Monaaditeisendajad

Monaadide kombineerimine

- Erinevaid monaade on võimalik kombineerida uute monaadide moodustamiseks.
- Kui monaadiline kood ei kasuta antud monaadi konkreetse realisatsiooni detaile, vaid ainult läbi liidese defineeritud operatsioone, siis võib konreetset realisatsiooni muuta ilma, et seda kasutav monaadiline kood muutuks.
- Sealhulgas, kui efektispetsiifilised operatsioonid on deklareeritud abstraktse liidesena, siis saab kirjutada konkreetsest realisatsioonist sõltumatud antud efekte produtseerivat koodi.

Monaaditeisendajad

Parserite monaad

```
newtype Parser s a = Parser{
  runParser :: [s] → [(a,[s])]
}
instance Monad (Parser s) where
  return x = Parser $ λss → [(x, ss)]
  p >>= f  = Parser $ λss → do
                      (x, ss') ← runParser p ss
                      runParser (f x) ss'
```

NB!

Kombineerib olekuteisendusmonaadi listide monaadiga.

Monaaditeisendajad

Listide monaad

$$
\begin{aligned}
&\textbf{instance } \mathit{Monad}\ [\,]\ \textbf{where}\\
&\quad (x : xs) \gg\!= f = f\ x +\!\!+ (xs \gg\!= f)\\
&\quad [\,] \qquad \gg\!= f = [\,]\\
&\quad \mathit{return}\ x \qquad = [x]\\
&\quad \mathit{fail}\ s \qquad = [\,]\\
&\textbf{instance } \mathit{MonadPlus}\ [\,]\ \textbf{where}\\
&\quad \mathit{mzero} \qquad = [\,]\\
&\quad \mathit{mplus} \qquad = (+\!\!+)
\end{aligned}
$$

Monaaditeisendajad

Olekuga monaadide liides

$$\begin{aligned}
&\textbf{class } Monad\ m \Rightarrow MonadState\ s\ m \mid m \to s \textbf{ where}\\
&\quad get :: m\ s\\
&\quad put :: s \to m\ ()\\
&modify :: MonadState\ s\ m \Rightarrow (s \to s) \to m\ ()\\
&modify\ f = get \gg\!= \lambda s \to put\ (f\ s)\\
&gets \quad :: MonadState\ s\ m \Rightarrow (s \to a) \to m\ a\\
&gets\ f \quad = get \gg\!= \lambda s \to return\ (f\ s)
\end{aligned}$$

NB!

Kasutab funktsionaalseid sõltuvusi!

Monaaditeisendajad

Olekuteisendusmonaad

```
newtype State s a = State{
  runState :: s → (a, s)
}
instance Monad (State s) where
  return a = State $ λs → (a, s)
  m >>= k  = State $ λs → let
                 (a, s') = runState m s
                 in runState (k a) s'
```

Olekuteisendusmonaadi operatsioonid

```
instance MonadState s (State s) where
  get   = State $ λs → (s, s)
  put s = State $ λ_ → ((), s)
```

Monaaditeisendajad

Parametriseeritud olekuteisendusmonaad

```
newtype StateT s m a = StateT{
  runStateT :: s → m (a, s)
}
instance Monad m ⇒ Monad (StateT s m) where
  return a = StateT $ λs → return (a, s)
  m >>= k  = StateT $ λs → do
               (a, s') ← runStateT m s
               runStateT (k a) s'
  fail str = StateT $ λ_ → fail str
instance Monad m ⇒ MonadState s (StateT s m) where
  get      = StateT $ λs → return (s, s)
  put s    = StateT $ λ_ → return ((), s)
```

NB!

Definitsioon kasutab ära parameetermonaadi operatsioone!

Monaaditeisendajad

NB!

- *StateT* lisab suvalisele monaadile olekuteisenduse funktsionaalsuse.
- Selleks, et parameetermonaadi enda originaalfunktsionaalsus oleks ka kombineeritud monaadis kasutatav, tuleb vastavad operatsioonid "kergitada".

