

Programowanie Funkcyjne

WPPT

Lista zadań

Jacek Cichoń, WPPT, PWi, 2022/23

Zadania oznaczone * są nieco trudniejsze od zadań bez gwiazdki. Zadania oznaczone ** są jeszcze trudniejsze.

1 Wstęp

1.1 Teoria

Iloczynem kartezjańskim funkcji $f : X \rightarrow Y$ i $g : A \rightarrow B$ nazywamy funkcję $f_1 \times f_2 : X \times A \rightarrow Y \times B$ określoną wzorem

$$(f_1 \times f_2)((x, y)) = (f_1(x), f_2(y)) .$$

Jeśli $f : X \rightarrow A$ oraz $g : X \rightarrow B$, to przez (f, g) oznaczamy funkcję $(f, g) : X \rightarrow A \times B$ określoną wzorem

$$(f, g)(x) = (f(x), g(x)) .$$

Zadanie 1 — Pokaż, że

1. $(\alpha, \beta) \circ f = (\alpha \circ f, \beta \circ g)$
2. $(\alpha \times \beta) \circ (f \times g) = (\alpha \circ f) \times (\beta \circ g)$ dla dowolnych funkcji α, β, f, g dla których złożenia $\alpha \circ f$ oraz β są dobrze określone.
3. Pokaż, że $(f, g) = (f \times g) \circ \Delta_X$ gdzie $\Delta_X : X \rightarrow X \times X : x \rightarrow (x, x)$ oraz $f, g : X \rightarrow Y$.

Zadanie 2 — Niech $f : X \rightarrow X$ i $g : X \times X \rightarrow X$. Przedstaw następujące funkcje jako złożenie funkcji $f, g, id_X : X \rightarrow X : x \rightarrow x, \Delta_X : X \rightarrow X \times X : x \rightarrow (x, x), fst_X : X \times X \rightarrow X : (x, y) \rightarrow x, snd_X : X \times X \rightarrow X : (x, y) \rightarrow y$:

1. $h_0(x) = f(f(x))$,
2. $h_1(x) = g(x, x)$
3. $h_2(x) = g(f(x), x)$
4. $f_3(x) = g(g(x, x), g(x, x))$

Dla ustalonych zbiorów A, B, C definiujemy $\text{curry}_{A,B,C} : C^{A \times B} \rightarrow (C^B)^A$ wzorem

$$\text{curry}_{A,B,C}(f)(a) = \lambda y. f(a, y).$$

* **Zadanie 3** — Niech $f : A \rightarrow A'$. Pokaż, że

$$(\text{curry}_{A',B,C}(\phi)) \circ f = \text{curry}_{A,B,C}(\phi \circ (f \times id))$$

dla wszystkich $\phi : A' \times B \rightarrow C$

* **Zadanie 4** — Niech $f : C \rightarrow D$. Pokaż, że

$$\text{curry}_{A,B,D}(f \circ \phi) = (\lambda x). (\lambda y). f((\text{curry}_{A,B,C}(\phi)(x))(y))$$

dla wszystkich $\phi : C \rightarrow D$.

Zadanie 5 — Niech $pr : (A \times B) \times C \rightarrow (A \times B \times C : ((x, y), z) \rightarrow (x, y, z))$ oraz

$$curry3\ f = curry(curry(f \circ pr)) .$$

Pokaż, że $curry3 : D^{A \times B \times C} \rightarrow D^{C^{B^A}}$.

Zadanie 6 — Funkcją Ackermana nazywamy funkcję $A : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ zdefiniowaną za pomocą wzoru

$$A(m, n) = \begin{cases} n + 1 & : m = 0 \\ A(m - 1, 1) & : m > 0 \wedge n = 0 \\ A(m - 1, A(m, n - 1)) & : \text{w pozostałych przypadkach} \end{cases}$$

- * 1. Pokaż, że funkcja Ackermana jest poprawnie określona.
- 2. Pokaż, że $A(1, y) = y + 2$, $A(2, y) = 2y + 3$, $A(3, y) = 2^{y+3} - 3$
- 3. Pokaż, że

$$A(4, y) = \underbrace{2^{2^{\cdot^{\cdot^2}}}}_{y+3} - 3$$

1.2 Praktyka

Zadanie 7 — Uprość następujące wyrażania:

1. $(\lambda x.(x\ y))(\lambda x.x)$.
2. $(\lambda x.(x\ y))(\lambda z.z)$
3. $((\lambda x.((\lambda y.(x\ y)))x))(\lambda z.w)$

Zadanie 8 — Rozważ my funkcję

$$f(x, y) = (\lambda x.(\lambda y.x + 2 * y))(x * y).$$

1. Zapisz tę funkcję w językach JavaScript (arrow notation), Python oraz w języku Haskell.
2. Zastosuj alfa transformację i beta redukcję do uproszczenia funkcji f .

