# Paradigme de Programare

S.I. dr. ing. Andrei Olaru
slides: Mihnea Muraru si Andrei Olaru

Catedra de Calculatoare

2013 – 2013, semestrul 2

# Cursul 7

# Evaluare Lenesă în Haskell

# Cuprins

1 Evaluare lenesă în Haskell

# Evaluare lenesă

# Programare orientată spre date

Prelucrări traduse în termenii unor operații pe structuri de date, posibil niciodată generate complet!

1
2
3
4
5
6
7
8
9

# Programare orientată spre date

Prelucrări traduse în termenii unor operații pe structuri de date, posibil niciodată generate complet!

**Exemplul 28.1 (Suma pătratelor).**

Suma pătratelor numerelor naturale până la `n` ca sumă pe o listă:

# Programare orientată spre date

Prelucrări traduse în termenii unor operații pe structuri de date, posibil niciodată generate complet!

**Exemplul 28.1 (Suma pătratelor).**

Suma pătratelor numerelor naturale până la `n` ca sumă pe o listă:

```
1  sum (map (^2) [1 .. n])
2  → sum (map (^2) 1 : [2 .. n])
3  → sum (1^2 : (map (^2) [2 .. n]))
4  → 1^2 + sum (map (^2) [2 .. n])
5  → 1 + sum (map (^2) [2 .. n])
6  ...
7  → 1 + (4 + sum (map (^2) [3 .. n]))
8  ...
9  → 1 + (4 + (9 + ... + n^2))
```

# Programare orientată spre date

Prelucrări traduse în termenii unor operații pe structuri de date, posibil niciodată generate complet!

**Exemplul 28.1 (Suma pătratelor).**

Suma pătratelor numerelor naturale până la `n` ca sumă pe o listă:

```
1  sum (map (^2) [1 .. n])
2  → sum (map (^2) 1 : [2 .. n])
3  → sum (1^2 : (map (^2) [2 .. n]))
4  → 1^2 + sum (map (^2) [2 .. n])
5  → 1 + sum (map (^2) [2 .. n])
6  ...
7  → 1 + (4 + sum (map (^2) [3 .. n]))
8  ...
9  → 1 + (4 + (9 + ... + n^2))
```

Nicio listă nu este efectiv construită în timpul evaluării.

**Exemplul 28.2 (Minimul unei liste – definiție).**

Minimul unei liste, drept prim element al acesteia, după sortarea prin inserție.

```
32  ins x []          = [x]
33  ins x (h : t)
34      | x <= h      = x : h : t
35      | otherwise = h : (ins x t)
36
37  isort []          = []
38  isort (h : t)     = ins h (isort t)
39
40  minList l = head (isort l)
```

## Exemplul 28.3 (Minimul unei liste – execuție).

```
43 minList [3, 2, 1]
44 = head (isort [3, 2, 1])
45 = head (isort (3 : [2, 1]))
46 = head (ins 3 (isort [2, 1]))
47 = head (ins 3 (isort (2 : [1])))
48 = head (ins 3 (ins 2 (isort [1])))
49 = head (ins 3 (ins 2 (isort (1 : []))))
50 = head (ins 3 (ins 2 (ins 1 (isort []))))
51 = head (ins 3 (ins 2 (ins 1 [])))
52 = head (ins 3 (ins 2 (1 : [])))
53 = head (ins 3 (1 : ins 2 []))
54 = head (1 : (ins 3 (ins 2 [])))   = 1
```

Lista nu este efectiv sortată, minimul fiind, pur și simplu, tras în fața acesteia și întors.

# Backtracking eficient

Găsirea eficientă a unui obiect, prin generarea aparentă, a tuturor acestora.

**Exemplul 28.4 (Accesibilitatea într-un graf).**

Accesibilitatea între două noduri, ca existență a elementelor în mulțimea tuturor căilor dintre cele două noduri:

```
66 theGraph = [(1, 2), (1, 4), (2, 1), (2, 3),
67              (3, 5), (3, 6), (5, 6), (6, 1)]
68 accessible source dest graph =
69     (routes source dest graph []) /= []
```

# Backtracking eficient

Găsirea eficientă a unui obiect, prin generarea aparentă, a tuturor acestora.

**Exemplul 28.4 (Accesibilitatea într-un graf).**

Accesibilitatea între două noduri, ca existență a elementelor în mulțimea tuturor căilor dintre cele două noduri:

```
66 theGraph = [(1, 2), (1, 4), (2, 1), (2, 3),
67              (3, 5), (3, 6), (5, 6), (6, 1)]
68 accessible source dest graph =
69     (routes source dest graph []) /= []
```

Backtracking desfășurat doar până la determinarea primului element al listei.

# Backtracking eficient

## Continuare exemplu

**Exemplul 28.5 (Accesibilitatea într-un graf – căi).**

```
69  neighbors node = map snd . filter ((== node) . fst)
70
71  routes source dest graph explored
72   | source == dest = [[source]]
73   | otherwise      = [ source : path
74    | neighbor <- neighbors source graph \\ explored
75    , path <- routes neighbor dest graph (source : explored)
76    ]
```

# Studiu de caz

Bibliotecă de parsare (din bibliografie).

Fișierul `ParserAO.hs`, de văzut împreună cu codul echivalent de la seria CA/CB.

# Bibliografie

[Thompson, S. (1999), Haskell: The Craft of Functional Programming, Second Edition, Addison-Wesley.]