Monaaditeisendajate klass

```
class MonadTrans t where
  lift :: Monad m ⇒ m a → t m a
```

Parametriseeritud olekuteisendusmonaadi instants

```
instance MonadTrans (StateT s) where
  lift m = StateT $ λs → do
                  a ← m
                  return (a, s)
```

Monaaditeisendajad

MonadPlus funktsionaalsuse "kergitamine"

```
instance MonadPlus m ⇒ MonadPlus (StateT s m) where
  mzero       = StateT $ λ_ → mzero
  m `mplus` n = StateT $ λs → runStateT m s
                              `mplus`
                              runStateT n s
```

NB!

Meetodi *mzero* oleksime võinud ka defineerida kui:

$$mzero = lift\ mzero$$

Monaaditeisendajad

NB!

Analoogiliselt saame defineerida monaaditeisendajaid teiste monaadide jaoks.

Parametriseeritud listide monaad

```
newtype ListT m a = ListT{
  runListT :: m [a]
}
instance Monad m ⇒ Monad (ListT m) where
  return a = ListT $ return [a]
  m >>= k  = ListT $ do
                a ← runListT m
                b ← mapM (runListT ∘ k) a
                return (concat b)
  fail _   = ListT $ return []
```

Monaaditeisendajad

Parametriseeritud listide monaad

```
instance Monad m ⇒ MonadPlus (ListT m) where
  mzero       = ListT $ return []
  m `mplus` n = ListT $ do
                  a ← runListT m
                  b ← runListT n
                  return (a ++ b)
instance MonadTrans ListT where
  lift m      = ListT $ do
                  a ← m
                  return [a]
```

MonadState funktsionaalsuse "kergitamine"

```
instance MonadState s m ⇒ MonadState s (ListT m) where
  get = lift get
  put = lift ∘ put
```

Monaaditeisendajad

NB!

- Monaadide kombineerimise järjekord on tähtis!
- Listidega olekuteisendusmonaad:

$$\begin{aligned} ListT\ (State\ s)\ a &\cong State\ s\ [a] \\ &\cong s \to ([a], s) \end{aligned}$$

- Olekutega listide monaad:

$$\begin{aligned} StateT\ s\ []\ a \qquad &\cong s \to []\ (a, s) \\ &\cong s \to [(a, s)] \end{aligned}$$

Monaaditeisendajad

Eeldefineeritud monaaditeisendajad

- Haskellis on eeldefineeritud terve hulk monaaditeisendajaid.
- Asuvad standardteekides *Control.Monad.**

Control.Monad.Identity	*Identity*		
Control.Monad.Error	*Either*	*ErrorT*	*MonadError*
Control.Monad.List	[]	*ListT*	*MonadPlus*
Control.Monad.State	*State*	*StateT*	*MonadState*
Control.Monad.Reader	*Reader*	*ReaderT*	*MonadReader*
Control.Monad.Writer	*Writer*	*WriterT*	*MonadWriter*
Control.Monad.Cont	*Cont*	*ContT*	*MonadCont*

Monaaditeisendajad: veatöötlusmonaad

Vigade klass

```
class Error a where
  noMsg :: a
  strMsg :: String → a
  noMsg = strMsg ""
  strMsg _ = noMsg
```

Vigade instantsid

```
instance Error [Char] where
  noMsg = ""
  strMsg = id
instance Error IOError where
  strMsg = userError
```

Monaaditeisendajad: veatöötlusmonaad

Veatöötlusmonaad

$$
\begin{aligned}
&\textbf{instance } \mathit{Error}\ e \Rightarrow \mathit{Monad}\ (\mathit{Either}\ e) \textbf{ where} \\
&\quad \mathit{return} \qquad\quad = \mathit{Right} \\
&\quad \mathit{Left}\ l \quad >\!\!>\!= \_ = \mathit{Left}\ l \\
&\quad \mathit{Right}\ r >\!\!>\!= k = k\ r \\
&\quad \mathit{fail}\ \mathit{msg} \qquad = \mathit{Left}\ (\mathit{strMsg}\ \mathit{msg})
\end{aligned}
$$

NB!

