# Paradigme de Programare

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

Catedra de Calculatoare

2013 – 2013, semestrul 2

# Cursul 6

# Programare funcţională în Haskell

# Cuprins



# Introducere

# Paralelă între limbaje

| Criteriu | Scheme | Haskell |
|---|---|---|
| Funcţii | *Curry* sau *uncurry* | *Curry* |
| Tipare | Dinamică, slabă | Statică, tare |
| Legarea variabilelor | Locale $\rightarrow$ statică, *top-level* $\rightarrow$ dinamică | Statică |
| Evaluare | Aplicativă | Normală |
| Transferul parametrilor | *Call by sharing* | *Call by need* |
| Efecte laterale | `set!` & co. | Interzise |

# Funcţii

- *Curry*
- Aplicabile asupra oricâtor parametri la un moment dat

**Exemplul 24.1.**

Definiţii echivalente ale funcţiei `add`:

```
1  add1 x y     =    x + y
2  add2         =    \x -> \y -> x + y
3  add3         =    \x y -> x + y
4
5  result       =    add1 1 2     -- echivalent, ((add1 1) 2)
6  result2      =    add3 1 2     -- echivalent, ((add3 1) 2)
7  inc          =    add1 1
```

# Funcţii şi operatori

- Aplicabilitatea parţială a operatorilor infixaţi
- Transformări operator $\rightarrow$ funcţie şi funcţie $\rightarrow$ operator

**Exemplul 24.2.**

Definiţii echivalente ale funcţiilor `add` şi `inc`:

```
1  add4          =    (+)
2  result1       =    (+) 1 2
3  result2       =    1 `add4` 2
4
5  inc1          =    (1 +)
6  inc2          =    (+ 1)
7  inc3          =    (1 `add4`)
8  inc4          =    (`add4` 1)
```

# Pattern matching

- Definirea comportamentului funcţiilor pornind de la structura parametrilor → traducerea axiomelor TDA.

## Exemplul 24.3.

```
1  add5 0 y          =   y                    -- add5 1 2
2  add5 (x + 1) y    =   1 + add5 x y
3
4  sumList []        =   0                    -- sumList [1,2,3]
5  sumList (hd:tl)   =   hd + sumList tl
6
7  sumPair (x, y)    =   x + y                -- sumPair (1,2)
8
9  sumTriplet (x, y, z@(hd:_)) =              -- sumTriplet
10     x + y + hd + sumList z                 --  (1,2,[3,4,5])
```

# List comprehensions

- Definirea listelor prin proprietăţile elementelor, ca într-o specificare matematică

## Exemplul 24.4.

```
1  squares lst      =   [x * x | x <- lst]
2
3  quickSort []     =   []
4  quickSort (h:t)  =   quickSort [x | x <- t, x <= h]
5                   ++  [h]
6                   ++  quickSort [x | x <- t, x > h]
7
8  interval         =   [0 .. 10]
9  evenInterval     =   [0, 2 .. 10]
10 naturals         =   [0 ..]
```

# Tipare

# Tipuri

- Tipuri ca mulţimi de valori:
  - `Bool = {True, False}`
  - `Natural = {0, 1, 2, ...}`
  - `Char = {'a', 'b', 'c', ...}`
- Rolul tipurilor
- Tipare statică:
  - etapa de tipare anterioară etapei de evaluare
  - asocierea fiecărei expresii din program cu un tip
- Tipare tare: absenţa conversiilor implicite de tip
- Expresii de:
  - program: `5, 2 + 3, x && (not y)`
  - tip: `Integer, [Char], Char -> Bool, a`

# Exemple de tipuri

## Exemplul 25.1.

```
1  5                  :: Integer
2  'a'                :: Char
3  inc                :: Integer -> Integer
4  [1,2,3]            :: [Integer] -- liste de un singur tip
5  (True, "Hello")    :: (Bool, [Char])
6
7  etc.
```

- Tipurile de bază sunt tipurile elementare din limbaj:
  `Bool`, `Char`, `Integer`, `Int`, `Float`, ...

# Constructori de tip

- Funcţii de tip, ce îmbogăţesc tipurile din limbaj.

