Programowanie Funkcyjne
WPPT
Lista zadań

Jacek Cichoń, WPPT, PWr, 2022/23

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

1 Wstęp

1.1 Teoria

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

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

Jeśli $f : X \to A$ oraz $g : X \to B$, to przez (f, g) oznaczamy funkcję $(f, g) : X \to A \times B$ 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 $\Delta_X : X \to X \times X : x \to (x, x)$ oraz $f, g : X \to Y$.

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

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 $\mathrm{curry}_{A,B,C} : C^{A\times B} \to (C^B)^A$ wzorem

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

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

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

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

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

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

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

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

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

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

Zadanie 6 — Funkcją Ackermana nazywamy funkcję $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ 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

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

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 następujące funkcję

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

Wyeliminuj 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 $[x_1, \ldots x_n]$ i jako wynik zwracają:

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

Zadanie 12 — Wyeliminuj z następującego kodu pętlę (zastąp ją rekursją i, oczywiście, pozbądź 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 rekurencyjne 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 pitagorejską jeśli $a^2 = b^2 + c^2$ oraz $\text{gcd}(b, c) = 1$. Wyznacz wszystkie właściwe trójki pitegorejskie 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 ^)` oraz `gg = (^ 2)`. Podaj interpretacją 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 przzetestuje je w GHCi. Po przetestowaniu zapisz ich kody w pliku o nazwie testy.hs

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

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

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

Zadanie 29 — Napisz funkcje 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 liste wszystkich par (ys,zs) takich, że `xs == ys++zs`.

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

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

Zadanie 35 — Napisz funkcje `permutations`, która dla danej listy wyznaczy listę wszystkich jej permutacji (możemy założyć, ze wszystkie elementy listy wejściowej sa 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 $[a_1, \ldots, a_n]$ oblicza ile liczb parzystych występuje w tej liście.

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

Zadanie 43 — Która z następujących równości jest prawdziwa ?

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

Zadanie 44 — Dla danej listy $xs = [x_1, \ldots, x_n]$ funkcja `ssm 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 `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 $[a_1, \ldots, a_n]$ 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 $[a_1, \ldots, a_n]$ oblicza $\sum_{k=1}^{n} (-1)^{k+1} a_k$

Zadanie 49 — 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 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 $[a_1, \ldots, a_n]$ elementu typu [Fractional a] wyznaczy średnią arytmetyczną oraz wariancję ciągu $(a_1, \ldots, a_n)$. Skorzystaj tylko raz z funkcji `fold`.

Zadanie 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 Elementy Teorii Kategorii - I

Zadanie 53 — Pokaż, że strzałka Id_A jest jednoznaczna, czyli, że jeśli Id1_A oraz Id2_A spełniają własności identyczności to $\text{Id1}_A = \text{Id2}_A$.

Zadanie 54 — Pokaż, że dowolne dwa elementy końcowe są izomorficzne. Wywnioskuj z tego podobny fakt dla elementów początkowych.

Zadanie 55 — Wyznacz elementy początkowe i końcowe w kategorii grup.

Zadanie 56 — Rozważmy przyporządkowanie $F : Set \to Set$ określone wzorem $F(X) = X \cap \mathbb{N}$ (gdzie $\mathbb{N}$ oznacza zbiór liczb naturalnych). Pokaż, że przyporządkowania F nie można rozszerzyć na morfizmy kategorii Set do funktora.

Zadanie 57 — Rozważmy odwzorowanie $K : Set \to Set$ określone wzorem $K(X) = \emptyset^X$. Pokaż, że przyporządkowania K nie można rozszerzyć na morfizmy kategorii Set do funktora.

Zadanie 58 — Wyraź prawo łączności składania $f \circ (g \circ h) = (f \circ g) \circ h$ w języku komutowania diagramów.