Võrreldes *Maybe*-monaadiga, võimaldab veatöötlusmonaad infomatiivsemaid veateateid.

Monaaditeisendajad: veatöötlusmonaad

Veatöötlusmonaadi operatsioonide liides

```
class Monad m ⇒ MonadError e m | m → e where
  throwError :: e → m a
  catchError :: m a → (e → m a) → m a

instance Error e ⇒ MonadError e (Either e) where
  throwError             = Left
  Left l  `catchError` h = h l
  Right r `catchError` _ = Right r
```

Monaaditeisendajad: veatöötlusmonaad

Parametriseeritud veatöötlusmonaad

```
newtype ErrorT e m a = ErrorT{
  runErrorT :: m (Either e a)
}
instance (Monad m, Error e) ⇒ Monad (ErrorT e m) where
  return a = ErrorT $ return (Right a)
  m >>= k  = ErrorT $ do
    a ← runErrorT m
    case a of
      Left  l → return (Left l)
      Right r → runErrorT (k r)
  fail msg = ErrorT $ return (Left (strMsg msg))
instance Error e ⇒ MonadTrans (ErrorT e) where
  lift m   = ErrorT $ do
               a ← m
               return (Right a)
```

Monaaditeisendajad: veatöötlusmonaad

Parametriseeritud veatöötlusmonaadi operatsioonid

```
instance (Monad m, Error e) ⇒
                              MonadError e (ErrorT e m) where
  throwError l      = ErrorT $ return (Left l)
  m `catchError` h = ErrorT $ do
    a ← runErrorT m
    case a of
      Left l → runErrorT (h l)
      Right r → return (Right r)
```

"Kergitatud" lisafunktsonaalsus

```
instance (Error e, MonadState s m) ⇒
                              MonadState s (ErrorT e m) where
  get = lift get
  put = lift ∘ put
```

Monaaditeisendajad: veatöötlusmonaad

MonadError liidese "kergitamine"

```
instance (MonadError e m) ⇒
                    MonadError e (ListT m) where
  throwError        = lift ∘ throwError
  m `catchError` h  = ListT $ runListT m
                      `catchError` λe → runListT (h e)
instance (MonadError e m) ⇒
                    MonadError e (StateT s m) where
  throwError        = lift ∘ throwError
  m `catchError` h  = StateT $ λs → runStateT m s
                      `catchError` λe → runStateT (h e) s
```

Monaaditeisendajad: jätkuedastusmonaad

Näide: *square* otsestiilis

$$
\begin{aligned}
&square :: Int \to Int \\
&square\ x = x * x \\
&main \quad = \mathbf{do} \\
&\qquad\quad \mathbf{let}\ x = square\ 3 \\
&\qquad\quad print\ x
\end{aligned}
$$

Näide: *square* jätkuedastusstiilis

$$
\begin{aligned}
&square :: Int \to (Int \to a) \to a \\
&square\ x\ k = k\ (x * x) \\
&main = square\ 3\ print
\end{aligned}
$$

Monaaditeisendajad: jätkuedastusmonaad

Jätkuedastusmonaad

```
newtype Cont r a = Cont{
  runCont :: (a → r) → r
}
instance Monad (Cont r) where
  return a = Cont ($a)
  m >>= k  = Cont $ λc → runCont m $ λa →
                         runCont (k a) c
```

Näide: *square* jätkuedastusmonaadis

```
square :: Int → Cont r Int
square x = return (x * x)
main     = runCont (square 3) print
```

Monaaditeisendajad: jätkuedastusmonaad

Jätkuedastusmonaadi operatsioonide liides

```
class (Monad m) ⇒ MonadCont m where
  callCC :: ((a → m b) → m a) → m a
```

NB!

callCC tähendab "*call with the current continuation*".

