

Programowanie Funkcyjne

Rafał Włodarczyk

INA 4, 2025

Contents

1 Haskell 27.05	1
1.1 Monady	1

1 Haskell 27.05

1.1 Monady

Rozważmy następujące przykłady:

```
> [1..5]>>= (\x -> [x,x])  
[1,1,2,2,3,3,4,4,5,5]
```

```
> [1..5]>>= (\x -> if even x then [x,x] else [])  
[2,2,4,4]
```

bind powoduje spłaszczenie, może zmieniać długość listy

```
> fmap (\x -> if even x then [x,x] else []) [1..5]  
[[],[2,2],[],[4,4],[]]
```

```
> concat $ fmap (\x -> if even x then [x,x] else []) [1..5]  
[2,2,4,4]
```

```
msublists [] = [[]]  
msublists (x:xs) = do  
  t <- [False, True]  
  y <- msublists xs  
  return (if t then x:y else y)
```

Konstruktor danych:

$$F_M(X) = X \times M \text{data Writer m a} = W(a, m) \quad (1)$$

$$\text{Funktor:} \quad (2)$$

$$F_M(f)(x, m) = (fx, m) \quad \text{fmap } f(W(x, m)) = W(fx, m) \quad (3)$$

$$\text{M-monoid} \quad (4)$$

$$(f, m) < * > (x, n) = (fx, m * n) \quad (M, *, e) \quad (5)$$

$$\text{pure } x = (x, e) \quad (6)$$

Rozważmy bind:

```
\text{>= :: Ma -> (a -> Mb) -> Mb}
((x,m) >= f) = let (y,n) = f x in
                (y m*n)
```

```
(mx >> my) = mx >= (\lambda _ -> my)
(x,m) >> (y,n) = (y, m * n)
```

```
tell m = ((),m) :: W_mon
```

Przykład z korporacji Przykład z korporacji

```
[(k_1, [l_1^1, \dots l_n^1]), (k_2, [l_1^2, \dots l_n^2]), \dots, (k_n, [l_1^n, \dots l_n^n])]
l_1^1, \dots l_n^1 < n
```