Zadanie 9 — Rozważmy następujące definicje

```
f(x) = x * x
g(y) = f (f y)
h(x) = g \circ g
Uprość funkcję h.
```

Zadanie 10 — Rozważmy następującą funkcję

```
function f(x) {
  let y = Math.sin(x);
  return y*y + y + x;
}
```

Wyliminuj zmienną y z tego kodu, bez pogarszania jego efektywności.

Uprość funkcję h .

Zadanie 11 — Oprogramuj w językach JavaScript i Python rekurencyjne wersje funkcji, które na wejściu mają podaną listę liczb rzeczywistych $[x_1, \dots, x_n]$ i jako wynik zwracają:

1. $\sum_{i=1}^n x_i$
2. $\prod_{i=1}^n x_i$
3. $\min\{x_i : i = 1, \dots, n\}$
4. $\max\{x_i : i = 1, \dots, n\}$

Zadanie 12 — Wyliminuj z następującego kodu pętlę (zastąp ją rekursją i, oczywiście, pozbadź się zmiennej pomocniczej s)

```
function sum_of_squares(n) {
  let s = 0;
  for (let i = 0; i <= n; i++) {
    s = s + i * i
  }
  return s
}
```

Zadanie 13 — Oprogramuj funkcję Ackermana w języku JS lub Python i wyznacz jej wartości dla małych wartości m i n .

1. Pierwsze rozwiązanie oprzyj bezpośrednio na rekurencyjnej definicji.
2. Drugie rozwiązanie oprzyj na metodzie "memoizacji".

2 Wprowadzenie do Haskell'a

Zadanie 14 — Zrób wszystkie zadania z książki Real World Haskell po rozdziale pierwszym.

Zadanie 15 — Oblicz w GHCi wartości wyrażeń $2 \wedge 3 \wedge 2$, $(2 \wedge 3) \wedge 2$ i $2 \wedge (2 \wedge 3)$. Dowiedz się jaka jest łączność operatora \wedge za pomocą polecenia `:i` (\wedge).

Zadanie 16 — Funkcją Eulera ϕ nazywamy funkcję określoną wzorem

$$\phi(n) = \text{card}(\{k \leq n : \text{gcd}(k, n) = 1\}) .$$

o dziedzinie \mathbb{N}^+ .

1. Oprogramuj funkcję ϕ (funkcja `gcd` jest w bibliotece `Prelude`)
2. Napisz funkcję, która dla danej liczby naturalnej n wyznacza liczbę $\sum_{k|n} \phi(k)$.

Zadanie 17 — Trójkę liczb naturalnych (a, b, c) nazywamy właściwą trójką pitagorską jeśli $a^2 = b^2 + c^2$ oraz $\text{gcd}(b, c) = 1$. Wyznacz wszystkie właściwe trójki pitagorejskie takie, że $a \leq 200$.

Zadanie 18 — Zaimplementuj na kilka sposobów funkcję służącą do wyznaczania liczb Fibbonacciego: rekurencyjnie, rekurencyjnie za pomocą wzorców.

Zadanie 19 — Zaimplementuj funkcję $\binom{n}{k}$. Nie stosuj tożsamości $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ - jest to kosztowne rozwiązanie; zastosuj wzór rekurencyjny na $\binom{n+1}{k+1}$.

Zadanie 20 — Liczbę naturalną n nazywamy doskonałą jeśli $n = \sum \{d : 1 \leq d < n \wedge d|n\}$. Np. 6 jest liczbą doskonałą, bo $6 = 1 + 2 + 3$. Wyznacz wszystkie liczby doskonałe mniejsze od 10000.

Zadanie 21 — Zapisz operacje binarne $(+)$, $(*)$ za pomocą lambda wyrażeń.

Zadanie 22 — Niech `ff` = $(2 \wedge)$ oraz `gg` = $(\wedge 2)$. Podaj interpretację tych funkcji.

Zadanie 23 — Sprawdź wartości wyrażeń

```
map (^ 2) [1..5]
```

oraz

```
map (2 ^) [1..5]
```

i wyjaśnij otrzymane wyniki.

Zadanie 24 — Dowiedz się jak można przekonwertować elementy typu `Int` oraz `Integer` na typy `Float` i `Double`. Dowiedz się jaki jest format funkcji typu `round` z `Float` to `Int`.

Zadanie 25 — Oszacuj złożoność obliczeniową następującej (kiepskiej) funkcji służącej do odwracania listy:

```
rev :: [a] -> [a]
```

```
rev [] = []
```

```
rev (x:xs) = (rev xs) ++ [x]
```

Zadanie 26 — Oprogramuj funkcję `fib`, która wyznacza n -tą liczbę Fibonacciego w czasie liniowym (zakładając, że operacje arytmetyczne wykonywane są w czasie stałym). **Wskazówka:** przyglądnij się parze (f_{n+1}, f_n) .

