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.

* **Zadanie 46** — Oprogramuj metodę przekształcania niedeterministycznych automatów skończonych w deterministyczne automaty skończone. Sprawdź poprawność działania dla NFA rozpoznającego, czy danych ciąg zero-jedynkowy kończy się symbolem 1.

Zadanie 47 — Napisz funkcję `subsetlist:: [a]->[a]->Bool`, która sprawdza, czy każdy element pierwszej listy jest elementem drugiej listy.

Zadanie 48 — 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`

Zadanie 49 — Zaimplementuj funkcję `dotprod :: (Num a) => [a] -> [a] -> a`, która służy do obliczania iloczynu skalarnego dwóch wektorów interpretowanych jako skończone ciągi numeryczne.

* **Zadanie 50** — Zaimplementuj moduł Matrix, który implementuje podstawowe operacje na macierzach. Macierze w nim interpetujemy jako elementy typu `[[a]]`. Moduł ten nie powinien korzystać z modułu `Data.Vector` (ani z podobnych modułów). Korzystać możesz tylko z modułu `Data.List` (np. może Ci się przydać funkcja `transpose`). Oto jak mniej więcej powinien być zbudowany ten moduł:

```
module Matrix(Matrix, addM, multM, intPowerM, dims, ...) where
type Matrix a = [[a]]

dims :: Matrix a -> (Int,Int)
dims m = ...
addM :: Num a => Matrix a -> Matrix a -> Matrix a
addM m1 m2 = ...
multM :: Num a => Matrix a -> Matrix a -> Matrix a
multM m1 m2 = ...
intPowerM Num a => Matrix a -> Int -> Matrix a
intPowerM m k = ...
```

Postaraj się aby w module tym nie korzystać z operatora `(!!)`. Możesz zakładać, że upublicznione przez ten moduł funkcje będą operowały tylko na poprawnych danych. Funkcję `intPowerM` wystarczy oprogramować dla nieujemnych wartości drugiego parametru. Wskazówka: Możesz skorzystać z list comprehension.

3 Elementy Teorii Kategorii

Zadanie 51 — Wyraź prawo łączności składania $f \circ (g \circ h) = (f \circ g) \circ h$ w języku komutowania diagramów.

Zadanie 52 — Pokaż, że morfizm Id_A jest jednoznaczny, czyli, że jeśli $\mathrm{Id1}_A$ oraz $\mathrm{Id2}_A$ spełniają własności identyczności to $\mathrm{Id1}_A = \mathrm{Id2}_A$.

Zadanie 53 — Pokaż, że dowolne dwa elementy końcowe są izomorficzne. Wywnioskuj z tego podobny fakt dla elementów początkowych.

Zadanie 54 — Wyznacz elementy początkowe i końcowe w kategoriach grup, pierścieni oraz monoidów.

Zadanie 55 — Niech $F : \mathcal{C} \to \mathcal{D}$ będzie funktorem z kategorii $\mathcal{C}$ do kategorii $\mathcal{D}$. Pokaż, że jeśli w kategorii $\mathcal{C}$ obiekty X i Y są izomorficzne, to w kategorii $\mathcal{D}$ obiekty $F(X)$ oraz $F(Y)$ są izomorficzne. Zadanie to można podsumować następująco: funktory zachowują izomorficzność obiektów.

Zadanie 56 — Załóżmy, że $F : \mathcal{C} \to \mathcal{D}$ i $G : \mathcal{D} \to \mathcal{E}$ są funktorami. Rozważamy odwzorowanie $H : \mathcal{C} \to \mathcal{E}$ określone dla obiektów $X \in \mathrm{ob}(\mathcal{C})$ wzorem $H(X) = G(F(X)$ oraz dla morfizmów $f : X \to Y$ kategorii $\mathcal{C}$ wzorem $H(f) = G(F(f))$. Odwzorowanie to oznaczamy symbolem $G \circ F$.

1. Sprawdź, że definicja ta jest poprawna.
2. Pokaż, że $G \circ F : \mathcal{C} \to \mathcal{E}$ jest funktorem.

Zadanie 57 — Niech $F, G, H : \mathcal{C} \to \mathcal{D}$ będą funktorami. Załóżmy, że $\alpha : F \dot{\to} G$ oraz $\beta : G \dot{\to} H$ będą naturalnymi transformacjami. Pokaż, że $\beta \circ \alpha : F \dot{\to} H$ jest naturalną transformacją.

