Programowanie funkcyjne - laboratoria

Wojciech Typer

zadanie 1

power x y = power y^x

p2 = power 4 $\rightarrow$ power 4 y = y^4

p3 = power 3

(p2 . p3) 2 = p2(p3 2) = p2 8 = 8^4 = 4096

p2 :: Int -> Int

p3 :: Int -> Int

(p2 . p3) :: Int -> Int

Wyrażenia lambda:

power = $\lambda x \rightarrow \lambda y \rightarrow y^x$

p2 = $\lambda y \rightarrow y^4$

p3 = $\lambda y \rightarrow y^3$

zadanie 4

plus = $\lambda xy \rightarrow x + y$

multi = $\lambda xy \rightarrow x * y$

zadanie 5

haskell:

$\lambda x \rightarrow 1 + x * (x + 1)$

python:

f = lambda x: 1 + x * (x + 1)

zadanie 6

Ustalmy zbiory A, B, C. Niech

$$\text{curry} : C^{B\times A} \rightarrow (C^B)^A$$

będzie funkcją zadaną wzorem:

$$\mathrm{curry}(\varphi) = \lambda a \in A \to (\lambda b \in B \to \varphi(b, a)).$$

oraz niech

$$\mathrm{uncurry} : (C^B)^A \to C^{B\times A}$$

będzie zadana wzorem:

$$\mathrm{uncurry}(\psi)(b, a) = (\psi(a))(b).$$

1. Pokaż, że $\mathrm{curry} \circ \mathrm{uncurry} = \mathrm{id}_{(C^B)^A}$ oraz $\mathrm{uncurry} \circ \mathrm{curry} = \mathrm{id}_{C^{B\times A}}$.
2. Wywnioskuj z tego, że $|(C^B)^A| = |C^{B\times A}|$. Przypomnij sobie dowód tego twierdzenia, który poznałeś na pierwszym semestrze studiów.
3. Spróbuj zdefiniować w języku Haskell odpowiedniki funkcji `curry` i `uncurry`.

---

1. Pokażemy, że $\mathrm{curry} \circ \mathrm{uncurry} = \mathrm{id}_{(C^B)^A}$ oraz $\mathrm{uncurry} \circ \mathrm{curry} = \mathrm{id}_{C^{B\times A}}$.
   - $\mathrm{curry} \circ \mathrm{uncurry}$

     $$(\mathrm{curry} \circ \mathrm{uncurry})(\psi) = \mathrm{curry}(\mathrm{uncurry}(\psi)) = \mathrm{curry}(\lambda a \in A \to (\lambda b \in B \to \psi(a)(b))) = \lambda a \in A \to (\lambda b \in$$

   - $\mathrm{uncurry} \circ \mathrm{curry}$

     $$(\mathrm{uncurry} \circ \mathrm{curry})(\varphi) = \mathrm{uncurry}(\mathrm{curry}(\varphi)) = \mathrm{uncurry}(\lambda a \in A \to (\lambda b \in B \to \varphi(b, a))) = \lambda b \in B \to (\lambda a$$

2. Możemy pokazać że `curry` i `uncurry` są iniekcjami niewprost, nakładając odpowiednio przeciwne funkcje na obie strony równości:
   - Załóżmy, że $\mathrm{curry}(\varphi_1) = \mathrm{curry}(\varphi_2)$. Wtedy:

     $$\mathrm{curry}(\varphi_1)(a)(b) = \mathrm{curry}(\varphi_2)(a)(b) \quad \Rightarrow \quad \varphi_1(b, a) = \varphi_2(b, a) \quad \Rightarrow \quad \varphi_1 = \varphi_2.$$

   - Załóżmy, że $\mathrm{uncurry}(\psi_1) = \mathrm{uncurry}(\psi_2)$. Wtedy:

     $$\mathrm{uncurry}(\psi_1)(b, a) = \mathrm{uncurry}(\psi_2)(b, a) \quad \Rightarrow \quad \psi_1(a)(b) = \psi_2(a)(b) \quad \Rightarrow \quad \psi_1 = \psi_2.$$

   A więc istnieje biekcja między $(C^B)^A$ i $C^{B\times A}$, co oznacza, że te zbiory mają taką samą moc.