3 Listy

Część z tych zadań może być omówionych na wykładzie. Mimo to samodzielnie zaimplementuj te funkcje i przetestuj je w GHCi. Po przetestowaniu zapisz ich kody w pliku o nazwie `testy.hs`

Zadanie 27 — Napisz funkcję `mymap :: (a->b) -> [a] -> [b]` która listę $[x_1, \dots, x_n]$ przekształca w listę $[fx_1, \dots, fx_n]$.

Zadanie 28 — Korzystając z funkcji `mymap` z poprzedniego zadania napisz kody następujących funkcji:

1. `mysum :: [x_1, ..., x_n] -> \sum_{i=1}^n x_i`
2. `myproduct :: [x_1, ..., x_n] -> \prod_{i=1}^n x_i`
3. `myfact :: n -> n!`

Zadanie 29 — Napisz funkcję `mynub`, która usunie z listy wszystkie duplikaty, np. `mynub [1,1,2,2,2,1,4,1] == [1,2,4]`

Zadanie 30 — Napisz funkcję `myinits`, która dla danej listy wyznaczy listę wszystkich jej odcinków początkowych, np.

`myinits [1,2,3,4] == [[], [1], [1,2], [1,2,3], [1,2,3,4]]`

Zadanie 31 — Napisz funkcję `mytails`, która dla danej listy wyznaczy listę wszystkich jej odcinków początkowych, np.

`mytails [1,2,3,4] == [[], [4], [3,4], [2,3,4], [1,2,3,4]]`

Zadanie 32 — Napisz funkcję `partitions`, która dla danej listy `xs` wyznaczy listę wszystkich par `(ys,zs)` takich, że `xs == ys++zs`.

Zadanie 33 — Napisz funkcję `nondec :: Ord a -> [a] -> Bool`, która sprawdza, czy podany argument $[x_1, \dots, x_n]$ jest ciągiem niemalejącym, czyli czy $x_1 \leq x_2 \leq \dots \leq x_n$.

Zadanie 34 — Zaimplementuj samodzielnie funkcję `zip` (nazwij ją `myzip`).

Zadanie 35 — Napisz funkcję `permutations`, która dla danej listy wyznaczy listę wszystkich jej permutacji (możemy założyć, że wszystkie elementy listy wejściowej są różne).

* **Zadanie 36** — Napisz funkcję, która oblicza iloma zerami (w układzie dziesiętnym) kończy się liczba $n!$.

Uwaga: taki pomysł: „mam dane n ; obliczam $n!$; zamieniam na łańcuch s ; odwracam go; liczę ilość początkowych zer” traktujemy jako kompletnie beznadziejny.

Wskazówka: Jaka wyznaczyć największą potęgę liczby 5 która dzieli daną liczbę n ?

Zadanie 37 — Ulepsz następującą "klasyczną" implementację funkcji `quick-sort`:

```
qs [] = []
qs (x:xs) = qs [t|t <- xs, t <= x] ++ [x] ++ qs [t|t <- xs, t > x]
```

Wskazówka: Czy warto z rekursją schodzić do list jednoelementowych?

Zadanie 38 — Niech `mmap f = map (map f)` oraz `mmmap f = map (map (map f))`.

1. Zbadaj typy tych odwzorowań.
2. Przetestuj ich działanie
3. Pokaż, że `mmap = map . map` oraz `mmmap = map . map . map`

4 Funkcje fold

Zadanie 39 — Sprawdź typy i przetestuj działanie funkcji `sum`, `product`, `all` i `any`.

Zadanie 40 — Przetestuj działanie funkcji `foldl (+) 0 xs`, `foldr (+) 0 xs`, `foldl' (+) xs`, `foldr' (+) xs` oraz `sum X` na dużych listach liczb `X`.

Wskazówka: skorzystaj z polecenia `GHCi :set +s`; w celu usunięcia wyświetlania informacji skorzystaj z polecenia `:unset +s`.

Zadanie 41 — Zdefiniuj za pomocą funkcji `foldr` funkcję, które dla listy liczb $[a_1, \dots, a_n]$ oblicza ile liczb parzystych występuje w tej liście.

Zadanie 42 — Napisz funkcję `nondec`, która sprawdza czy dany ciąg $[x_1, \dots, x_n]$ jest niemalejący, czyli, czy $x_1 \leq x_2 \leq \dots \leq x_n$. Znajdź implementację rekurencyjną oraz implementację opartą na zbadaniu listy `zip xs (tail xs)`.