Zadanie 58 — Załóżmy, że $F : \mathcal{C} \to \mathcal{C}$ jest funktorem kontrawariantnym. Pokaż, że $F \circ F$ jest funktorem. Zastosuj powyższy fakt do kontrawariantnego funktora $F_A(X) = A^X$ i uzupełnij następujący kod

```
data UP2 a b = UP2 ((b -> a) -> a)
instance Functor (UP2 a) where
  fmap f (UP2 phi) = UP2 (???)
```

Zadanie 59 — Ustalmy zbiór A. Rozważamy odwzorowanie $F_A : \mathbf{Set} \to \mathbf{Set}$ określone wzorem $F_A(X) = A \times X$.

1. Rozszerz odwzorowanie F_A do funktora kategorii **Set**.
2. Zaimplementuj funktor F_A w języku Haskell za pomocą funkcji `((,))`.
3. W zadaniu 6 zdefiniowaliśmy funkcję `uncurry`. Oblicz `uncurry ((,))`.

Zadanie 60 — Znajdź naturalną transformacje między następującymi parami funktorów i zaimplementuj je w języku Haskell:

1. η_1 : Maybe $\overset{\bullet}{\to}$ [ ]
2. η_2 : Maybe $\overset{\bullet}{\to}$ Either ()

Zadanie 61 — Pokaż, że następujące dwa typy

```
data T1 a b c = TT1 (b -> a , c -> a)
data T2 a b c = TT2 (Either b c) -> a
```

są izomorficzne i zaimplementuj je w języku Haskell. Wskazówka: Wyznacz moce obu typów dla ustalonych a, b, c.

Zadanie 62 — Rozważmy odwzorowanie $K : \mathrm{ob}(\mathbf{Set}) \to \mathrm{ob}(\mathbf{Set})$ określone wzorem $K(X) = \emptyset^X$. Pokaż, że odwzorowania K nie można rozszerzyć do funktora kategorii **Set**.

Zadanie 63 — Rozważmy odwzorowania $PS_1, PS_2 : \mathrm{ob}(\mathbf{Set}) \to \mathrm{ob}(\mathbf{Set})$ zdefiniowane wzorem $PS_1(X) = PS_2(X) = \{Y : Y \subseteq X\}$.

1. Rozszerzamy odwzorowanie PS_1 na morfizmy $f : X \to Y$ kategorii **Set** wzorem $PS_1(f)(A) = f[A]$. Pokaż, że PS_1 jest funktorem.
2. Rozszerzamy odwzorowanie PS_2 na morfizmy $f : X \to Y$ kategorii **Set** wzorem $PS_2(f)(A) = f[A] \cup (Y \setminus f[X])$. Pokaż, że PS_2 jest również funktorem.

Zadanie 64 — Pokaż, że `map f (xs ++ ys) = (map f xs) ++ (map f ys)`. Wywnioskuj z tego następującą własność `(map f) . concat = concat . map (map f)`.

Zadanie 65 — Niech `f:[a] -> [a]` będzie określona wzorem `f x = x++x`.

1. Pokaż, że nie istnieje funkcja `h:a->a` taka, że `f xs = map h xs` dla każdego łańcucha `xs`.
2. Pokaż, że nie istnieje funkcja h taka, że `f xs = map h xs`, gdzie

$$f[x_1, \ldots, x_n] = [[x_1], [x_1, x_2], \ldots, [x_1, \ldots, x_n]] \ .$$

Zadanie 66 — Rozważmy następujące definicje typów

```
data T1 a = T1 (Int -> a)
data T2 a = T2 (a -> Int)
data T3 a = T3 (a -> a)
data T4 a = T4 ((Int -> a) -> Int)
data T5 a = T5 ((a -> Int) -> Int)
```

Które z nich są funktorami?

Zadanie 67 — Rozważmy następujący konstruktor drzew

```
data MyTree a = Empty | Leaf a | Node (MyTree a) (MyTree a)
     deriving (Eq)
```

1. Przypisz konstruktor `MyTree` do klasy `Show`. Musisz to zrobić warunkowo: typ `a` musi być klasy `Show`.
2. Przypisz konstruktor `MyTree` do klasy `Functor`.
3. Przypisz konstruktor `MyTree` do klasy `Foldable`.
4. Oprogramuj rozsądnie funkcję `height:: MyTree a -> Int`.
5. Oprogramuj funkcję `nbLeafs:: MyTree a -> Int`, która wyznacza liczbę liści w drzewie typu `MyTree`.
6. Oprogramuj funkcję `nbNodes`, która wyznacza liczbę liści w drzewie typu `MyTree`.
7. Dowiedz się jak można wykorzystać quickCheck do sprawdzania samodzielnie zdefiniowanych typów i sprawdź, czy równość `nbLeafs == nbNodes` jest prawdziwa. Jeśli nie, to postaw inną hipotezę i przetestuj ją na 10^5 testach.

