

Programare funcțională

Introducere în programarea funcțională folosind Haskell

Ioana Leuștean
Traian Florin Șerbănuță

Departamentul de Informatică, FMI, UB
ioana@fmi.unibuc.ro
traian.serbanuta@unibuc.ro

Organizare

Resurse

- Instructor curs: Traian Florin Șerbănuță
- Instructori laborator:
 - 231 Traian Șerbănuță
 - 232 Ana Pantilie
 - 233 Călin Nicolau
 - 234 Adrian Budău
- Paginile cursului:
 - Moodle UB
 - Microsoft Teams
 - Prezentările cursurilor, forumuri, resurse electronice
 - <http://bit.do/unibuc-pf>
 - Dropbox cu cele mai noi variante ale cursurilor si laboratoarelor.

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)

Evaluare

- Test laborator
 - Din prima parte a materiei
 - Valorează 3 puncte din nota finală
 - În săptămâna 23-29 noiembrie
 - Prima ora din curs, test pe moodle
- Examen final
 - Valorează 6 puncte din nota finală
 - În sesiune
 - Acoperă toată materia
 - Durată: 1-2 ore

Plan curs

Programare funcțională în 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
- Functori, monade

Resurse

- Pagina Haskell <http://haskell.org>
 - Hoogle <https://www.haskell.org/hoogle>
 - Haskell Wiki <http://wiki.haskell.org>
 - Haskell Humor <https://wiki.haskell.org/Humor>
- Cartea „Haskell Programming from first principles”
<https://haskellbook.com/>
- Cartea online „Learn You a Haskell for Great Good”
<http://learnyouahaskell.com/>
- [https://wiki.haskell.org/H-99:
_Ninety-Nine_Haskell_Problems](https://wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems)
- ...

Programare funcțională

Programare imperativă vs. declarativă

Cum vs. Ce

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 imperativă vs. declarativă

Cum vs. Ce

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:

- Funcții anonime (λ -abstracții)

Java 8 $(x, y) \rightarrow x * y$

C++ 11 $[](x, y) \{ \text{return } x * y; \}$

Python **lambda** $x, y: x * y$

JavaScript $(x, y) \Rightarrow x * y$

Haskell $\backslash x y \rightarrow x * y$

- Funcții de procesare a fluxurilor de date: map, filter, reduce/fold

λ -calcul

- În 1929-1932 Church a propus λ -calculul ca sistem formal pentru logica matematică. În 1935 a argumentat că orice funcție calculabilă peste numere naturale poate fi calculată în λ -calcul.

$$\begin{array}{ll}
 t = & x \quad \text{(variabilă)} \\
 & | \lambda x. t \quad \text{(abstractizare)} \\
 & | t \ t \quad \text{(aplicare)}
 \end{array}$$

- În 1935, independent de Church, Turing a dezvoltat mecanismul de calcul numit astăzi Mașina Turing. În 1936 și el a argumentat că orice funcție calculabilă peste numere naturale poate fi calculată de o mașină Turing. De asemenea, a arătat echivalența celor două modele de calcul. Această echivalență a constituit o indicație puternică asupra "universalității" celor două modele, conducând la ceea ce numim astăzi "Teza Church-Turing".

Programare funcțională în Haskell

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.

Programare funcțională în Haskell



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.

```
primes = sieve [2..]
```

```
sieve (p:ps) = p : sieve [ x | x <- ps, x `mod` p /= 0 ]
```


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 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.

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, modular și reutilizabil
- 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

```
firstPrimes k = take k primes
```

- 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 = foldr (+) 0 numbers  
doubled = map (* 2) numbers
```

Exemplu

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

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

```
qsort (p:xs) = (qsort lesser) ++ [p] ++ (qsort greater)  
  where  
    (lesser, greater) = partition (< p) xs
```

Succes!