseks nummerdamiseks tuleb loendurit vastavates arvutustes kaasa tassida ning erinevate alamarvutuste vahel korrektses järjekorras edastada.
  - On väga vigade altis, kuna väga lihtne on kogemata edastada loendurist vale versioon.
  - Imperatiivkeeltes võiksime loendurina kasutada globaalset muutujat, mille edastamine alamarvutustele toimub varjatult.

Olekust sõltuvad arvutused

Üldistus

- Olekust sõltuv arvutus kasutab väärtuse arvutamiseks mingit olekut ning arvutuse käigus võib muuta ka olekut.
- Olekust sõltuvat arvutust saame esitada olekuteisendajana, so. funktsioonina mis argumendina saab sisendoleku ning väljastab resultaadi koos väljundolekuga.
  - Arvutus võib mitte vajada olekut, misjuhul väljundolek on sama mis sisendolek ning resultaat on olekust sõltumatu.
  - Olekust sõltuvate arvutuste järjestikku komponeerimisel tuleb eelneva alamarvutuse väljundolek anda järgneva arvutuse sisendolekuks.
  - Ainult olekut manipuleerivad olekuteisendajad on efektid.

Olekust sõltuvad arvutused

Olekuteisndajad

$$
\begin{aligned}
&\textbf{type } IntSt\ a = Int \to (a, Int) \\
&sReturn \quad :: a \to IntSt\ a \\
&sReturn\ x \quad = \lambda s \to (x, s) \\
&inc \quad :: IntSt\ Int \\
&inc \quad = \lambda s \to (s, s+1) \\
&sBind \quad :: IntSt\ a \to (a \to IntSt\ b) \to IntSt\ b \\
&m\ `sBind`\ f \quad = \lambda s \to \textbf{let } (x1, s1) = m\ s \\
&\qquad\qquad\qquad\qquad\quad\ (x2, s2) = f\ x1\ s1 \\
&\qquad\qquad\qquad\qquad \textbf{in } (x2, s2)
\end{aligned}
$$

Puu lehtede nummerdamine

Lehtede nummerdamine

$$
\begin{aligned}
&labelTree \;\; :: \; Tree\ a \to Tree\ Int \\
&labelTree\ t = fst\ (labAux\ t\ 0) \\
&labAux \;\; :: \; Tree\ a \to IntSt\ (Tree\ Int) \\
&labAux\ (Leaf\ x) \;\; = inc\ `sBind`\ \lambda i \to \\
&\qquad\qquad\qquad\qquad sReturn\ (Leaf\ i) \\
&labAux\ (Branch\ t1\ t2) = labAux\ t1\ `sBind`\ \lambda t1' \to \\
&\qquad\qquad\qquad\qquad labAux\ t2\ `sBind`\ \lambda t2' \to \\
&\qquad\qquad\qquad\qquad sReturn\ (Branch\ t1'\ t2')
\end{aligned}
$$

Monaadid

Võrdlus

- Mõlema näite korral on põhiomaduseks efektidega arvutuste teostamine.
- Mõlemal juhul sai efektidega arvutusi selgelt esitada tuues sisse identse struktuuriga järgmised abstraktsioonid:
  - tüüp, mis esitab arvutusi;
  - kombinaator, mis esitab ilma efektideta väärtusi;
  - kombinaator arvutuste järjestikku komponeerimseks.
- Vastava struktuuriga arvutused ongi monaadid.