3.1 Naturalne transformacje

Zadanie 68 — Rodzina $\mathrm{id} = (\mathrm{id}_A)_{A \in \mathrm{ob}(\mathbf{Set})}$ jest naturalnym odwzorowaniem z funktora Id w funktor Id, czyli $\mathrm{id} : Id \stackrel{\bullet}{\to} Id$.

1. Jak zaimplementowana jest w języku Haskell ta naturalna transformacja?
2. Załóżmy, że $\eta : Id \stackrel{\bullet}{\to} Id$. Pokaż, że $\eta = \mathrm{id}$.

Zadanie 69 — Niech $F : \mathbf{Set} \to \mathbf{Set}$ będzie funktorem oraz niech $\bullet$ będzie ustalonym elementem. Dla dowolnego niepustego zbioru A oraz elementu $x \in A$ niech $c_{\bullet,x} : \{\bullet\} \to A$ będzie funkcją zadaną wzorem $c_{\bullet,x}(\bullet) = x$. Załóżmy, że $e \in F(\{\bullet\})$.

1. Niech $\eta_A : A \to F(A)$ będzie określona wzorem $\eta_A(x) = F(c_{\bullet,x})(e)$ oraz niech $\eta = (\eta_A)_{A \in \mathrm{ob}(\mathbf{Set})}$. Pokaż, że $\eta : Id \stackrel{\bullet}{\to} F$
2. Pokaż, że zdania "$(\exists \eta)(\eta : Id \stackrel{\bullet}{\to} F)$" oraz "$F(\{\bullet\}) \neq \emptyset$" są równoważne.

Zadanie 70 — Pokaż, że istnieje nieskończenie wiele naturalnych transformacji z funktora identyczności w funktor tworzenia list [ ]

Zadanie 71 — Niech R będzie funktorem `Reader` z parametrem $\mathbb{N}$, czyli $R(X) = X^{\mathbb{N}}$ i $R(f)(\alpha) = f \circ \alpha$ dla $\alpha \in R(X)$ oraz $f : X \to Y$.

1. Znajdź naturalną transformację $\eta : R \xrightarrow{\bullet} Maybe$.
2. Znajdź naturalną transformację $\eta : R \xrightarrow{\bullet} L$, gdzie $L(X) = [X]$
3. Znajdź naturalną transformację $\eta : R \xrightarrow{\bullet} Id$.
4. Niech $Dp(X) = X \times X$ oraz $(Dp(f))(x_1, x_2) = (f(x_1), f(x_2))$. Sprawdź, że Dp jest funktorem oraz znajdź jakąś naturalną transformację $\eta : R \xrightarrow{\bullet} Dp$
5. Zaimplementuj znalezione transformacje w języku Haskell.

Może potrafisz uogólnić powyższe fakty?

Zadanie 72 — Rozważamy funktor dp z poprzedniego zadania.

1. Spróbuj wyznaczyć wszystkie naturalne transformacje $\eta : Id \xrightarrow{\bullet} Dp$
2. Spróbuj wyznaczyć wszystkie naturalne transformacje $\eta : Dp \xrightarrow{\bullet} Id$
3. Spróbuj wyznaczyć wszystkie naturalne transformacje $\eta : Dp \xrightarrow{\bullet} Dp$
4. Zaimplementuj znalezione transformacje w języku Haskell.

Zadanie 73 — Zaimplementuj funkcję która z listy $[x_1, \ldots, x_n]$ wybiera te x_i, że $i < n$ oraz $x_i > x_{i+1}$.

Zadanie 74 — Zaimplementuj funkcję która z listy $[x_1, \ldots, x_n]$ wybiera te x_i, że

$$x_i > \max[x_{i+1}, x_{i+2}, \ldots, x_n] \ .$$

Pewnie pierwsza implementacja będzie działała w czasie $O(n^2)$. Ulepsz ją do implementacji działającej w czasie $O(n)$. Zainstaluj następnie bibliotekę `QuickCheck` i sprawdź, czy obie funkcje zwracają ten sam wynik.

C.D.N.

Powodzenia,
Jacek Cichoń