Programowanie Funkcyjne - zadania

Radosław Rowicki

20 maja 2019

Informacje wstępne:

1. Zadania wysyłamy na adres radrowicki@gmail.com. Chciałbym, żeby w temacie wiadomości było w formacie "[WWW] <imię> <nazwisko>"

2. Wszystkie zadania typu "napisz funkcję" można zaprogramować w dowolnym języku programowania (ale w takim, w którym w ogóle występują funkcje lub metody...), pseudokodzie lub matematyce **bez używania jakichkolwiek pętli i tym bardziej instrukcji goto**. Można natomiast korzystać z rekurencji.

3. Nie trzeba zrobić wszystkich zadań, ale warto przynajmniej spróbować. Mogę pozytywnie ocenić nawet nieudaną próbę.

4. Nie korzystamy z chamskich gotowców (jak jest polecenie 'napisz funkcję która mnoży' oczywiste jest, że nie korzystamy z mnożenia ani dzielenia).

5. Przy rozwiązaniach w prawdziwych językach programowania liczba rzeczywista może być interpretowana jako dowolny typ zmiennoprzecinkowy. Nie zważamy na precyzję floata, chyba że treść stanowi inaczej.

6. Polecam niektóre punkty opisać matematycznie lub pseudokodem, gdyż wiele typowych języków programowania nie dopuszcza przyjmowania funkcji jako argumentu. Jednakże nie mam nic przeciwko rozwiązaniom typu opakowanie funkcji w klasę, użycie wskaźników na funkcje, lambdy z Java8/C++11 itp. Jeśli jednak chcesz mieć możliwość fizycznego przetestowania swojej implementacji, to z powszechnie znanych języków powinien nadać się JavaScript.

7. W razie jakichkolwiek wątpliwości - pisz.

Zad 1. Matematyka

Napisz funkcję która:

1. Przyjmie liczbę naturalną n i zwróci liczbę naturalną, która zaokrągla w dół $\sqrt{n}$. (3 pkt)
2. Przyjmie funkcję $f \in (\mathbb{R} \to \mathbb{R})$ i zwróci funkcję o takiej samej dziedzinie i przeciwdziedzinie, której wykres otrzymuje się przez przesunięcie wykresu funkcji f w lewo o 1 i dwukrotnym spłaszczeniu (powinowactwo prostokątne) w pionie. (2 pkt)
3. Przyjmie wartość rzeczywistą ϵ i zwróci funkcję, która przyjmuje funkcję i zwraca funkcję, która jest jej ilorazem różnicowym (pochodną) przy zadanym ϵ. (3 pkt)

Zad 2. Teoria

Dowiedz się, czym jest *rekurencja ogonowa* oraz *złożoność czasowa i pamięciowa algorytmu* i opisz mi swoimi słowami, na czym to polega. Zależy mi na tym, byś udowodnił mi, że rozumiesz o co chodzi, więc liczę na dość nieformalne wytłumaczenie. (po 2 pkt)

Zad 3. Wydajność

Wyposażony w wiedzę z poprzedniego zadania, napisz funkcję (zważając na zadaną złożoność) która:

1. Obliczy silnię n.
   - Czasowa: $O(n)$, Pamięciowa stała (3 pkt)
   - Czasowa: $O(n)$, Pamięciowa $O(n)$ (1 pkt)
2. Obliczy k-tą potęgę liczby n.
   - Czasowa: $O(\log_2 k)$, Pamięciowa stała (5 pkt)
   - Czasowa: $O(\log_2 k)$, Pamięciowa $O(\log_2 k)$ (3 pkt)
   - Czasowa: $O(k)$, Pamięciowa stała (3 pkt)
   - Czasowa: $O(k)$, Pamięciowa $O(k)$ (1 pkt)

W każdym podpunkcie oceniony będzie tylko najwyżej punktowany wariant. Należy założyć że użyty język programowania wspiera rekurencję ogonową.

Zad 4. Punkty stałe

Dana jest funkcja $cons : A \to (\mathbb{N} \to A) \to \mathbb{N} \to A$ oraz $series : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$ które są zdefiniowane następująco:

$$cons(x)(f) = \lambda n.\ \text{if } n = 0 \text{ then } x \text{ else } f(n-1)$$

$$series(s) = \lambda n. \sum_{i=0}^{n} s_i$$

A także funkcja f określona tak:

$$f(s) = \text{cons}(0) \ \circ \ \text{series} \ \circ \ \text{cons}(1)$$

1. Jaka jest najbardziej ogólna dziedzina i przeciwdziedzina funkcji f? (2 pkt)
2. Znajdź wszystkie punkty stałe funkcji $cons(1)$ (lub pokaż że takich nie ma) (3 pkt)
3. Znajdź wszystkie punkty stałe funkcji f (lub pokaż że takich nie ma) (6 pkt)

Zad 5. Teoria zbiorów

Pokaż, że zbiory $A \to (B \to C)$ oraz $(A \times B) \to C$ mają zawsze tyle samo elementów (tzn. istnieje *bijekcja* między nimi). (3 pkt)

Za rozwiązanie tego zadania przez konstrukcję bijekcji lub skorzystanie z twierdzenia Cantora-Bernsteina-Schrödera dodatkowo (1 pkt)

Zad 6. Lambada*

Wyobrażacie sobie matematykę bez liczb? Co by było gdybym w tym momencie powiedział Wam że liczby nie istnieją? Bardziej obeznany matematyk już sięga do kieszeni i wyciągnąwszy z niej teorię zbiorów i ZFC zaczyna lepić sobie arytmetykę Peana, lecz ja w tym momencie mówię – zbiory też Wam zabieram. Matematyka bez ZF(C)? Czy mogę być bardziej pazerny? Mogę. Bowiem zabieram wam również to, co dla wielu z was jest zapewne najcenniejsze – logikę.