3. W języku Haskell funkcje `curry` i `uncurry` można zdefiniować następująco:

```
curry :: ((b, a) -> c) -> a -> b -> c
curry f x y = f (y, x)
```

```
uncurry :: (a -> b -> c) -> (a, b) -> c
uncurry f (x, y) = f x y
```

zadanie 13

- Funkcja phi Eulera:

  ```
  phi :: Int -> Int
  phi n = length [x | x <- [1..n - 1], gcd x n == 1]
  ```

  tworzy tablicę liczb od 1 do n-1 i następnie filtruje te, które są względnie pierwsze z n. length zwraca długość tej tablicy, co można utożsamiać z mocą zbioru.

- Funkcja $\sum_{k|n} \phi(k)$:

  ```
  phi2 :: Int -> Int
  phi2 n = sum [phi x | x <- [1..n], n `mod` x == 0]
  ```

  tworzy tablicę liczb od 1 do n, filtruje te, które są dzielnikami liczby n i liczy sumę funkcji phi dla tych liczb. Zauważmy, że $\sum_{k|n} \phi(k) = n$, ponieważ każdą liczbę można zapisać jako sumę liczb względnie pierwszych w jej dzielnikach.

zadanie 14

Liczba doskonała: $n = \sum\{d : 1 \leq d < n,\ d \mid n\}$

Na początku zdefiniujmy funkcję, która sprawdza czy dana liczba jest doskonała:

```
isPerfect :: Int -> Bool
isPerfect n = n == sum [k | k <- [1..n-1], n `mod` k == 0]
```

Następnie zdefiniujmy funkcję, która zwróci wszystkie liczby doskonałe mniejsze od n:

```
allPerfect :: Int -> [Int]
allPerfect n = [k | k <- [1..n], isPerfect k]
```

Dla n = 10000 otrzymamy: [6, 28, 496, 8128]

zadanie 15

Na początku zdefiniujmy funkcję, która zwraca sumę dzielników:

```
sumOfDivisors :: Int -> Int
sumOfDivisors n = sum [k | k <- [1..n-1], n `mod` k == 0]
```

Następnie zdefiniujmy funkcję, która sprawdza, czy 2 liczby są zaprzyjaźnione:

```
areSociable :: Int -> Int -> Bool
areSociable a b = sumOfDivisors a == b && sumOfDivisors b == a && a /= b
```

Na koniec, zdefiniujmy funkcję, która zwróci wszystkie pary liczb zaprzyjaźnionych, mniejsze od podanego limitu:

```
socialPairs :: Int -> [(Int, Int)]
socialPairs limit =
    [(a, b) | a <- [1..limit],
              let b = sumOfDivisors a,
              areSociable a b && a < b && b < limit]
```

Dla limit $= 10^5$ otrzymamy: [(220,284),(1184,1210),(2620,2924),(5020,5564),(6232,6368),(10744,10856), (12285,14595),(17296,18416)...]

zadanie 16

Definiujemy funkcje: $dcp(n) = \frac{1}{n^2}|\{(k,l) \in \{1,2,...,n\} : gcd(k,l) = 1\}|$

- Implementacja funkcji za pomocą list comprehension:

```
dcp1 :: Int -> Double
dcp1 n = fromIntegral a / fromIntegral b
    where a = length [(k, l) | k <- [1..n], l <- [1..n], gcd k l == 1]
          b = n^2
```

- Implementacja funkcji rekurencyjnej

```
dcp' :: Int -> Double
dcp' n = fromIntegral (countCoprimes n n 1 1) / fromIntegral (n ^ 2)

countCoprimes :: Int -> Int -> Int -> Int -> Int
countCoprimes n m i j
    | i > m = 0 --koniec
    | j > m = countCoprimes n m (i + 1) 1 --przejscie do nowego weirsza
    | gcd i j == 1 = 1 + countCoprimes n m i (j + 1) --liczmy wzglednie pierwsza
    | otherwise = countCoprimes n m i (j + 1) --nie jest wzglednie pierwsza
```

- $lim_{n\to\infty} dcp(n)$

  kolejne wartości:

```
[0.6087,0.611575,0.6088333333333333,0.60846875,0.608924,
0.6083305555555556,0.608234693877551,
0.6085921875,0.6082111111111111,0.608383,0.6084586776859504,
0.6080354166666667,0.6080988165680473,
0.6082525510204082,0.6081613333333333,0.607993359375,0.6083678200692042,
0.6080601851851852,0.6080096952908587,
0.60829375,0.6080823129251701,0.6079518595041322,0.6081570888468809,
0.6081019097222222,0.6079608,0.6081087278106508]
```

  Widzimy, że wraz ze wzrostem n, wartość dcp(n) zbiega do ≈ 0.608.

  Możemy wysnuć hipotezę, że $lim_{n\to\infty} dcp(n) = \frac{6}{\pi^2}$.