Zadanie 59 — Niech A będzie ustalonym zbiorem. Pokaż, że następujące odwzorowania są endofunktorami kategorii Set:

1. $C(X) = A; C(f :: X \to Y) = id_A$
2. $S(X) = X \times X, S(f :: X \to Y) = (\lambda(x_1, x_2) :: X \times X \to (f(x_1), f(x_2)))$
3. $R_A(X) = X^A, R_A(f :: X \to Y) = (\lambda\phi :: X^A \to f \circ \phi)$

Zadanie 60 — Zinterpretuj w języku informatyki komutowanie następującego diagramu

$$\begin{array}{ccc} Int & \xrightarrow{succ_{Int}} & Int \\ {\scriptstyle toReal}\downarrow & & \downarrow{\scriptstyle toReal} \\ Real & \xrightarrow[succ_{Real}]{} & Real \end{array}$$

6 Funkcje

Zadanie 61 — 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
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 . Wskazówka: Skorzystaj z funkcji permutations z modułu Data.List.

Zadanie 62 — Dla dowolnej liczby rzeczywistej $a > 0$ ciąg rekurencyjny $x_0 = a$, $x_{n+1} = g(x_n)$, gdzie $g_a(x) = (x + \frac{a}{x})/2$ zbiega (i to dosyć szybko) do liczby $\sqrt{a}$. Napisz swoją funkcję służącą do wyznaczania pierwiastka $\sqrt{a}$ określą dla liczb typu `Double` która tak długo będzie iterować funkcję g_a aż otrzyma liczbę x taką $g_a(x) = x$. Porównaj dokładność tak zaimplementowanego pierwiastka z funkcją `sqrt` Haskell'a.

* **Zadanie 63 —** Na ile sposobów możesz zapisać liczbę naturalną n za pomocą dodawania liczb $1, 2, 5$? Kolejność dodawania nie odgrywa roli, czyli, np. przedstawienia $1+2+2+5$ i $2+1+5+2$ są równoważne. Postaraj się, oczywiście, napisać możliwie uniwersalny kod.

Zadanie 64 — W języku Haskell często używa się operacji `<$>` definiowanej wzorem `f <$> x = map f x`. Czyli jest operacja `map` zapisana jako operator binarny. Zapisz za pomocą tej operacji złożenie `map.map`.

7 Funktory i operatory

Zadanie 65 — Rozważmy następująco zdefiniowany typ drzew binarnych

```
data Tree a = Leaf a | Inner (Tree a) (Tree a)
```

1. Zaprogramuj instancję typu `Tree` dla klasy `Eq`.
2. Napisz funkcję, która zlepia dwa drzewa łącząc je wspólnym korzeniem.
3. Napisz funkcję `treeDepth` obliczającą głębokość drzewa typu `Tree`. Funkcja ta dla drzewa składającego się tylko z liścia ma zwracać 0, czyli `treeDepth (Leaf _) = 0`.
4. Napisz funkcję `treeB :: Int -> Tree Int` która dla danego $n \geq 0$ konstruuje drzewo głębokości n, którego wszystkie lewe poddrzewa są pojedynczym liściem `Leaf 0`, a najgłębszy liść jest równy `Leaf 1`.
5. Napisz funkcję która przekształca dane drzewo typu `BinTree a` za funkcji `f:a -> BinTree b` zastępującą liście `Leaf x` drzewami `f x`.

Zadanie 66 — Zdefiniuj typ `RF2` mający reprezentować przestrzeń R^2 za pomocą dwóch liczb typu `Float`. Następnie

1. Zdefiniuj klasę `VectorSpace a` zawierającej funkcje `vnull`, `vmult`, `vadd` reprezentujące wektor zerowy, mnożenie przez skalar oraz dodawanie wektorów
2. Oprogramuj instancję typu `RF2` do klasy `VectorSpace`
3. Dodaj do klasy `VectorSpace` funkcję `isBasis::[a] -> Bool`, która rozstrzyga, czy dana lista wektorów jest bazą przestrzeni.
4. Dodaj do klasy `VectorSpace` funkcję `<.>` implementującą iloczy skalarny wektorów.
5. Zdefiniuj typ `RF3` mający reprezentować przestrzeń R^3 za pomocą trzech liczb typu `Float` oraz oprogrmuj jej instancję do klasy `VectorSpace`.

