Metody programowania 2015

Lista zadań nr 15

Na zajęcia 15 czerwca 2015

Niech

```
class Monad m ⇒ Failable m where
   failure :: m a
   handle :: m a → a → a

instance Failable Maybe where
   failure = Nothing
   (Just x) `handle` _ = x
   Nothing `handle` x = x

class Monad (m s) ⇒ Stateful m s where
   update :: (s → s) → m s s
   getState :: m s s
   getState = update id
   putState :: s → m s ()
   putState s = update (const s) >> return ()

newtype SC s a = SC { exec :: s → (a,s) }

instance Monad (SC s) where
   t >>= f = SC $ uncurry (exec . f) . exec t
   return = SC . (,)

newtype Id a = Id { unId :: a }

newtype Writer a =
   Writer { unWriter :: (a, String) }

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

Zadanie 1 (2 pkt). Zainstaluj typy `Id`, `Writer` i `(s→)` w klasie `Monad`. Zdefiniuj typy

```
IdT, WriterT, ReaderT, MaybeT, ListT,
   StateT s :: (* → *) → * → *
```

będące transformatorami monad odpowiadającymi monadom `Id`, `Writer`, `(s→)`, `Maybe`, `[]` i `StateComp s`. Dla dowolnej monady `m` zainstaluj typy

```
IdT m, WriterT m, ReaderT s m, MaybeT m, ListT m,
   StateT s m :: * → *
```

w klasie `Monad`. *Uwaga:* nie zawsze powyższe typy będą spełniać aksjomaty monad! Zainstaluj typy `IdT`, `WriterT`, `(s→)`, `MaybeT`, `ListT`, `StateT s` w klasie `MonadTrans`. Dla dowolnej monady `m` zainstaluj typ `MaybeT m` w klasie `Failable`, typ `StateT s m` w klasie `Stateful`, a typ `ListT m` w klasie `MonadPlus`.

Zadanie 2 (1 pkt). Jak działa typ `MaybeT Id`? W jakim sensie jest on izomorficzny z `Maybe`? Rozważ typy

```
MaybeT (StateComp s)
   StateT s Maybe
MaybeT (StateT s Id)
StateT s (MaybeT Id)
```

W jakim sensie są one izomorficzne?

Zadanie 3 (1 pkt). Oto parser

```
newtype Parser token m value =
   Parser ([token] → m ([token],value))
```

tj. monada stanowa, w której stanem obliczeń jest lista tokenów (typu `token`), a dostarczaniem wyników parsowania zajmuje się monada `m` (np. lista, jeśli chcemy mieć parser z nawracaniem lub `Maybe`, jeśli wolimy parser deterministyczny).

Zainstaluj typ `Parser token m` w klasie `MonadPlus` i zaprogramuj biblioteczkę następujących kombinatorów parsujących:

```
parse :: Monad m ⇒ Parser token m value
            → [token] → m value
isElem :: (Eq token, MonadPlus m) ⇒ [token]
            → Parser token m token
isEmpty :: MonadPlus m ⇒ Parser token m ()
many :: MonadPlus m ⇒ Parser token m value
          → Parser token m [value]
many1 :: MonadPlus m ⇒ Parser token m value
            → Parser token m [value]
option :: MonadPlus m ⇒ Parser token m value
            → Parser token m (Maybe value)
```

Zadanie 4 (1 pkt). Do strumienia tokenów dodajemy dodatkowy stan `st`:

```
newtype Parser token st m value
   = Parser (([token],st)
      → m ([token],st,value))
```

Rozwiąż poprzednie zadanie dla tej implementacji.

Zadanie 5 (1 pkt). Rozważmy wyrażenia złożone z identyfikatorów (ciągi małych i wielkich liter oraz cyfr zaczynające się literą), literałów całkowitoliczbowych (niepuste ciągi cyfr) oraz operatorów `+`, `-`, `*`, `/`, `^`. Ostatni operator (potęgowanie) wiąże najsilniej i w prawo, pozostałe operatory — w lewo, przy czym `+` i `-` słabiej, niż `*` i `/`. W wyrażeniach można ponadto używać nawiasów i konstrukcji x`=`e`:`e', która wiąże najsłabiej i oznacza związanie wartości wyrażenia e z identyfikatorem x w wyrażeniu e', np. wyrażenie

```
x=1+2*3:2*x^2
```

ma wartość 98.

Uczyń stanem obliczeń prasera z poprzedniego zadania słownik odwzorowujący nazwy identyfikatorów w liczby całkowite (tak by nie trzeba go było jawnie przekazywać do i z funkcji parsującej) i napisz kalkulator wyznaczający wartość opisanych wyżej wyrażeń.