

Programare declarativă

Introducere în programarea funcțională folosind Haskell

Ioana Leuștean
Ana Cristina Turlea

Departamentul de Informatică, FMI, UB
ioana@fmi.unibuc.ro
ana.turlea@fmi.unibuc.ro

Organizare

Plan curs

Programare funcțională (folosind Haskell)

- Funcții, recursie, funcții de ordin înalt, tipuri
- Operații pe liste: filtrare, transformare, agregare
- Polimorfism, clase de tipuri, modularizare
- Tipuri de date algebrice - evaluarea expresiilor
- Operațiuni Intrare/Ieșire
- Agregare pe tipuri algebrice
- Functori, monade

Resurse

- Paginile cursului:
 - <http://moodle.fmi.unibuc.ro/course/view.php?id=367> (Seria 33)
 - <http://moodle.fmi.unibuc.ro/course/view.php?id=668> (Seria 34)
 - <http://cs.unibuc.ro/~ileustean/PD.html>
 - Prezentările cursurilor, forumuri, resurse electronice
 - Știri legate de curs vor fi postate pe ambele pagini și pe moodle
- <https://tinyurl.com/y5txtkwz> sau <http://bit.do/unibuc-pd>
 - Cele mai noi variante ale cursurilor si laboratoarelor.
- Cartea online „Learn You a Haskell for Great Good”
<http://learnyouahaskell.com/>
- Pagina Haskell <http://haskell.org>
 - Hoogle <https://www.haskell.org/hoogle>
 - Haskell Wiki <http://wiki.haskell.org>

Evaluare

Notare

- Testare laborator (lab), examen (ex)
- Nota finală: 1 (oficiu) + lab + ex

Condiție de promovabilitate

- Nota finală **cel puțin 5**
 - $5 > 4.99$

Activitate laborator

- La sugestia profesorului coordonator al laboratorului, se poate nota activitatea în plus față de cerințele obșnuite.
- Maxim 1 punct (bonus la nota finală)

Test laborator

- Valorează 2 puncte din nota finală
- În săptămâna a 7-a
- Pe calculatoare - la laborator
- Acoperă materia din laboratoarele 1-6
- Fără acces la rețea/internet și fără materialele de la curs/laborator
- Materiale ajutătoare: funcții utile

Examen final

- Valorează 7 puncte din nota finală
- În sesiune
- Pe calculator
- Acoperă toată materia
- Durată: 2 ore
- Materiale ajutătoare: cursul tipărit și legat
- Fără acces la rețea/internet

Evaluare

- Prezența la testarea din săptămîna a 7-a nu este obligatorie, **dar**
- studenții absenți la testarea din săptămîna a 7-a vor pierde 2 puncte din nota finală.
- În restanțe se va da un singur examen!
- NU se păstrează punctaje parțiale între sesiuni!

Legătură foarte utilă!

https://wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems

Programare funcțională

Programare declarativă vs. imperativă

Ce vs. cum

Programare imperativă (Cum)

Explic mașinii, pas cu pas, algoritmic, **cum** să facă ceva și se întâmplă **ce** voiam să se întâmple ca rezultat al execuției mașinii.

- limbaje procedurale
- limbaje de programare orientate pe obiecte

Programare declarativă (Ce)

Îi spun mașinii **ce** vreau să se întâmple și o las pe ea să se descurce **cum** să realizeze acest lucru. :-)

- limbaje de programare logică
- limbaje de interogare a bazelor de date
- limbaje de programare funcțională

Agregarea datelor dintr-o colecție (JS)

C. Boesch, Declarative vs Imperative Programming - Talk.JS

<https://www.youtube.com/watch?v=M2e5sq1rnvc>



Engineers.SG
Meetups Videos

Will I ever prefer to read declarative javascript?

```
function multiply(array) {  
  return array.reduce( (a,b) => a*b, 1);  
}
```

```
function multiply(array) {  
  var total = 1;  
  for (var i = 0; i < array.length; i++){  
    total = total * array[i];  
  }  
  return total;  
}
```



Agregarea datelor dintr-o colecție (JS)