Zadanie 43 — Która z następujących równości jest prawdziwa ?

1. `foldl (-1) e xs = e - sum xs`
2. `foldr (-1) e xs = e - sum xs`

Zadanie 44 — Dla danej listy $xs = [x_1, \dots, x_n]$ funkcja `ssm xs` wyznacza najdłuższą listę $[x_{j_1}, \dots, x_{j_k}]$ taką, że $j_1 = 1$ oraz $x_{j_a} < x_{j_{a+1}}$ dla wszystkich $a = 1 \dots, k-1$.

Na przykład, dla ciągu `xs = [3, 2, 1, 5, 3, 2, 6, 2, 3, 8]` mamy `ssm xs = [3, 5, 6, 8]`. Zdefiniuj funkcję `ssm` za pomocą funkcji `foldl`.

Zadanie 45 — Funkcja `remdupl` usuwa z listy przylegające duplikaty, np. `remdupl [1, 1, 2, 1, 1, 3, 3, 4, 4]` = `[1, 2, 1, 3, 4]`. Oprogramuj tę funkcję za pomocą `foldr` lub `foldl`.

Zadanie 46 — Zdefiniuj za pomocą funkcji `foldr` funkcję, które dla listy liczb $[a_1, \dots, a_n]$ oblicza ile liczb parzystych występuje w tej liście.

Zadanie 47 — Korzystając z funkcji `foldl` i `foldr` napisz funkcję `approx n` zdefiniowaną następująco

$$\text{approx}(n) = \sum_{k=1}^n \frac{1}{k!}$$

Zadanie 48 — Napisz, korzystając z funkcji `foldl`, funkcję która dla ciągu liczb $[a_1, \dots, a_n]$ oblicza $\sum_{k=1}^n (-1)^{k+1} a_k$

Zadanie 49 — Funkcja `filter` może być zdefiniowana za pomocą funkcji `map` i `concat`:

```
filter p = concat . map box
  where box x =
```

Podaj definicję tej funkcji `box`.

Zadanie 50 — Funkcje `takeWhile` i `dropWhile` są podobne do funkcji `take` i `drop`, jednakże ich pierwszym argumentem jest funkcja boolowska zamiast liczby naturalnej. Na przykład

```
takeWhile even [2, 4, 6, 7, 8, 9] = [2, 4, 6]
```

oraz

```
dropWhile even [2, 4, 6, 7, 8, 9] = [7, 8, 9]
```

Podaj rekurencyjne definicje tych funkcji.

Zadanie 51 — Napisz funkcję która dla zadanej listy $[a_1, \dots, a_n]$ elementu typu `[Fractional a]` wyznaczy średnią arytmetyczną oraz wariancję ciągu (a_1, \dots, a_n) . Skorzystaj tylko raz z funkcji `fold`.

Zadanie 52 — Pokaż, że `map f (xs ++ ys) = (map f xs) ++ (map f ys)`. Wywnioskuj z tego następującą własność `(map f) . concat = concat . map (map f)`.

5 Zadania dodatkowe

Zadanie 53 — Na wykładzie sformułowaliśmy następujące twierdzenie:

Jeśli $R \subseteq X \times X$ jest ufundowana oraz $F : V \times X \rightarrow V$ (gdzie V oznacza klasę wszystkich zbiorów), to istnieje dokładnie jedna funkcja $g : X \rightarrow V$ taka, że

$$(\forall x \in X)(g(x) = F(g \upharpoonright \text{prec}(x), x)) ,$$

gdzie $\text{prec}(x) = \{t \in X : (t, x) \in R\}$.

Oto dowód tego twierdzenia rozbity na kilka kroków:

1. Pokaż, że relacja $R \subseteq X \times X$ jest ufundowana wtedy i tylko wtedy, gdy

$$(\forall A \subseteq X)(A \neq \emptyset \rightarrow (\exists a \in A)(A \times \text{prec}(a) = \emptyset)) .$$

2. Niech GPF (good partial functions) oznacza rodziną wszystkich funkcji f takich, że $\text{dom}(f) \subseteq X$ oraz dla każdego $x \in \text{dom}(f)$ mamy $\text{prec}(x) \subset \text{dom}(f)$ oraz $f(x) = F(f \upharpoonright \text{prec}(x), x)$. Pokaż, że jeśli $f_1, f_2 \in GPF$ i $x \in \text{dom}(f_1) \cap \text{dom}(f_2)$ to $f_1(x) = f_2(x)$.
3. Pokaż, że dla każdego $x \in X$ istnieje $f \in GPF$ taka, że $x \in \text{dom}(f)$.
4. Udowodnij twierdzenie.

C.D.N.

Powodzenia,
Jacek Cichoń