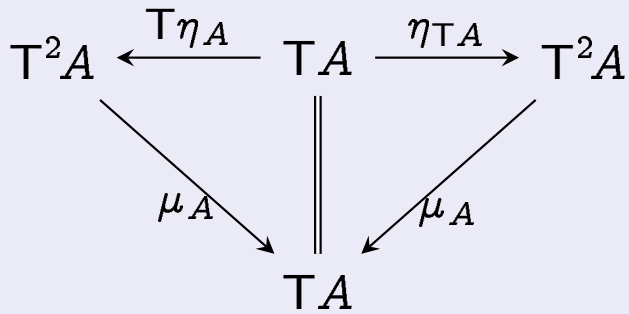
Monaadid

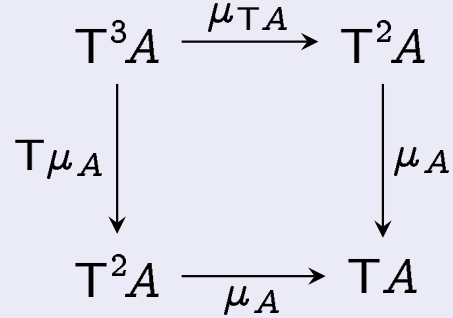
Definitsioon

Monaad kategoorial $\mathcal{C}$ koosneb:

- endofunktorist $\mathsf{T} : \mathcal{C} \to \mathcal{C}$
- loomulikust teisendusest $\eta_A : A \to \mathsf{T}A$
  - nn. monaadi ühik
- loomulikust teisendusest $\mu_A : \mathsf{T}\mathsf{T}A \to \mathsf{T}A$
  - nn. monaadi multiplikatsioon

selliselt, et järgnevad diagrammid kommuteeruvad:





Monaadid

Definitsioon

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

- kujutisest $\mathsf{T} : \mathrm{Obj}(\mathcal{C}) \to \mathrm{Obj}(\mathcal{C})$
- $\mathrm{Obj}(\mathcal{C})$-indekseeritud kujutiste perest $\eta_A : A \to \mathsf{T}A$
- operatsioonist $-^\star$, mis igale morfismile $f : A \to \mathsf{T}B$ seab vastavusse morfismi $f^\star : \mathsf{T}A \to \mathsf{T}B$
  - nn. monaadi ekstensioon

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

$$\begin{aligned} {\eta_A}^\star &= id_{TA} \\ f^\star \cdot \eta_A &= f \\ (g^\star \cdot f)^\star &= g^\star \cdot f^\star \end{aligned}$$

Monaadid

Lemma

Monaadid ja Kleisli kolmikud on ekvivalentsed mõisted.

Tõestus

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

- $(\mathsf{T}, \eta, \mu) \Longrightarrow (\mathsf{T}, \eta, -^\star)$

$$f^\star = \mu_B \cdot \mathsf{T}f$$

- $(\mathsf{T}, \eta, -^\star) \Longrightarrow (\mathsf{T}, \eta, \mu)$

$$\begin{aligned} \mathsf{T}g &= (\eta_B \cdot g)^\star \\ \mu_A &= id_{\mathsf{T}A}{}^\star \end{aligned}$$

Monaadid Haskellis

Monaadide klass

$$
\begin{aligned}
&\textbf{class } \mathit{Monad}\ m \textbf{ where} \\
&\quad \mathit{return} :: a \to m\ a \\
&\quad (\gg\!=) \quad :: m\ a \to (a \to m\ b) \to m\ b \\
&\quad (\gg) \quad :: m\ a \to m\ b \to m\ b \\
&\quad \mathit{fail} \quad :: \mathit{String} \to m\ a \\
&\quad m \gg k = m \gg\!= \lambda\_ \to k \\
&\quad \mathit{fail}\ s \quad = \mathit{error}\ s
\end{aligned}
$$

Monaadid Haskellis

Monaadide seadused

$$
\begin{aligned}
return\ x >\!\!>\!= f &= f\ x \\
m >\!\!>\!= return &= m \\
(m >\!\!>\!= \lambda x \rightarrow k) >\!\!>\!= f &= m >\!\!>\!= \lambda x \rightarrow (k >\!\!>\!= f)
\end{aligned}
$$

NB!

- Viimases võrduses on eeldatud, et muutuja x ei sisaldu vabalt avaldises f.
- Iga monaad peaks antud võrdusi rahuldama, aga Haskelli kompilaator võrduste kehtivust ei kontrolli. See on programmeerija ülesanne, et tema defineeritud monaad neid võrdusi tõepoolest rahuldab!