C. Boesch, Declarative vs Imperative Programming - Talk.JS

<https://www.youtube.com/watch?v=M2e5sq1rnvc>



Engineers.SG
Meetups Videos

Reasons to be More Declarative

- Better readability
- Better scalability
- Fewer state-related bugs
- Stand on the shoulders of giants



Programare funcțională

- Programare funcțională în limbajul vostru preferat de programare:
 - Java 8, C++11, C#, Python, PHP, JavaScript, Ruby, ...
 - Funcții anonime (λ -abstracții)
 - Funcții de procesare a fluxurilor de date: filter, map, reduce

λ -calcul (A. Church, 1929-1932)

$t =$	x	(variabilă)
	$ \lambda x. t$	(abstractizare)
	$ t t$	(aplicare)

Funcții anonime în Haskell

În Haskell, \backslash e folosit în locul simbolului λ și \rightarrow în locul punctului.

$\lambda x. x * x$ este $\backslash x \rightarrow x * x$

$\lambda x. x > 0$ este $\backslash x \rightarrow x > 0$

Programare funcțională în Haskell

- Haskell e folosit în proiecte de Facebook, Google, Microsoft, ...
 - Programarea funcțională e din ce în ce mai importantă în industrie
 - mai multe la https://wiki.haskell.org/Haskell_in_industry
- Oferă suport pentru paralelism și concurență.



De ce Haskell? (din cartea Real World Haskell)

The illustration on our cover is of a **Hercules beetle**. These beetles are among the largest in the world. They are also, in proportion to their size, the strongest animals on Earth, able to lift up to 850 times their own weight. Needless to say, we like the association with a creature that has such a high power-to-weight ratio.

Haskell este un limbaj funcțional pur



- Funcțiile sunt valori.
- În loc să modificăm datele existente, calculăm valori noi din valorile existente, folosind funcții
- Funcțiile sunt **pure**: aceleași rezultate pentru aceleași intrări.
- O bucată de cod nu poate corupe datele altei bucăți de cod.
- Distincție clară între părțile pure și cele care comunică cu mediul extern.

Haskell este un limbaj elegant

- Idei abstracte din matematică devin instrumente puternice practice
 - recursivitate, compunerea de funcții, functori, monade
 - folosirea lor permite scrierea de cod compact și modular
- Rigurozitate: ne forțează să gândim mai mult înainte, dar ne ajută să scriem cod mai corect și mai curat
- Curbă de învățare în trepte
 - Putem scrie programe mici destul de repede
 - Expertiza în Haskell necesită multă **gândire** și **practică**
 - Descoperirea unei lumi noi poate fi un drum distractiv și provocator
<http://wiki.haskell.org/Humor>



- Haskell e **leneș**: orice calcul e amânat cât de mult posibil
 - Schimbă modul de concepere al programelor
 - Permite lucrul cu colecții potențial infinite de date precum [1..]
 - Evaluarea leneșă poate fi exploatată pentru a reduce timpul de calcul fără a denatura codul

```
firstK k xs = take k (sort xs)
```

- Haskell e **minimalist**: mai puțin cod, în mai puțin timp, și cu mai puține defecte
 - ...rezolvând totuși problema :-)

```
numbers = [1,2,3,4,5]  
total = foldl (*) 0 numbers  
doubled = map (* 2) numbers
```

Exemplu

```
qsort :: Ord a => [a] -> [a]
```

```
qsort [] = []
```

```
qsort (p:xs) = (qsort lesser) ++ [p] ++ (qsort greater)  
  where  
    lesser  = filter (< p) xs  
    greater = filter (>= p) xs
```

Exemplu

```

import Control.Monad (foldM)
import Data.List ((\\))

queens :: Int -> [[Int]]
queens n = foldM f [] [1..n]
  where
    f qs _ = [q:qs | q <- [1..n] \\ qs, q 'notDiag' qs]
    q 'notDiag' qs = and [abs (q - qi) /= i |
                          (qi,i) <- qs 'zip' [1..]]

```

https://rosettacode.org/wiki/N-queens_problem

Succes!