Programare funcțională

Introducere în programarea funcțională folosind Haskell
C13

Ana Iova
Denisa Diaconescu

Departamentul de Informatică, FMI, UB

Monada IO

Monada IO

IO a corespunde comenzilor care produc rezultate de tip a.

Exemplu: citește un caracter

IO Char corespunde comenzilor care produc rezultate de tip **Char**

```
getChar :: IO Char
```

Poate sa citeasca de la tastatura o litera.

IO () corespunde comenzilor care nu produc rezultate

- () este tipul unitate care conține doar valoarea ()

Combinarea comenzilor cu valori

```
(>>=) :: IO a -> (a -> IO b) -> IO b
```

Exemplu

```
getChar :: IO Char
putChar :: Char -> IO ()

getChar >>= \x -> putChar (toUpper x)
```

Poate sa citeasca de la tastatura o litera si sa o afiseze pe ecran, trasformata in litera mare.

Operatorul de legare / bind

```
(>>=) :: IO a -> (a -> IO b) -> IO b
```

- Dacă fiind o comandă care produce o valoare de tip a
  m :: **IO** a
- Data fiind o funcție care pentru o valoare de tip a se evaluează la o comandă de tip b
  k :: a -> **IO** b
- Atunci
  m >>= k :: **IO** b
  este comanda care, dacă se va executa:
  - Mai întâi efectuează m, obținând valoarea x de tip a
  - Apoi efectuează comanda k x obținând o valoare y de tip b
  - Produce y ca rezultat al comenzii

Exemplu: citește o linie

```
myGetLine :: IO String
myGetLine = getChar >>= \x ->
                    if x == '\n' then
                       return []
                    else
                       myGetLine >>= \xs ->
                       return (x:xs)
```

Exemplu: citește o linie in notatia "do"

Urmatoarele sunt echivalente:

```
myLine :: IO String
myLine = getChar >>= \x ->
         if x == '\n' then
           return []
         else
           myLine >>= \xs ->
           return (x:xs)
```

```
myLineDo :: IO String
myLineDo = do
           x <- getChar
           if x == '\n' then
             return []
           else do
             xs <- myLineDo
             return (x:xs)
```

Exemplu: de la intrare la ieșire

```
echo :: IO ()
echo = getLine >>= \line ->
         if line == "" then
           return ()
         else
           putStrLn (map toUpper line) >>
           echo
```

```
*C13> echo
One line
ONE LINE
And, another line!
AND, ANOTHER LINE!
...
```

Exemplu: de la intrare la ieșire in notatia "do"

```
echo :: IO ()
echo = getLine >>= \line ->
       if line == "" then
         return ()
       else
         putStrLn (map toUpper line) >>
         echo
```

Echivalent cu

```
echoDo :: IO ()
echoDo = do
       line <- getLine
       if line == "" then
           return ()
       else do
           putStrLn (map toUpper line)
           echoDo
```

Exemplu: citirea si afisarea unui numar

```
myNumber = putStrLn "Please enter a number: " >>
           readLn >>= \n ->
           let m = n + 1 in
           print m -- putStrLn (show m)

myNumberDo = do
           putStrLn "Please enter a number: "
           n <- readLn
           let m = n + 1
           print m
```

Examen

Evaluare

Notare

- Nota finală: 1 (oficiu) + parțial + examen
- Restanță: 1 (oficiu) + examen
  (parțialul nu se ia în calcul la restanță)

Condiție de promovabilitate

- cel puțin 5 > 4.99

Puncte bonus

Activitate laborator

- La sugestia instructorilor de la laborator, se poate nota activitatea în plus față de cerințele obișnuite.
- Maxim 1 punct (bonus la nota finală)

Punctele bonus nu se pot folosi în condiția de promovabilitate!

Punctele bonus nu se pot folosi în restanță!

Examen final

- valorează 6 puncte din nota finală
- 1 februarie
  - seria 23 + 251 - ora 10:00 [o ora]
  - seria 24 + 252 - ora 12:00 [o ora]
- pe calculatoarele din laboratoarele facultatii
- vom posta pe Teams o distributie in laboratoare
- acoperă toată materia
- materiale ajutătoare: suporturile de curs si de laborator
- fara internet
- va conține exerciții asemănătoare cu cele de la laborator [vezi model de examen]

Concluzii

Conținutul cursului de PF 2022

- Elemente de baza
- Functii
- Liste
- Currying
- Operatori. Sectiuni
- Functii de nivel inalt
- Tipuri de date algebrice
- Clase de tipuri
- Functor
- Semigroup
- Monoid
- Foldable
- Applicative
- Monad

Haskell - best for

Application Domains

- Compilers
- Server-side web programming
- Scripting / Command-line applications
- ...

Common Programming Needs

- Maintenance
- Single-machine Concurrency
- Types / Type-driven development
- Parsing / Pretty-printing
- Domain-specific languages (DSLs)
- ...

Sursa https://github.com/Gabriella439/post-rfc/blob/main/sotu.md

More to explore

- Testing; Property-Based Testing
- Random Numbers
- Monad Transformers
- Parallel and Concurrent Programming
- Dependent Types

Feedback

Seria 23: https://questionpro.com/t/AT4qgZwUpF

Seria 24: https://questionpro.com/t/AT4NiZwUI9

Seria 25: https://questionpro.com/t/AT4qgZwUpH

Mulțumim că ați participat la acest curs!

Multă baftă la examen!