Monaadid Haskellis

do-süntaks

$$
\begin{array}{lcl}
\mathit{expr} & = & \mathbf{do}\ \{\mathit{stmt}; \ldots; \mathit{stmt}\} \\
\mathit{stmt} & = & \mathit{pat} \leftarrow \mathit{expr} \\
 & | & \mathit{expr} \\
 & | & \mathbf{let}\ \mathit{decls}
\end{array}
$$

Transleerimisreeglid

$$
\begin{array}{ll}
\mathbf{do}\,\{e\} & = e \\
\mathbf{do}\,\{e; \mathit{stmts}\} & = e \gg \mathbf{do}\,\{\mathit{stmts}\} \\
\mathbf{do}\,\{v \leftarrow e; \mathit{stmts}\} & = e \gg\!= \lambda v \rightarrow \mathbf{do}\,\{\mathit{stmts}\} \\
\mathbf{do}\,\{\mathbf{let}\ \mathit{decls}; \mathit{stmts}\} & = \mathbf{let}\ \mathit{decls}\ \mathbf{in}\ \mathbf{do}\,\{\mathit{stmts}\}
\end{array}
$$

Monaadid Haskellis

Identsusmonaad

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

$$\textbf{instance } Monad\ Id \textbf{ where}$$
$$(Id\ x) >\!\!>\!= k = k\ x$$
$$return\ x \quad = Id\ x$$

NB!

Identsusmonaad esitab ”puhtaid” (so. ilma efektideta) arvutusi.

Monaadid Haskellis

Maybe monaad

$$
\begin{aligned}
&\textbf{instance } \mathit{Monad}\ \mathit{Maybe} \textbf{ where} \\
&\quad \mathit{Nothing} \gg\!= k = \mathit{Nothing} \\
&\quad (\mathit{Just}\ x) \gg\!= k = k\ x \\
&\quad \mathit{return}\ x \quad\ = \mathit{Just}\ x \\
&\quad \mathit{fail}\ s \quad\ = \mathit{Nothing}
\end{aligned}
$$

Monaadid Haskellis

Olekuteisendusmonaad

$$\begin{aligned}
&\textbf{newtype } S\ s\ a = S\ (s \to (a, s)) \\
&\textbf{instance } \mathit{Monad}\ (S\ s)\ \textbf{where} \\
&\quad S\ m \gg\!= k = S\ \$\ \lambda s \to \\
&\qquad\qquad\qquad \textbf{case } m\ s\ \textbf{of} \\
&\qquad\qquad\qquad\quad (x, s') \to \textbf{case } k\ x\ \textbf{of} \\
&\qquad\qquad\qquad\qquad\qquad\quad S\ g \to g\ s' \\
&\quad \mathit{return}\ x \quad = S\ (\lambda s \to (x, s))
\end{aligned}$$

Monaadid Haskellis

Monaadispetsiifilised operatsioonid

- Monaadi baasoperatsioonid *return* ja (>>=) võimaldavad tekitada puhtaid arvutusi ning arvutusi järjestikku komponeerida.
- Et monaad omaks mõtet, peab ta lisaks veel omama lisaoperatsioone, mis on spetsiifilised monaadi poolt esitatavatele efektidele.
- Reeglina on kahesuguseid monaadispetsiifilisi operatsioone:
  - operatsioonid konkreetse efekti tekitamiseks;
  - operatsioonid arvutustulemuse väliseks inspekteerimiseks.