**Exemplul 25.2 (Constructori de tip predefiniți).**

```
1  -- Constructorul de tip functie: ->
2  (-> Bool Bool) ⇒ Bool -> Bool
3  (-> Bool (Bool -> Bool)) ⇒ Bool -> (Bool -> Bool)
4
5  -- Constructorul de tip lista: []
6  ([] Bool) ⇒ [Bool]
7  ([] [Bool]) ⇒ [[Bool]]
8
9  -- Constructorul de tip tuplu: (,...,)
10 ((,) Bool Char) ⇒ (Bool, Char)
11 ((,,) Bool ((,) Char [Bool]) Bool)
12 ⇒ (Bool, (Char, [Bool]), Bool)
```

# Tipurile funcţiilor

- Constructorul `->` este asociativ dreapta:

  `Integer -> Integer -> Integer`

  ≡ `Integer -> (Integer -> Integer)`

## Exemplul 25.3.

```
1  add6          ::   Integer -> Integer -> Integer
2  add6 x y      =    x + y
3
4  f             ::   (Integer -> Integer) -> Integer
5  f g           =    (g 3) + 1
6
7  idd           ::   a -> a          -- functie polimorfica
8  idd x         =    x               -- a: variabila de tip!
```

# Polimorfism

**Definiția 25.4 (Polimorfism parametric).**

Manifestarea aceluiaşi comportament pentru parametri de tipuri diferite. Exemplu: `idd`.

**Definiția 25.5 (Polimorfism ad-hoc).**

Manifestarea unor comportamente diferite pentru parametri de tipuri diferite. Exemplu: `==`.

# Constructorul de tip `Natural`

## Exemplu de definire TDA

**Exemplul 25.6.**

```
1  data Natural     =   Zero
2                   | Succ Natural
3      deriving (Show, Eq)
4
5  unu              =   Succ Zero
6  doi              =   Succ unu
7
8  addNat Zero n    =   n
9  addNat (Succ m) n =  Succ (addNat m n)
10
11 -- try addNat (Succ (Succ doi)) (Succ (Succ (Succ Zero)))
```

# Constructorul de tip `Natural`

## Comentarii

- Constructor de tip: `Natural`
  - nular
  - se confundă cu tipul pe care-l construieşte
- Constructori de date:
  - `Zero`: nular
  - `Succ`: unar
- Constructorii de date ca funcţii, dar utilizabile în *pattern matching*

```
1 Zero :: Natural
2 Succ :: Natural -> Natural
```

# Constructorul de tip `Pair`

## Exemplu de definire TDA

**Exemplul 25.7.**

```
1  data Pair a b    = P a b
2      deriving (Show, Eq)
3
4  pair1            =   P 2 True
5  pair2            =   P 1 pair1
6
7  myFst (P x y)    =   x
8  mySnd (P x y)    =   y
```

# Constructorul de tip `Pair`

## Comentarii

- Constructor de tip: `Pair`
  - polimorfic, binar
  - generează un tip în momentul aplicării asupra 2 tipuri

- Constructor de date: `P`, binar:

```
1  P :: a -> b -> Pair a b
```

# Uniformitatea reprezentării tipurilor

**Exemplul 25.8 (Tipurile de bază).**

```
1  data Integer = ... | -2 | -1 | 0 | 1 | 2 | ...
2
3  data Char = 'a' | 'b' | 'c' | ...
4
5  data [a] = [] | a : [a]
6
7  data (a, b) = (a, b)
```

etc.

# Proprietăţi induse de tipuri

**Definiția 25.9 (Progres).**

O expresie bine-tipată (căreia i se poate asocia un tip):

- este o valoare *sau*
- poate fi redusă.

**Definiția 25.10 (Conservare).**

Evaluarea unei expresii bine-tipate produce o expresie bine-tipată – de obicei, cu acelaşi tip.

# Sinteza

# Sinteza de tip

## Definiție

**Definiția 26.1 (Sinteză de tip — *type inference*).**

Determinarea automată a tipului unei expresii, pe baza unor reguli precise.

- Adnotările explicite de tip, deşi posibile, nenecesare în majoritatea cazurilor
- Dependentă de:
  - componentele expresiei
  - contextul lexical al expresiei
- Reprezentarea tipurilor $\rightarrow$ expresii de tip:
  - constante de tip: tipuri de bază
  - variabile de tip: pot fi legate la orice expresii de tip
  - aplicaţii ale constructorilor de tip pe expresii de tip

# Reguli simplificate de sinteză de tip

## Exemple

- Formă: $\dfrac{\texttt{premisa-1 ... premisa-m}}{\texttt{concluzie-1 ... concluzie-n}}\ \texttt{(nume)}$

