Programowanie Funkcyjne
Lista zadań

Jacek Cichoń, WIT, PWr, 2024/25

1 Wstęp

Zadanie 1 — Zdefiniujmy następujące funkcje

```
power :: Int -> Int -> Int
power x y = y ^ x

-- partial application
p2 = power 4
p3 = power 3
```

1. Wyznacz w GCHI wartość wyrażenia `(p2 . p3) 2` i wyjaśnij dlaczego otrzymałeś ten wynik.
2. Zbadaj typy funkcji `p2`, `p3` i `(p2 . p3)`
3. Zapisz powyższe funkcje za pomocą lambda wyrażeń.

Zadanie 2 — 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 oraz siła operatora $\wedge$ za pomocą polecenia :i $(\wedge)$.

Zadanie 3 — Definiujmy następujące funkcje:

```
f :: Int -> Int
f x  = x ^ 2

g:: Int -> Int -> Int
g x y = x+2*y

h :: ....
h x y = f(g x y)
```

1. Jaki jest typ funkcji h? (tzn. uzupełnij `...` w powyższym listingu)
2. Czy `h = f . g` ?
3. Czy `h x = f( g x)` ?

Zadanie 4 — Zapisz operacje binarne $(+)$, $(*)$ za pomocą lambda wyrażeń.

Zadanie 5 — Zapisz funkcje $f(x) = 1 + x * (x + 1)$, $g(x, y) = x + y^2$, $h(y, x) = x + y^2$ za pomocą λ-wyrażeń w językach C++, Python, JavaScript oraz w języku Haskell.

Zadanie 6 — Ustalmy zbiory A, B, C. Niech $\text{curry} : C^{B\times A} \to (C^B)^A$ będzie funkcją zadaną wzorem

$$\text{curry}(\phi) = (\lambda a : A \to (\lambda b : B \to \phi(b, a))) \ .$$

oraz niech $\text{uncurry} : (C^B)^A \to C^{B\times A}$ będzie zadana wzorem

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

1. Pokaż, że $\text{curry} \circ \text{uncurry} = id_{(C^B)^A}$ oraz $\text{uncurry} \circ \text{curry} = id_{C^{B\times A}}$.

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`.

Zadanie 7 — Podaj przykłady funkcji następujących typów

```
(Int -> Int) -> Int
(Int->Int)->(Int->Int)
(Int->Int)->(Int->Int)->(Int->Int)
```

Zadanie 8 — Załóżmy, że chcesz oprogramować funkcję która dla danych liczb a, b oraz funkcji $f : \mathbb{R} \to \mathbb{R}$ oblicza $\int_a^b f(x)dx$. Jaki powinien być typ tej funkcji?

Zadanie 9 — **(Eliminacja pętli)** Wybierz jeden z języków Python, C++ lub JavaScript.

1. Masz daną (czyli oprogramowaną) funkcję $f : \mathbb{N} \to \mathbb{N}$. Oprogramuj funkcję, która dla danego $n \in \mathbb{N}$ oblicza $\sum_{k=0}^{n} f(k)$. Zrób to najpierw (standardowo) za pomocą pętli, a potem oprogramuj ją bez użycia pętli, za pomocą rekursji.
2. Rozważamy następującą funkcję napisaną w pseudokodzie:

   ```
   FUNCTION f(x: DOUBLE): DOUBLE
   BEGIN
     DOUBLE y = sin(x);
     RETURN y*y + y + x;
   ENDFNC
   ```

   Oprogramuj tę funkcję w wybranym języku i następnie wyeliminuj zmienną lokalną y z tego kodu, bez pogarszania jego efektywności.

Zadanie 10 — Zaimplementuj samodzielnie następujące funkcje działające na listach z Prelude: map, zip, zipWith, filter, take, drop

Zadanie 11 — Niech `ff = (2 ^)` oraz `gg = (^ 2)`. Podaj interpretacją tych funkcji. Sprawdź wartości wyrażeń `map (^ 2) [1..10]` oraz `map (2 ^) [1..10]` i wyjaśnij otrzymane wyniki.

Zadanie 12 — Dowiedz się jak można przekonwertować elementy typu Int oraz Integer na typy Float i Double. Dowiedz się jaki jest format funkcji typu `round` z `Double` to `Int`.

