Programare declarativă

Proprietatea de universalitate a funcției **foldr**[1]

Traian Florin Șerbănuță
Ioana Leuștean

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

[1] bazat pe Graham Hutton, A tutorial on the universality and expressiveness of fold, J. of Functional Programming, 9 (4): 355-372, 1999

foldr și foldl

Definiție

Date fiind o funcție de actualizare a valorii calculate cu un element curent, o valoare inițială, și o listă, calculați valoare obținută prin aplicarea repetată a funcției de actualizare fiecărui element din listă.

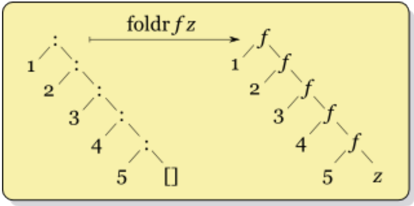
Funcția *foldr*

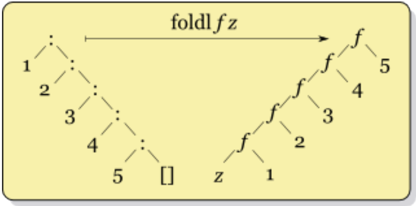
```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f i []     = i
foldr f i (x:xs) = f x (foldr f i xs)
```

Funcția *foldl*

```
foldl :: (b -> a-> b) -> b -> [a] -> b
foldl h i []     = i
foldl h i (x:xs) = foldl h (h i  x) xs
```

foldr și foldl





https://en.wikipedia.org/wiki/Fold_(higher-order_function)

- **foldr** poate fi folosită pe liste infinite!
- **foldl** nu poate fi folosită pe liste infinite!

În continuare, listele sunt considerate finite.

foldr - proprietatea de universalitate

Proprietatea de universalitate

Observație

```haskell
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f i :: [a] -> b
```

Proprietatea de universalitate

Observație

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f i :: [a] -> b
```

Teoremă

Fie g o funcție care procesează liste finite. Atunci

$$\begin{array}{lcl} g\ [] & = & i \\ g\ (x : xs) & = & f\ x\ (g\ xs) \end{array} \quad \Leftrightarrow \quad g = foldr\ f\ i$$

Demonstrație:

$\Rightarrow$ Înlocuind $g = foldr\ f\ i$ se obține definiția lui **foldr**

$\Leftarrow$ Prin inducție dupa lungimea listei.

Proprietatea de universalitate

Observație

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f i :: [a] -> b
```

Teoremă

Fie g o funcție care procesează liste finite. Atunci

$$\begin{array}{lcl} g\ [] & = & i \\ g\ (x : xs) & = & f\ x\ (g\ xs) \end{array} \quad \Leftrightarrow \quad g = foldr\ f\ i$$

Teorema determină condiții necesare și suficiente pentru ca o funcție g care procesează liste să poată fi definită folosind **foldr**.

Generarea funcțiilor cu **foldr**

Compunerea funcțiilor

În definiția lui **foldr**

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

b poate fi tipul unei funcții.

Compunerea funcțiilor

În definiția lui **foldr**

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

b poate fi tipul unei funcții.

```
compose :: [a -> a] -> (a -> a)
compose = foldr (.) id
```

Compunerea funcțiilor

În definiția lui **foldr**

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

b poate fi tipul unei funcții.

```
compose :: [a -> a] -> (a -> a)
compose = foldr (.) id
```

```
Prelude> foldr (.) id [(+1), (^2)] 3
10

-- functia (foldr (.) id [(+1), (^2)]) aplicata lui 3
```

Suma

Definiți o funcție care dată fiind o listă de numere întregi calculează suma elementelor din listă.

Soluție cu foldr

```haskell
sum = foldr (+) 0
```

Suma

Definiți o funcție care dată fiind o listă de numere întregi calculează suma elementelor din listă.

Soluție cu **foldr**

```
sum = foldr (+) 0
```

În definiția de mai sus elementele sunt procesate de la dreapta la stânga:
$\mathbf{sum}[x_1, \ldots, x_n] = (x_1 + (x_2 + \ldots (x_n + 0) \ldots)$

Problemă

Scrieți o definiție a sumei folosind **foldr** astfel încât elementele să fie procesate de la stânga la dreapta.

Suma

sum cu acumulator

```haskell
sum :: [Int] -> Int
sum  xs = suml xs 0
          where
             suml [] n =n
             suml (x:xs) n = suml xs (n+x)
```

Suma

sum cu acumulator

```
sum :: [Int] -> Int
sum  xs = suml xs 0
          where
              suml [] n =n
              suml (x:xs) n = suml xs (n+x)
```

În definiția de mai sus elementele sunt procesate de la stânga la dreapta:
suml $[x_1, \ldots, x_n]$ $0 = (\ldots(0 + x_1) + x_2) + \ldots x_n)$

Suma

sum cu acumulator

```
sum :: [Int] -> Int
sum  xs = suml xs 0
          where
              suml [] n =n
              suml (x:xs) n = suml xs (n+x)
```

În definiția de mai sus elementele sunt procesate de la stânga la dreapta:
suml $[x_1, \ldots, x_n]\ 0 = (\ldots(0 + x_1) + x_2) + \ldots x_n)$

Definim suml cu **foldr**

- Obervăm că

```
suml :: [Int] -> (Int -> Int)
```

- Definim suml cu **foldr** aplicând proprietatea de universalitate.