---

- Funcţie: $\dfrac{\texttt{Var :: a} \qquad \texttt{Expr :: b}}{\texttt{\textbackslash Var -> Expr :: a -> b}}\ \texttt{(TLambda)}$
- Aplicaţie: $\dfrac{\texttt{Expr1 :: a -> b} \quad \texttt{Expr2 :: a}}{\texttt{(Expr1 Expr2) :: b}}\ \texttt{(TApp)}$
- Operatorul +: $\dfrac{\texttt{Expr1 :: Int} \quad \texttt{Expr2 :: Int}}{\texttt{Expr1 + Expr2 :: Int}}\ \texttt{(T+)}$
- Literali întregi: $\dfrac{}{\texttt{0, 1, 2, ... :: Int}}\ \texttt{(TInt)}$

# Exemple de sinteză de tip

## Transformare de funcție

**Exemplul 26.2.**

```
1  f g = (g 3) + 1
```

# Exemple de sinteză de tip

## Transformare de funcție

### Exemplul 26.2.

```
1  f g = (g 3) + 1
```

$$\frac{\texttt{g :: a} \quad \texttt{(g 3) + 1 :: b}}{\texttt{f :: a -> b}}\ \text{(TLambda)}$$

$$\frac{\texttt{(g 3) :: Int} \quad \texttt{1 :: Int}}{\texttt{(g 3) + 1 :: Int}}\ \text{(T+)}$$

$$\Rightarrow \texttt{b = Int}$$

$$\frac{\texttt{g :: c -> d} \quad \texttt{3 :: c}}{\texttt{(g 3) :: d}}\ \text{(TApp)}$$

$$\Rightarrow \texttt{a = c -> d, c = Int, d = Int}$$

$$\Rightarrow \texttt{f :: (Int -> Int) -> Int}$$

# Exemple de sinteză de tip

## Combinator de punct fix

**Exemplul 26.3.**

```
1  fix f = f (fix f)
```

# Exemple de sinteză de tip

## Combinator de punct fix

**Exemplul 26.3.**

```
1  fix f = f (fix f)
```

$$\frac{\texttt{f :: a} \quad \texttt{f (fix f) :: b}}{\texttt{fix :: a -> b}}\ \text{(TLambda)}$$

$$\frac{\texttt{f :: c -> d} \quad \texttt{(fix f) :: c}}{\texttt{f (fix f) :: d}}\ \text{(TApp)}$$

$$\Rightarrow \texttt{a = c -> d, b = d}$$

$$\frac{\texttt{fix :: e -> g} \quad \texttt{f :: e}}{\texttt{(fix f) :: g}}\ \text{(TApp)}$$

$$\Rightarrow \texttt{a -> b = e -> g, a = e, b = g, c = g}$$

$$\Rightarrow \texttt{f :: (c -> d) -> b = (g -> g) -> g}$$

# Exemple de sinteză de tip

## O funcție ne-tipabilă

**Exemplul 26.4.**

```
1  f x = (x x)
```

# Exemple de sinteză de tip

## O funcție ne-tipabilă

**Exemplul 26.4.**

```
1  f x = (x x)
```

$$\frac{\texttt{x :: a} \qquad \texttt{(x x) :: b}}{\texttt{f :: a -> b}}\ \texttt{(TLambda)}$$

$$\frac{\texttt{x :: c -> d} \qquad \texttt{x :: c}}{\texttt{(x x) :: d}}\ \texttt{(TApp)}$$

Ecuaţia `c -> d = c` nu are soluţie ($\nexists$ tipuri recursive)
$\Rightarrow$ funcţia nu poate fi tipată.

# Unificare

## Definiție

**Definiția 26.5 (Unificare).**

Procesul de identificare a valorilor variabilelor din 2 sau mai multe formule, astfel încât substituirea variabilelor prin valorile asociate să conducă la coincidenţa formulelor.

**Definiția 26.6 (Substituţie).**

O substituție este o mulţime de legări variabilă - valoare.

# Unificare

## Exemplu

**Exemplul 26.7.**

- Pentru expresiile de tip:
  - `t1 = (a, [b])`
  - `t2 = (Int, c)`
- Putem avea substituţiile (variante):
  - `S1 = {a ← Int, b ← Int, c ← [Int]}`
  - `S1 = {a ← Int, c ← [b] }`
- Forme comune pentru `S1` respectiv `S2`:
  - `t1/S1 = t2/S1 = (Int, [Int])`
  - `t1/S2 = t2/S2 = (Int, [b])`

**Definiția 26.8 (*Most general unifier* – MGU).**

Cea mai generală substituţie sub care formulele unifică.
Exemplu: `S2`.

