Muzyka algorytmiczna, wykład 2

Maciej Grześkowiak

10 marca 2021

Wyrażenia Lambda

Lambda

Jeden argument

```
square1 :: Integer -> Integer
square1 x = x^2

square2 :: Integer -> Integer
square2 = \ x -> x^2
```

Dwa argumenty

```
m1 :: Integer -> Integer -> Integer
m1 = \ x -> \ y -> x*y

m2 :: Integer -> Integer -> Integer
m2 = \ x y -> x*y ---> skrocony zapis m1
```

Lambda

```
(#) = \x -> \y ->   x # y
(#) = \x  y ->   x # y
```

Przykład

```
(*) = \x -> \y ->   x * y
(+) = \x -> \y ->   x + y
```

Lambda

```
(x #) = \y -> x # y
(# y) = \x -> x # y
```

Przykład

```
( 2 * ) = \y -> 2 * y
(+ 5) = \x x + 5
```

Listy

Listy

```
l1 = [1,2,3]     -- lista trzech elementow
l2 = [1..10]     -- lista liczb z przedzialu [1,10]
l3 = [1,3..10]   -- lista liczb nieparzystych ...
```

```
l4 = 0 : [ ] -- konstruktor listy
l5 = [1..5] ++ [6..10] - konkatenacja list
```

Listy, funkcje `head`, `tail`

```
1 tl (x:xs) = xs
2 tl []     = []
3
4 hd (x:xs) = x
5 hd []     = error "hd []"
```

Listy, funkcja take

```haskell
take :: Int -> [a] -> [a]
take 0 _      = []
take n (x:xs) = x : take (n-1) xs
take _ []     = []
```

Symbol podkreślenia występujący we wzorcu oznacza argument, który nie będzie używany po prawej stronie równania definiującego funkcję

Listy, funkcja cycle, zip, unzip, zipWith

```
cycle [] = []
cycle xs = xs ++ cycle xs
```

```
zip   :: [a] -> [b] -> [(a,b)]
unzip :: [(a, b)] -> ([a], [b])
```

Przykład:

```
zip [1..] (cycle [7,5]) ---> [(1,7), (2,5), ...]
zipWith (+) [3,4,5] [6,6,8,9]
```

Listy, przykład

n-ty element ciągu Fibonacciego

```
fib :: Int -> Int
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

```
fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

Listy, funkcja `filter`

```
evens = filter even [1..]
```

```
zip    :: [a] -> [b] -> [(a,b)]
unzip  :: [(a, b)] -> ([a], [b])
```

Przykład:

```
zip [1..] (cycle [7,5]) ⟶ [(1,7), (2,5), ...]
zipWith (+) [3,4,5] [6,6,8,9]
```

Listy, funkcja map

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

```
map (*10) [3,5,6] ---> [30,50,60]
map (mod 6) [10,89,5] ---> [6,6,1]
map even [10,89,5] ---> [True,False,False]
```

Przykład:

```
wiel2 = 1 : map (2 *) wiel2
```

Wyrażenie if

Wyrażenie if ... then ...else

```
if e then e' else e''
```

Nie ma konstrukcji if ... then!!!!!

```
sgn x = if x < 0 then -1
        else if x == 0 then 0
             else 1
```

Wyrażenie let ... in

Wyrażenie let... in

Wyrażenie let pozwala nam na użycie lokalnych definicji pomocniczych dla obliczenia jakiegoś wyrażenia:

```
let
    x = 1
    y = 2
in x + y
```

```
f xs = let
         len [] = 0
         len (x:xs) = 1 + len xs
       in len xs
```

Wyrażenie let ... in

```
1 volume r = let a = 4 / 3
2                  cube = r ^ 3
3              in a * pi * cube
```

Klauzula where

Klauzula where

```
volume r = a * pi * cube
           where a = 4 / 3
           cube = r ^ 3
```

Definicje po let przesłaniaja te po where:

```
fun x = let y = x + 1
        in y
        where y = x + 2
```

Wyrażenie case ... of

Wyrażenie case .. of

```
fun 1 = 2
fun 2 = 3
fun _ = -1
```

```
fun n = case n of
            1 -> 2
            2 -> 3
            _ -> -1
```

```
fun n = case n < 0 of
          True -> "ujemna"
          False -> "dodatnia"
```

Strażnicy

Wyrażenie let ... in

```
sgn x | x < 0 = -1
      | x == 0 = 0
      | x > 0 = 1
```

```
sgn x | x < 0 = -1
      | x == 0 = 0
      | otherwise = 1
```

Przykład

```
factor :: Integer->[Integer]
factor n | n == 1 = []
         | otherwise = p : factor (div n p)
             where p = ld n
```