Definirea suml cu foldr

Proprietatea de universalitate

$$\begin{array}{lcl} g\ [] & = & i \\ g\ (x : xs) & = & f\ x\ (g\ xs) \end{array} \quad \Leftrightarrow \quad g = foldr\ f\ i$$

Definirea suml cu foldr

Proprietatea de universalitate

$$\begin{array}{lcl} g\ [] & = & i \\ g\ (x : xs) & = & f\ x\ (g\ xs) \end{array} \quad \Leftrightarrow \quad g = foldr\ f\ i$$

Observăm că

```
suml [] = id    -- suml [] n = n
```

Definirea suml cu foldr

Proprietatea de universalitate

$$\begin{array}{lcl} g\ [] & = & i \\ g\ (x : xs) & = & f\ x\ (g\ xs) \end{array} \quad \Leftrightarrow \quad g = foldr\ f\ i$$

Observăm că

```
suml [] = id    -- suml [] n = n
```

Vrem să găsim *f* astfel încât

$$suml\ (x : xs) = f\ x\ (suml\ xs)$$

deoarece, din proprietatea de universalitate, va rezulta că

$$suml = foldr\ f\ id$$

Definirea suml cu foldr

```
suml :: [Int] -> (Int -> Int)
```

$$
\begin{array}{lcl}
suml\ (x : xs) & = & f\ x\ (suml\ xs) \\
suml\ (x : xs)\ n & = & f\ x\ (suml\ xs)\ n \\
suml\ xs\ (n + x) & = & f\ x\ (suml\ xs)\ n
\end{array}
$$

Definirea suml cu foldr

```haskell
suml :: [Int] -> (Int -> Int)
```

$$
\begin{aligned}
suml\ (x : xs) &= f\ x\ (suml\ xs) \\
suml\ (x : xs)\ n &= f\ x\ (suml\ xs)\ n \\
suml\ xs\ (n + x) &= f\ x\ (suml\ xs)\ n
\end{aligned}
$$

Notăm $u = suml\ xs$ și obținem

$$
\begin{aligned}
u\ (n + x) &= f\ x\ u\ n \\
u\ (n + x) &= (f\ x\ u)\ n
\end{aligned}
$$

Definirea suml cu foldr

```
suml :: [Int] -> (Int -> Int)
```

$$
\begin{aligned}
suml\ (x : xs) &= f\ x\ (suml\ xs) \\
suml\ (x : xs)\ n &= f\ x\ (suml\ xs)\ n \\
suml\ xs\ (n + x) &= f\ x\ (suml\ xs)\ n
\end{aligned}
$$

Notăm $u = suml\ xs$ și obținem

$$
\begin{aligned}
u\ (n + x) &= f\ x\ u\ n \\
u\ (n + x) &= (f\ x\ u)\ n
\end{aligned}
$$

Rezultă că $f = \lambda xu.(\lambda n.u(n + x))$

Soluție

```
f = \ x u -> \ n -> u (n+x)
suml = foldr (\ x u -> \ n -> u (n+x)) id
```

Definirea sum cu foldr

```haskell
sum :: [Int] -> Int

sum xs = foldr (\ x u -> \ n -> u (n+x)) id xs 0

-- sum xs = suml xs 0
```

Definirea sum cu foldr

```haskell
sum :: [Int] -> Int

sum xs = foldr (\ x u -> \ n -> u (n+x)) id xs 0

-- sum xs = suml xs 0
```

```
Prelude> sum xs = foldr (\ x u -> \ n -> u (n+x)) id xs 0
Prelude> sum [1,2,3]
6
```

foldl

Definiție

Funcția *foldl*

```
foldl :: (b -> a-> b) -> b -> [a] -> b
foldl h i []     =  i
foldl h i (x:xs) = foldl h (h i  x) xs
```

foldl

Definiție

Funcția *foldl*

```haskell
foldl :: (b -> a-> b) -> b -> [a] -> b
foldl h i []     =  i
foldl h i (x:xs) = foldl h (h i  x) xs
```

```haskell
foldl h i xs = foldl' h xs i
               where
                 foldl' h [] i = i
                 foldl' h (x:xs) i = foldl' h xs (h i x)
```

foldl

Definiție

Funcția *foldl*

```haskell
foldl :: (b -> a-> b) -> b -> [a] -> b
foldl h i []       =  i
foldl h i (x:xs) = foldl h (h i  x) xs
```

```haskell
foldl h i xs = foldl' h xs i
               where
                 foldl' h [] i = i
                 foldl' h (x:xs) i = foldl' h xs (h i x)
```

```haskell
foldl' :: (b -> a -> b) -> [a] -> b -> b
foldl' h :: [a] -> (b ->b)
foldl' h xs :: b -> b
```

foldl' cu **foldr**

Observăm că

```haskell
foldl' h [] = id      -- suml [] n = n
```

foldl' cu **foldr**

Observăm că

```haskell
foldl' h [] = id     -- suml [] n = n
```

Vrem să găsim f astfel încât

$$foldl'\ h\ (x : xs) = f\ x\ (foldl'\ h\ xs)$$

deoarece, din proprietatea de universalitate, va rezulta că

$$foldl'\ h = foldr\ f\ id$$

foldl cu **foldr**

Soluție

```haskell
h :: b -> a -> b

foldl' h = foldr f id

f = \ x u -> \ y -> u (h y x)


foldl h i xs = foldl' h xs i

foldl h i xs = foldr (\ x u -> \ y -> u (h y x)) id xs i
```

foldl cu **foldr**

```
Prelude> let myfoldl h i xs =
                   foldr (\x u -> \y -> u (h y x)) id xs i

Prelude> myfoldl (+) 0 [1,2,3]
6
```

foldl cu **foldr**

```
Prelude> let myfoldl h i xs =
                    foldr (\x u -> \y -> u (h y x)) id xs i

Prelude> myfoldl (+) 0 [1,2,3]
6


Prelude> let sing = (:[])

Prelude> take 3 (foldr (++) [] (map sing [1..]))
[1,2,3]

Prelude> take 3 (myfoldl (++) [] (map sing [1..]))
```

Interrupted.