# Unificare

## Condiții

- O variabilă de tip `a` unifică cu o expresie de tip `E` doar dacă:
  - `E` $=$ `a` *sau*
  - `E` $\neq$ `a` şi `E` nu conţine `a` (*occurrence check*).
- 2 constante de tip unifică doar dacă sunt egale.
- 2 aplicaţii de tip unifică doar dacă implică acelaşi constructor de tip şi argumente ce unifică recursiv.

# Tip principal

## Exemplu și definiție

**Exemplul 26.9.**

- Tipurile: `t = (a, [b]) , t' = (Integer, c)`
  MGU: `S = {a ← Integer, c ← [b]}`
  Tipuri mai particulare (instanţe): `(Integer, [Integer])`, `(Integer, [Char])`, etc
- Funcția: `\x -> x`
  Tipuri corecte: `Int -> Int , Bool -> Bool , a -> a`

**Definiția 26.10 (Tip principal al unei expresii).**

Cel mai general tip care descrie complet natura expresiei.
Se obţine prin utilizarea MGU.

# Evaluare

# Evaluare

- Evaluare leneşă: parametri evaluaţi la cerere, cel mult o dată, eventual parţial, în cazul obiectelor structurate
- Transferul parametrilor: *call by need*
- Funcţii nestricte!

**Exemplul 27.1.**

```
1  f (x, y) z = x + x
```

Evaluare:

```
1  f (2 + 3, 3 + 5) (5 + 8)
2
3
4
```

# Evaluare

- Evaluare leneşă: parametri evaluaţi la cerere, cel mult o dată, eventual parţial, în cazul obiectelor structurate
- Transferul parametrilor: *call by need*
- Funcţii nestricte!

**Exemplul 27.1.**

```
1  f (x, y) z = x + x
```

Evaluare:

```
1  f (2 + 3, 3 + 5) (5 + 8)
2  → (2 + 3) + (2 + 3)
3
4
```

# Evaluare

- Evaluare leneşă: parametri evaluaţi la cerere, cel mult o dată, eventual parţial, în cazul obiectelor structurate
- Transferul parametrilor: *call by need*
- Funcţii nestricte!

**Exemplul 27.1.**

```
1  f (x, y) z = x + x
```

Evaluare:

```
1  f (2 + 3, 3 + 5) (5 + 8)
2  → (2 + 3) + (2 + 3)
3  → 5 + 5        reutilizăm rezultatul primei evaluări!
4  → 10             ceilalți parametri nu sunt evaluați
```

# Paşi în aplicarea funcţiilor

## Exemplu

**Exemplul 27.2.**

```
1  front (x:y:zs)   =    x + y
2  front [x]        =    x
3
4  notNil []        =    False
5  notNil (_:_)     =    True
6
7  f m n
8      | notNil xs =    front xs
9      | otherwise =    n
10     where
11         xs       =    [m .. n]
```

# Paşi în aplicarea funcţiilor

## Ordine

1. Pattern matching: evaluarea parametrilor suficient cât să se constate (ne-)potrivirea cu *pattern*-ul
2. Evaluarea gărzilor ( `|` )
3. Evaluarea variabilelor locale, la cerere (`where`, `let`)

# Paşi în aplicarea funcţiilor

## Exemplu – revisited

**Exemplul 27.2 (execuție).**

```
1  f 3 5                                  evaluare pattern
2  ??  notNil xs                            evaluare prima gardă
3  ??      where                            necesar xs → evaluare where
4  ??          xs = [3 .. 5]
5  ??          → 3:[4 .. 5]
6  ??  → notNil (3:[4 .. 5])
7  ??  → True
8  → front xs                                   evaluare valoare gardă
9      where
10         xs = 3:[4 .. 5]                  xs deja calculat
11         → 3:4:[5]
12 → front (3:4:[5])
13 → 3 + 4 → 7
```

# Consecinţe

- Evaluarea parţială a structurilor – liste, tupluri etc.
- Listele sunt, implicit, văzute ca fluxuri!

**Exemplul 27.3.**

```
1  ones              =   1 : ones
2
3  naturalsFrom n    =   n : (naturalsFrom (n + 1))
4  naturals1         =   naturalsFrom 0
5  naturals2         =   0 : (zipWith (+) ones naturals2)
6
7  evenNaturals1     =   filter even naturals1
8  evenNaturals2     =   zipWith (+) naturals1 naturals2
9
10 fibo              =   0 : 1 : (zipWith (+) fibo (tail fibo))
```

# Sfârșitul cursului 6

## Ce am învățat

· Haskell, diferențe față de Scheme, pattern matching și list comprehensions, tipuri în Haskell, construcție de tipuri, sinteză de tip, unificare, evaluare în Haskell.