Monaadid Haskellis

Maybe monaadi spetsiifilised operatsioonid

$$
\begin{aligned}
&raise &&:: Maybe\ a \\
&raise &&= Nothing \\
&handle &&:: Maybe\ a \to Maybe\ a \to Maybe\ a \\
&handle\ m\ h &&= \mathbf{case}\ m\ \mathbf{of} \\
&&&\quad Nothing \to h \\
&&&\quad Just\ \_ \to m
\end{aligned}
$$

Monaadid Haskellis

Olekuteisendusmonaadi spetsiifilised operatsioonid

$$
\begin{aligned}
&getS &&:: S\ s\ s\\
&getS &&= S\ (\lambda s \to (s, s))\\
&setS &&:: s \to S\ s\ ()\\
&setS\ x &&= S\ (\lambda s \to ((), x))\\
&runS &&:: S\ s\ a \to s \to (a, s)\\
&runS\ (S\ f)\ s &&= f\ s
\end{aligned}
$$

Monaadid Haskellis

Listide monaad

$$
\begin{aligned}
&\textbf{instance } \mathit{Monad}\ [\,]\ \textbf{where}\\
&\qquad \mathit{xs} \gg\!= f \;\; = \mathit{concat}\ (\mathit{map}\ f\ \mathit{xs})\\
&\qquad \mathit{return}\ x = [x]\\
&\qquad \mathit{fail}\ s \;\;\;\; = [\,]\\
&\mathit{deadlock} :: [a]\\
&\mathit{deadlock} = [\,]\\
&\mathit{choice} :: [a] \to [a] \to [a]\\
&\mathit{xs}\ \text{`}\mathit{choice}\text{`}\ \mathit{ys} = \mathit{xs} +\!\!+ \mathit{ys}
\end{aligned}
$$

NB!

Listide monaad vastab mittedetermineeritud arvutustele.

Monaadid Haskellis

Keskkondade monaad

$$
\begin{aligned}
&\textbf{instance } \mathit{Monad}\ ((\rightarrow)\ e)\ \textbf{where}\\
&\qquad m \gg\!= f \;\; = \lambda e \rightarrow f\ (m\ e)\ e\\
&\qquad \mathit{return}\ x = \lambda e \rightarrow x\\
&\mathit{getEnv} \;\; :: \;\; ((\rightarrow)\ e)\ e\\
&\mathit{getEnv} \;\; = \lambda e \rightarrow e\\
&\mathit{withEnv} :: \;\; ((\rightarrow)\ e)\ a \rightarrow e \rightarrow a\\
&\mathit{withEnv}\ f\ \ e = f\ \ e
\end{aligned}
$$

NB!

$(\gg\!=)$, return, getEnv vs. S, K, I kombinaatorid.

Monaadid Haskellis

Klass *MonadPlus*

$$\begin{aligned}&\textbf{class } Monad\ m \Rightarrow MonadPlus\ m \textbf{ where}\\ &\quad mzero :: m\ a\\ &\quad mplus :: m\ a \to m\ a \to m\ a\end{aligned}$$

Nõutud võrdused

$$\begin{aligned}mzero\ `mplus`\ m &= m\\ m\ `mplus`\ mzero &= m\\ (m_1\ `mplus`\ m_2)\ `mplus`\ m_3 &\\ &= m_1\ `mplus`\ (m_2\ `mplus`\ m_3)\\ mzero >\!\!>\!= f &= mzero\\ m >\!\!> mzero &= mzero\end{aligned}$$

Monaadid Haskellis

Maybe monaad

```
instance MonadPlus Maybe where
  mzero                 = Nothing
  Nothing `mplus` m2    = m2
  m1      `mplus` m2    = m1
```

Listide monaad

```
instance MonadPlus [] where
  mzero = []
  mplus = (++)
```

Monaadid Haskellis

Parserite monaad

```
newtype Parser a = P (String → [(a, String)])
instance Monad Parser where
  return a  = P $ λcs → [(a, cs)]
  p >>= f   = P $ λcs → concat
                $ [runP (f a) cs' | (a, cs') ← runP p cs]
instance MonadPlus Parser where
  mzero     = P $ λcs → []
  mplus p q = P $ λcs → runP p cs ++ runP q cs
item :: Parser Char
item = P $ λcs → [(head cs, tail cs) | ¬ (null cs)]
runP :: Parser a → String → [(a, String)]
runP (P p) = p
```