No ale co dalej panie autorze? Zostawiłeś nas w jakimś archaicznym świecie gdzie nie możemy nawet zbadać prawdy. Co możemy robić?

Ale autor zadań okazał się być miłosierny. Jest jedna rzecz którą wam zostawiam – funkcje.

Funkcje? Ale jak to? Bez zbiorów? Co z dziedzinami?

To będą specjalne, informatyczne funkcje. To będą **lambda termy**. W naszym świecie wszystko jest funkcją. Ja jestem funkcją, ty jesteś funkcją, argumenty i wartości zwracane są funkcjami. Czymże jest więc wspomniany lambda term?

- Lambda abstrakcja lub inaczej funkcja (np $\lambda a.B$) gdzie a jest zmienną a B lambda termem
- Aplikacja (np AB), gdzie A i B są lambda termami
- Zmienne (np x). Mogą one występować tylko w ciałach lambda abstrakcji, w których były one argumentami, lub jako argumenty.

Lambda termy opisują funkcje w naszym świecie. Może się zdarzyć że kilka lambda termów opisuje ten sam obiekt. Taka "relacja" rządzi się następującymi zasadami:

- α**-konwersja** – lambda term $\lambda a.B$ opisuje ten sam obiekt co $\lambda x.Y$ wtedy, gdy zmienna x nie występuje w B oraz Y jest termem otrzymanym po podstawieniu wszystkich wystąpień a na x w podwyrażeniach które nie są lambda abstrakcjami z a jako argumentem. Przykład: $\lambda a.a(\lambda a.a) =_\alpha \lambda x.x(\lambda a.a)$

- β**-redukcja** – aplikacja $(\lambda a.A)B$ opisuje ten sam obiekt co A po podstawieniu wszystkich wystąpień a na B w podwyrażeniach które nie są lambda abstrakcjami z a jako argumentem. Przykład: $(\lambda a.a(\lambda a.a))(\lambda x.xx) =_\beta (\lambda x.xx)(\lambda a.a)$

- η**-redukcja** – Wyrażenie $(\lambda x.fx)$ redukuje się do f, zakładając że f jest tu legalnie.

No to co? Lecimy! W poniższych zadaniach można korzystać z lambda termów zdefiniowanych ściśle wyżej od nich. W szczególności nie można tworzyć wartości rekurencyjnie (ich lambda termy musiałyby być nieskończone...).

Logika (3 pkt)

Zdefiniujmy **prawdę** jako $\lambda a.\lambda b.a$ oraz **fałsz** jako $\lambda a.\lambda b.b$. Zdefiniuj operacje logiczne: negacja, koniunkcja, alternatywa, implikacja, równoważność, wyrażenie if-then-else.

Liczby naturalne (6 pkt)

Będziemy opisywać liczby naturalne w nastepujący sposób: liczba naturalna n oznacza funkcję która przyjmie f i zwróci f n-krotnie złożone same ze sobą. W szczególności $0 := \lambda f.\lambda x.x$, $2 := \lambda f.\lambda x.f(fx)$ oraz intuicyjnie $n := \lambda f.f^n$.

- Zdefiniuj funkcję następnika, tzn funkcję która przyjmie liczbę i zwróci o 1 większą.

- Zdefiniuj dodawanie, tj intuicyjnie $plus := \lambda n.\lambda m.\lambda f.f^{n+m}$

- Zdefiniuj test na zero, tzn funkcję która zwróci prawdę gdy argument jest zerem, a fałsz gdy nie jest

- Zdefiniuj mnożenie

- Zdefiniuj funkcję poprzednika która zwraca liczbę o 1 mniejszą, lub 0 jeśli argumentem jest 0. Tutaj wytłumacz jak twoje rozwiązanie działa.

- Zdefiniuj odejmowanie

Listy (10 pkt)

To jest sekcja trudniejsza, nie trzeba tego robić. Niemniej, doceniam wszelkie refleksje i próby.

Określam listy analogicznie do liczb. Tak jak liczba naturalna przyjmuje jako argument "co zrobić z zerem" oraz "co zrobić z następnikiem" (liczba naturalna w systemie Peano zapisana jako $S(S(ZERO))$ w świecie lambd ma reprezentację $\lambda S.\lambda Z.S(S(Z))$), to lista będzie opisywana przez funkcję która mówi "co podstawić pod listę pustą" oraz "co zrobić z elementem listy i całą resztą listy na prawo".

Przykładowo $[] := \lambda c.\lambda e.e$ albo $[1,2] := cons(1)(cons(2)([])) := \lambda c.\lambda e.c(1)(c(2)e)$

- Zdefiniuj *cons* – funkcję która przyjmie element wraz z listą i zwróci tą listę z doklejonym owym elementem z lewej strony.
- Napisz funkcję która zwróci długość listy
- Napisz funkcję która zwróci sumę wszystkich elementów listy
- Napisz funkcję przyjmie listę i zwróci ją odwróconą
- Napisz funkcję która przyjmie listę, f oraz a i będzie intuicyjnie robić coś takiego: $fun(f)(a)([x, y, z]) = f(f(f(a)(x))(y))(z)$