Programowanie funkcyjne

Wojciech Typer

W katalogu przykłady pojawiać się będą przykłady z wykładów.

Funkcja lambda

$$\exp = (\lambda a : \text{Typ} \to (a \to a))$$

$$\exp(\text{Int}) :: \text{Int} \to \text{Int}$$

$$\exp(\text{Double}) :: \text{Double} \to \text{Double}$$

Matematyczny zapis inkrementacji

$$\text{inc}\ x = x + 1$$

$$>: t \quad \text{inc}$$

$$\text{inc} :: \text{hum}(a) \implies (a \to a)$$

$$\exp = (\forall a : \text{hum} \to (a \to a))$$

$$\exp(\text{Int}) :: \text{Int} \to \text{Int}$$

$$\exp(\text{Bool}) \leftarrow (\text{błąd})$$

Typy w Haskellu

- Typy proste:
  - `Int`
  - `Double`
  - `Char`
  - `Bool`
- Typy złożone:
  - Listy
  - Krotki
  - Funkcje

Funkcja collatz'a

Funkcja collatz'a jest nierozstrzygnięty dotychczas problem o wyjątkowo prostym jak wiele innych problemów teorii liczb sformułowaniu.

$$c_{n+1} = \begin{cases} \frac{1}{2}c_n & \text{gdy } c_n \text{ jest parzysta} \\ 3c_n + 1 & \text{gdy } c_n \text{ jest nieparzysta} \end{cases}$$

lub

$$c_{n+1} = \frac{1}{2}c_n - \frac{1}{4}(5c_n + 2)((-1)^{c_n} - 1).$$

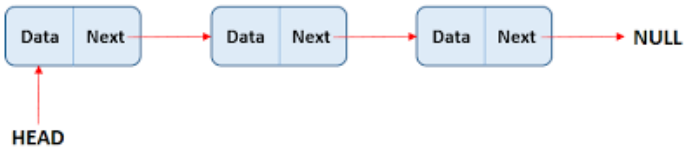
przykłady/W2 → funkcja collatz'a

Listy w Haskellu

Definicja list w Haskellu:

$$[a] = \text{lista elementów } a$$

$$= \{[a_1, ..., a_k] : a_1, ..., a_k, k \in \mathbb{N}\}$$



Rysunek 1: Przykładowa lista w Haskellu

Struktura listy:

$$x_0 : [x_1, x_2, ..., x_k] = [x_0, x_1, ..., x_k]$$

Lista pusta:

$$[] \quad \text{(lista pusta)}$$

Przykłady list:

$$[1, 2, 3] \quad \text{można zapisać jako} \quad 1 : 2 : 3 : []$$

Dodawanie (konkatenacja) dwóch list:

$$[1, 2, 3] + [4, 5] = [1, 2, 3, 4, 5]$$

Prelude

Prelude to standardowa biblioteka Haskella

- Dodawanie elementu na początku listy

```
>:t (1:[2,3])
(1:[2,3]) :: Num a => [a]
```

- Konkatenacja list

```
>:t [1,2]++[3,4]
[1,2]++[3,4] :: Num a => [a]
```

0.0.1 Podstawowe funkcje operujące na listach

- `length :: [a] → Int`
  - `length [] = 0`
  - `length (x:xs) = 1 + length xs`
- `head :: [a] → a`
  zwraca pierwszy element listy
  - `head (x:xs) = x`
  - `head [] = error "empty list"`
- `tail :: [a] → [a]`
  zwraca listę bez pierwszego elementu
  - `tail (x:xs) = xs`
  - `tail [] = error "empty list"`
- `last :: [a] → a`
  zwraca ostatni element listy
  - `last [x] = x`
  - `last (x:xs) = last xs`
  - `last [] = error "empty list"`
- `filter :: (a → Bool) → [a] → [a]`
  - `filter p [] = []`
  - `filter p (x:xs) = if p x then x : filter p xs else filter p xs`
  - `filter` $(\lambda n \to n > 0)$ `[-1,2,-3,4] = [2,4]`
  - `filter even [1..10] = [2,4,6,8,10]`

  Jak zdefiniować funkcję **filter**:

```
filter p [] = []
filter p (x:xs)
    | p x = x : filter p xs
    | otherwise = filter p xs
```

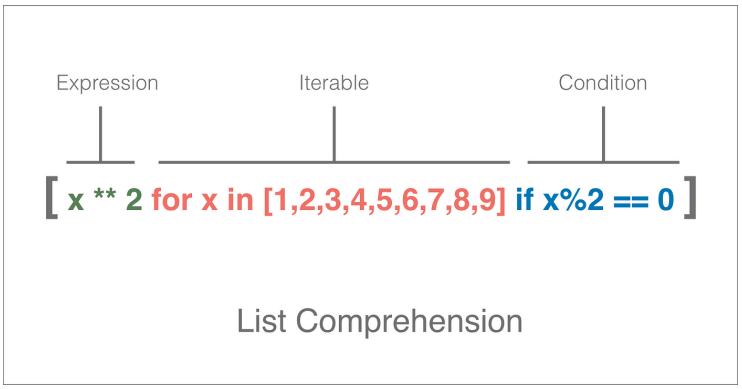
- `map :: (a → b) → [a] → [b]`
  zwraca listę, która powstaje poprzez zastosowanie funkcji do każdego elementu listy
  - `map f [] = []`
  - `map f (x:xs) = f x : map f xs`

- `map (`$\lambda n \to n * n$`) [1,2,3] = [1,4,9]`
- `map (`$\lambda n \to n^3$`) [1..10] = [1,8,27,64,125,216,343,512,729,1000]`

gdzie **[1..10]** to skrót od **[1,2,3,4,5,6,7,8,9,10]**.

List comprehension

$[fx_1x_2x_3|x_1 \leftarrow xs, x_2 \leftarrow ys, x_3 \leftarrow zs]$
$[fx_1x_2x_3|x_1 \leftarrow xs, x_2 \leftarrow, x_1 < x_2, x_3 \leftarrow zs]$



Przykład: trójki pitagorejskie

$[(x, y, z)|z \leftarrow [1..100], y \leftarrow [1..z], x \leftarrow [1..y], x^2 + y^2 + z^2,$ gcd x y $== 1]$