Zadanie 67 — Zdefiniuj samodzielnie typ `Complex` mający reprezentować liczby zespolone. Oprogramuj instancję `Complex` do klasy `Num`.

Zadanie 68 — Zdaniem rachunku zdań jest wyrażenie następującej postaci: `nazwa zmiennej` (łańcuch), F, T, $p \wedge q$, $p \vee q$, $\neg p$ gdzie p i q zdaniami.

1. Zdefiniuj typ danych `Prop` służących do reprezentacji zdań rachunku zdań w języku Haskell.
2. Napisz funkcję `vars: Prop -> [String]`, która wyznacza zmienne występujące w zdaniach. Funkcja ta powinna usuwać duplikaty.
3. Załóżmy, że masz daną listę zawierającą nazwy zmiennych i ich wartości, np [("p", False),("q", True)]. Napisz funkcję `eval:: Prop -> [(String,Bool)]-> Bool`, która wykonuje ewaluację przekazanego zdania i wartościowań zmiennych.
4. Napisz funkcję `tautology:: Prop -> Bool`, która sprawdza, czy dane zdanie jest taulologią.
5. Napisz funkcję `simpl : Prop -> Prop`, która wykonuje uproszczenia przekazanego wyrażenia, np. $simpl(T \vee \phi) = simpl(\phi)$, $simpl(F \wedge \phi) = simpl(\phi)$.

8 Monady

Zadanie 69 — Rozważmy konstruktor typów data Pkt a = Pkt (a,a)

1. Zaimplementuj Pkt jako funktor
2. Zaimplementuj Pkt jako monadę; podaj odpowiednią wersję <*> tak aby móc zaliczyć Pkt do klasy Applicative
3. Oblicz `sequence` [Pkt (x_1, y_1), ... Pkt (x_n, y_n)].

Zadanie 70 — Pokaż, że typ `Tree a` z zadania 65 jest monadą i wyznacz dla niego wszystkie typowe związane z monadą funkcje.

Zadanie 71 — Niech $Z(X) = \{a\}$, gdzie a jest ustalonym obiektem oraz $Z(f) = Id_{\{a\}}$ dla dowolnego morfizmu $f : X \to Y$. Pokaż, że Z jest monadą.

Zadanie 72 — Niech I będzie funktorem identycznościowym. Pokaż, że I jest monadą. Wyznacz wszystkie typowe funkcje monadyczne dla tej monady.

9 Zadania dodatkowe

Zadanie 73 — Na wykładzie sformułowaliśmy następujące twierdzenie:

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

$$(\forall x \in X)(g(x) = F(g \restriction prec(x), x)) ,$$

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

Oto dowód tego twierdzenia rozbity na kilka kroków:

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

$$(\forall A \subseteq X)(A \neq \emptyset \to (\exists a \in A)(A \times prec(a) = \emptyset)) .$$

2. Niech GPF (good partial functions) oznacza rodziną wszystkich funkcji f takich, że $dom(f) \subseteq X$ oraz dla każdego $x \in dom(f)$ mamy $prec(x) \subset dom(f)$ oraz $f(x) = F(f \restriction prec(x), x)$. Pokaż, że jeśli $f_1, f_2 \in GPF$ i $x \in dom(f_1) \cap dom(f_2)$ to $f_1(x) = f2(x)$.
3. Pokaż, że dla każdego $x \in X$ istnieje $f \in GPF$ taka, że $x \in dom(f)$.
4. Udowodnij twierdzenie.

C.D.N.

Powodzenia,
Jacek Cichoń