1.1 Elementy Teorii Liczb

Zadanie 13 — Funkcją *phi* Eulera nazywamy funkcję określoną wzorem

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

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 14 — Liczbę naturalną n nazywamy *doskonałą* jeśli $n = \sum\{d : 1 \leqslant d < n \wedge d|n\}$. Np. 6 jest liczbą doskonałą, bo $6 = 1 + 2 + 3$. Wyznacz wszystkie liczby doskonałe mniejsze od 10000. **Uwaga:** Do tej pory nie wiadomo, czy istnieje nieskończenie wiele liczb doskonałych.

Zadanie 15 — Parę liczb naturalnych (m, n) nazywamy *zaprzyjaźnionymi* jeśli suma dzielników właściwych każdej z nich równa się drugiej. Znajdź wszystkie zaprzyjaźnione pary o oby składnikach mniejszych od 10^5. **Uwaga:** Do tej pory nie wiadomo, czy istnieje nieskończenie wiele par liczb zaprzyjaźnionych.

Zadanie 16 — Dla $n \in \mathbb{N}^+$ definiujemy

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

1. Zaimplementuj tę funkcję w jezyku Haskell za pomocą "list comprehension".
2. Zoptymalizuj ten kod pisząc rekurencyjną wersję tej funkcji.
3. Wyznacz wartości tej funkcji dla $n = 100, 200, 300, \ldots, 10000$ i postaw jakąś rozsądną hipotezę o $\lim_{n\to\infty} dcp(n)$

1.2 Listy - część I

Kilka z funkcji (np. `nub`, `inits`, `tails`) które będziesz miał do oprogramowania w tej części znaleźć można w module `Data.List`. Mimo to, oprogramuj je samodzielnie (w celu uniknięcia kolizji zmień ich nazwę np. z `tails` na `tails'`).

Zadanie 17 — Napisz funkcje `nub`, która usunie z listy wszystkie duplikaty, np.
`nub [1,1,2,2,2,1,4,1] == [1,2,4]`

Zadanie 18 — Napisz funkcję `inits`, która dla danej listy wyznaczy listę wszystkich jej odcinków początkowych, np.
`inits [1,2,3,4] == [[],[1],[1,2],[1,2,3],[1,2,3,4]]`

Zadanie 19 — Napisz funkcję `tails`, która dla danej listy wyznaczy listę wszystkich jej odcinków końcowych, np.
`tails [1,2,3,4] == [[],[4],[3,4],[2,3,4],[1,2,3,4]]`

Zadanie 20 — Napisz funkcję `splits`, która dla danej listy xs wyznaczy listę wszystkich par (ys,zs) takich, że `xs == ys++zs`.

Zadanie 21 — Oto jedna z możliwych implementacji funkcji `partition`:

```
partition :: (a -> Bool) -> [a] -> ([a], [a])
partition p xs = (filter p xs, filter (not.p) xs)
```

Ulepsz implementację tej funkcji: powinna zwracać ten sam wynik, ale powinna przechodzić przez listę tylko raz.

Zadanie 22 — Zaimplementuj samodzielnie funkcje `permutations` (znajduje się ona w module Data.List), która dla danej listy wyznaczy listę wszystkich jej permutacji (możemy założyć, ze wszystkie elementy listy wejściowej sa różne).

* **Zadanie 23** — **(Klasyczny Problem hetmanów)** Celem jest umieszczenie ośmiu hetmanów na szachownicy, tak aby żadne dwie hetmany nie atakowały się nawzajem; tzn. nie ma dwóch hetmanów w tym samym rzędzie, tej samej kolumnie lub na tej samej przekątnej.

1. Zaimplementuj problem wyszukiwania położeń Hetmanów w Haskell'u korzystając z funkcji permutations.
2. Dwa rozwiązania nazywamy równoważne jeśli pierwsze z nich można otrzymać za pomocą złożeń odbicia poziomego (reverse) oraz odbicia pionowego (np. `map (λ x-> n+1-x)`) z drugiego. Ile jest nierównoważnych poprawnych rozstawień hetmanów ?

Wskazówka: Przedstaw pozycje hetmanów jako listę liczb $[1 \ldots n]$. Przykład: ciąg $[4, 2, 7, 3, 6, 8, 5, 1]$ oznacza, że hetman w pierwszej kolumnie jest w rzędzie 4, hetman w drugiej kolumnie jest w rzędzie 2 itd

* **Zadanie 24** — 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: Jak można wyznaczyć największą potęgę liczby 5 która dzieli daną liczbę n ?

