

Języki i paradygmaty programowania (Info, III rok) 16/17

Kokpit ► Moje kursy ► JiPP.INFO.III.16/17 ► 27.3-2.4 Składnia 1 ► Lab Monady 2

Lab Monady 2

Uwaga: na poniższe zadania można przeznaczyć 3, a w miarę potrzeby i możliwości nawet 4 zajęcia.

Zadanie 1. Monada List

Funkcja

```
allPairs :: [a] -> [a] -> [[a]]
allPairs xs ys = [[x,y] | x <- xs, y <- ys]
```

daje listę wszystkich list dwuelementowych, gdzie pierwszy element należy do pierwszego argumentu, drugi do drugiego, np.

```
*Main> allPairs [1,2,3] [4,5]
[[1,4],[1,5],[2,4],[2,5],[3,4],[3,5]]
```

- przepisz na notację monadyczną
- uogólnij tę funkcję do `allCombinations :: [[a]] -> [[a]]`, która dla n-elementowej listy list da listę wszystkich n-elementowych list takich, że i-ty element należy do i-tego elementu argumentu, np

```
Main> allCombinations [[1,2], [4,5], [6], [7]]
[[1,4,6,7],[1,5,6,7],[2,4,6,7],[2,5,6,7]]
```

Zadanie 2. Reader

a.

```
data Tree a = Empty | Node a (Tree a) (Tree a) deriving (Eq, Ord, Show)
```

Napisz funkcję

```
renumber :: Tree a -> Tree Int
```

która dla danego drzewa da drzewo, gdzie w każdym węźle przechowywana będzie głębokość tego węzła (odległość od korzenia).

Porównaj rozwiązania z użyciem monady Reader i bez.

b. Dane typy dla wyrażeń arytmetycznych

```
type Var = String
data Exp = EInt Int
         | EOp Op Exp Exp
         | EVar Var
         | ELet Var Exp Exp  -- let var = e1 in e2

data Op = OpAdd | OpMul | OpSub
```

Napisz funkcję

```
evalExp :: Exp -> Int
```

która obliczy wartość takiego wyrażenia, np

```
--
--      let x =
--          let y = 6 + 9
--          in y - 1
--      in x * 3
--
-- ==>  42
--
test = ELet "x" (ELet "y" (EOp OpAdd (EInt 6) (EInt 9))
                      (EOp OpSub y (EInt 1)))
      (EOp OpMul x (EInt 3))
  where x = EVar "x"
        y = EVar "y"
```

(można też użyć typu wyrażeń z jednego z poprzednich labów)

Użyj monady czytelnika środowiska (Reader). Środowisko może być np. jednego z typów

```
Map Var Int
[(Var, Int)]
Var -> Maybe Int
```

Zadanie 3. Monada State

a. Dany typ drzew

```
data Tree a = Empty | Node a (Tree a) (Tree a) deriving (Eq, Ord, Show)
```

Napisz funkcję

```
renumberTree :: Tree a -> Tree Int
```

która ponumeruje węzły drzewa tak, że każdy z nich będzie miał inny numer. Porównaj rozwiązania z użyciem monady State i bez.

możliwe dodatkowe wymaganie: ponumeruj węzły drzewa w kolejności infiksowej.

```
(toList $ renumber $ fromList "Learn Haskell") == [0..12]
```

b. Rozszerzmy język z poprzedniego zadania o instrukcje języka Tiny (patrz przedmiot Semantyka i Weryfikacja Programów)

```
Stmt:  S ::= skip | x := e | S1;S2
      | if b then S1 else S2 | while b do S
```

korzystając z wcześniej napisanej funkcji `evalExp`, napisz funkcję

```
execStmt :: Stmt -> IO ()
```

która wykona podaną instrukcję (program) i wypisze stan końcowy (w tym wypadku wartości zmiennych)

c. Następny krok to dodanie deklaracji zmiennych lokalnych:

```
Stmt: S ::= ... | begin [D] S end
Decl: D ::= var x=e;
```

Tutaj dla obliczania Stmt najlepiej użyć jednocześnie monady Reader i State: część Reader odczytuje funkcje z Var w nowy typ "lokacji pamięci" Loc(=Int), a część State operuje na funkcjach z Loc w Int, implementowanych jako mapy. Trzeba też zaimplementować funkcję `alloc :: (Map Loc Int) -> Loc` która zwraca nieużywaną lokację.

Zadanie 4. Transformatory monad

a. Zaimplementuj moduły StateTParser i SimpleParsers z wykładu

b. Zaimplementuj moduł IdentityTrans dostarczający identycznościowego transformatora IdentityT:

```
newtype IdentityT = ...
instance (Functor m) => Functor (IdentityT m) where
instance (Monad m) => Monad (IdentityT m) ...
instance MonadTrans IdentityT ...
instance MonadPlus m => MonadPlus (IdentityT m) ...
```

c. Zaimplementuj moduł MaybeTrans dostarczający transformatora MaybeT - analogicznie jak wyżej, za wyjątkiem

```
instance MonadPlus (MaybeT m) ...
```

d. Zaimplementuj moduł StateTParser2 z wykładu:

```
module StateTParser2(Parser,runParser,item) where
import Control.Monad.State
import MaybeTrans
import Control.Monad.Identity

-- Use the StateT transformer on MaybeT on Identity
type Parser a = StateT [Char] (MaybeT Identity) a

runParser :: Parser a -> [Char] -> Maybe (a,String)
item :: Parser Char
```

i przetestuj go z SimpleParsers (zastępując import StateTParser przez import StateTParser2)

```

module SimpleParsers where
import Data.Char(isDigit,digitToInt)
import Control.Monad

import StateTParser2

pDigit1 :: Parser Int
pDigit1 = item >= test where
    test x | isDigit x = return $ digitToInt x
           | otherwise = mzero

sat :: (Char->Bool) -> Parser Char
sat p = do {x <- item; if p x then return x else mzero}

char :: Char -> Parser Char
char x = sat (==x)

pDigit :: Parser Int
pDigit = fmap digitToInt $ sat isDigit

pDigits :: Parser [Int]
pDigits = many pDigit

pNat :: Parser Int
-- pNat = pDigits >= return . foldl (\x y -> 10*x+y) 0
pNat = fmap (foldl (\x y -> 10*x+y) 0) pDigits

many :: Parser a -> Parser [a]
many p = many1 p `mplus` return []

many1 p = do { a <- p; as <- many p; return (a:as)}

test123 = runParser pNat "123 ala"

```

Ostatnia modyfikacja: piątek, 27 marzec 2015, 20:44

NAWIGACJA



Kokpit

- Strona główna

Strony

Moje kursy


JNPI.INFO.II.16/17

JiPP.INFO.III.16/17

Uczestnicy

 Odznaki

 Kompetencje

 Oceny

Główne składowe

T1 27.2-5.3 Haskell 1

T2 6-12.3 Haskell 2


T3 13-19.3 Monady 1

20-26.3 Monady 2

27.3-2.4 Składnia 1

 **Lab Monady 2**

 SimpleParsers.hs

 Wykład 5: Składnia 1

 MaybeTrans.hs

3.4-9.4

10.4-23.4

24-30.4

1.5-14.5

15-21.5

22.5-28.5

29.5-4.6

5.6-11.6

12-14.6

Bonus

TOPI*.MAT.16/17

ADMINISTRACJA



Administracja kursem

Jesteś zalogowany(a) jako Szymon Pajzert (Wyloguj)
JiPP.INFO.III.16/17

Moodle, wersja 3.2.2+ | moodle@mimuw.edu.pl