Jätkuedastusmonaadi instants

```
instance MonadCont (Cont r) where
  callCC f = Cont $ λc →
                runCont (f (λa → Cont $ λ_ → c a)) c
```

Monaaditeisendajad: jätkuedastusmonaad

Parametriseeritud jätkuedastusmonaad

```
newtype ContT r m a = ContT{
  runContT :: (a → m r) → m r
}
instance Monad m ⇒ Monad (ContT r m) where
  return a  = ContT ($a)
  m >>= k   = ContT $ λc →
                runContT m (λa → runContT (k a) c)
instance (Monad m) ⇒ MonadCont (ContT r m) where
  callCC f = ContT $ λc →
                runContT (f (λa → ContT $ λ_ → c a)) c
instance MonadTrans (ContT r) where
  lift m    = ContT (m>>=)
```

Monaaditeisendajad: jätkuedastusmonaad

MonadState liidese "kergitamine"

```
instance (MonadState s m) ⇒
                          MonadState s (ContT r m) where
  get = lift get
  put = lift ∘ put
```

MonadCont liidese "kergitamine"

```
instance (MonadCont m) ⇒
                       MonadCont (StateT s m) where
  callCC f = StateT $ λs →
    callCC $ λc →
    runStateT (f (λa → StateT $ λs′ → c (a, s′))) s
```

Monaaditeisendajad: jätkuedastusmonaad

Näide 1

```
bar :: Cont r Int
bar = callCC $ λk → do
        let n = 5
        k n
        return 25
main = runCont bar print
```

Näide 2

```
foo :: Int → Cont r String
foo n = callCC $ k → do
          let x = n * n + 3
          when (x > 20) $ k "over twenty"
          return (show $ x - 4)
```

Monaaditeisendajad: jätkuedastusmonaad

Näide 3

```
divExcpt :: Int → Int → (String → Cont r Int) → Cont r Int
divExcpt x y handler =
  callCC $ λok → do
    err ← callCC $ λnotOk → do
      when (y == 0) $ notOk "Denominator 0"
      ok $ x `div` y
    handler err
```

Monaaditeisendajad: jätkuedastusmonaad

Näide 4: keerulise kontrollvooga funktsioon

Sisend (n)	Väljund	Väljundlist
0-9	n	puudub
10-199	arvu (n/2) numbrite arv	arvu (n/2) numbrid
200-19999	n	arvu (n/2) numbrid
20000-1999999	(n/2) tagurpidi	puudub
$\geqslant$ 2000000	arvu (n/2) numbrite summa	arvu (n/2) numbrid

Monaaditeisendajad: jätkuedastusmonaad

Näide 4 (järg)

```
fun :: Int → String
fun n = (`runCont` id) $ do
  str ← callCC $ λexit1 → do
    when (n < 10) (exit1 $ show n)
    let ns = map digitToInt (show $ n `div` 2)
    n' ← callCC $ λexit2 → do
      when (length ns < 3) (exit2 $ length ns)
      when (length ns < 5) (exit2 n)
      when (length ns < 7) $ do
        let ns' = map intToDigit (reverse ns)
        exit1 (dropWhile (=='0') ns')
      return $ sum ns
    return $ "(ns = " ++ show ns ++ ")  " ++ show n'
  return $ "Answer: " ++ str
```

Monaaditeisendajad: jätkuedastusmonaad

Näide: listielementide korrutis (ver. 4)

```
prod xs = (`runCont` id) $ do
  callCC $ λk →
    let prod' []       = return 1
        prod' (0 : xs) = k 0
        prod' (x : xs) = do n ← prod' xs
                            return (x * n)
    in prod' xs
```