Zadanie 25 — 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 26 — Napisz funkcję `isSorted :: (Ord a) => [a] -> Bool`, która sprawdza, czy podany argument $[x_1, \ldots, x_n]$ jest ciągiem niemalejącym, czyli czy $x_1 \leqslant x_2 \leqslant \ldots \leqslant x_n$.

Zadanie 27 — Zaimplementuj w języku Haskell algorytm `Bubble Sort`.

Zadanie 28 — Typowe implementacje algorytmu `Quick Sort` sprawdzają przed wywołaniem długość listy i jeśli ma ona długość mniejszę lub równą 10, to do posortowania używają metody `Insertion Sort`. Zaimplementuj tę metodę w języku Haskell.

Zadanie 29 — Oszacuj złożoność obliczeniową następującej (kiepskiej) funkcji służącej do odwracania listy:

```
rev :: [a] -> [a]
rev []     = []
rev (x:xs) = (rev xs) ++[x]
```

Zadanie 30 — Funkcja `filter` może myć zdefiniowana za pomocą funkcji `map` i `concat`:

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

Podaj definicję tej funkcji `box`.

Zadanie 31 — 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 32 — Napisz funkcję która dla ciągu łańcuchów $[L_1, \ldots, L_n]$ wyznaczy ich najdłuższy wspólny prefix. Wskazówka: Możesz skorzystać z funkcji `transpose` z modułu `Data.List`.

Zadanie 33 — Napisz funkcję `subCard :: Int -> [a] ->[[a]]` która dla danych parametrów k i $[x_1, \ldots, x_n]$ wyznacza wszystkie podciągi $[x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$ takie, że $1 \leqslant i_1 < i_2 < \ldots < i_k$.

2 Foldy

Zadanie 34 — Sprawdź typy i przetestuj działanie funkcji `sum`, `product`, `all` i `any`.

Zadanie 35 — Przetestuj działanie funkcji `foldl (+) 0 xs`, `foldr (+) 0 xs`, `foldl1 (+) xs`, `foldr1 (+) 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`
Przetestuj następnie działanie funkcji `foldl'` oraz `foldr'` (znajdują się one w module `Data.List`).

Zadanie 36 — Samodzielnie zaimplementuj (oraz przetestuj) funkcję reverse działającą w czasie liniowym. Porój jej skuteczność z algorytmem z Zadania 29.

Zadanie 37 — Zdefiniuj za pomocą funkcji `foldr` funkcję, które dla listy liczb $[a_1, \ldots, a_n]$ oblicza ile liczb parzystych występuje w tej liście.

Zadanie 38 — Korzystając z funkcji `foldl` napisz funkcję `dec2Int` która konwertuje ciąg cyfr na liczbę całkowita, np. `dec2Int [1,2,1] = 121`.

Zadanie 39 — Która z następujących równości jest prawdziwa ?

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

* **Zadanie 40** — Dla danej listy $xs = [x_1, \ldots, x_n]$ funkcja `lmss xs` wyznacza najdłuższą listę $[x_{j_1}, \ldots, x_{j_k}]$ taką, że $j_1 = 1$ oraz $x_{j_a} < x_{j_{a+1}}$ dla wszystkich $a = 1 \ldots, k-1$.
Na przykład, dla ciągu `xs = [3,2,1,5,3,2,6,2,3,8]` mamy `lmss xs = [3,5,6,8]`.

Zadanie 41 — 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 42 — Korzystając z funkcji `foldl` i `foldr` napisz funkcję `approx n` zdefiniowaną następująco

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

Zadanie 43 — Napisz, korzystając z funkcji `foldl`, funkcję która dla ciągu liczb $[a_1, \ldots, a_n]$ oblicza $\sum_{k=1}^{n} (-1)^{k+1} a_k$

Zadanie 44 — Napisz funkcję która dla zadanej listy $[a_1, \ldots, a_n]$ elementu typu [Fractional a] wyznaczy średnią arytmetyczną oraz wariancję ciągu $(a_1, \ldots, a_n)$. Skorzystaj tylko raz z funkcji `fold`.

Zadanie 45 — Zaimplementuj deterministyczny automat skończony który rozpoznaje język tych zero-jedynkowych ciągów która zaczynają się od `01` i zawierają parzystą liczbę jedynek.

C.D.N.

Powodzenia,
Jacek Cichoń