Programowanie funkcyjne - zadania kwalifikacyjne

Grzegorz Fabiański

03.VI.2016

Uwagi wstępne:

1. to jest wersja 0.4.1. Na końcu jest changelog.
2. Ilekroć zadani mówi zaimplementuj konieczne jest podanie kodu. Dozwolona jest implementacje w pseudokodzie. Kod powinien być bardzo czytelny - nawet jeśli implementacja będzie w jakimś istniejącym języku, to prawdopodobnie nie będę jej uruchamiał - jej poprawność powinna być oczywista przy czytaniu.
3. Polecenie opisz algorytm nie wymaga podawania kodu. Jeśli uważasz, że fragment kodu (bądź cała implementacja) pomoże Ci uczynić Twój opis lepszym to możesz go umieścić. Jednak opis słowny jest całkowicie wystarczający.
4. Zrozumiem kod napisany w każdym z języków C/C++/Java/Pascal/Python/JavaScript, jeśli jednak chciałbyś pisać w czymś bardziej egzotycznym to prosiłbym wcześniej o spytanie się. Jeśli chcesz oddać kod w Haskellu to znaczy, że te warsztaty raczej nie są dla Ciebie.
5. Nie trzeba zrobić wszystkich zadań by dostać się na warsztaty. Wydaje mi się że przedstawione poniżej zadnia są ciekawe (czyli w szczególności niełatwe), ale i rozwiązywalne - gdybyś jednak uważał że zadania są beznadziejnie trudne (np. w porównaniu do innych serii zadań kwalifikacyjnych) to prosiłbym o sygnał.
6. Gdyby cokolwiek było niejasnego to nie dociekajcie sami o co chodzi, nie googlujcie, tylko od razu piszcie maila (dokładnie odwrotnie niż na StackOverflow). Adres: grzegorz.fabianski+www12@gmail.com

Zad 1. Bez pętelek.

Zaimplementuj, w dowolnym języku programowania (wliczając w to pseudokod) poniższe funkcje. Nie wolno ci jednak używać jakiejkolwiek odmiany pętli (możesz jednak używać funkcji rekurencyjnych).

1. napisz funkcję sprawdzającą pierwszość w czasie $O(\sqrt{n})$.
2. napisz funkcję liczącą $a^b \bmod p$ przy pomocy szybkiego potęgowania (czas działania proporcjonalny do logarytmu z b).
3. rozwiąż powyższe tak, by każdy RETURN zawierało tylko wywołanie jednej funkcji (być może rekurencyjne). Dopuszczamy napis: A=X*X; RETURN F(A, X), ale nie RETURN 2*F(X).

   Jeśli to wymaganie wydaje Ci się sztuczne, to możesz się zastanowić jak wywoła stos wygląda rekurencyjnych w każdym z tych przypadków (i jakie optymalizacji może zrobić kompilator) - choć to pytanie jest poza kwalifikacją (bo sprawdza znajomość niskopoziomowego działania komputera, a to jest poza bezpośrednim tematem warsztatów).

Przed każdą funkcją rekurencyjną powinien być komentarz opisujący co ona finalnie liczy.

Zad 2. Truistyczne tożsamości

Udowodnij że jest bijekcja pomiędzy poniższymi zbiorami (zbiory A, B, C są parami rozłączne, E, F niekoniecznie):

1. $(A \times B) \to C$ oraz $A \to (B \to C)$). W pogadance niżej jest wskazówka.
2. $(A \cup B) \to C$ oraz $(A \to C) \times (B \to C)$.
3. $A \to (E \cap F)$ oraz $(A \to E) \cap (A \to F)$.

W powyższym zadaniu strzałka oznacza zbiór funkcji między zbiorami. Krzyżyk oznacza iloczyn zbiorów, czyli zbiór par.

Wszystkie powyższe napisy są w sensie teorii mnogości.

Część druga tego zadania polega na pokazaniu, że bijekcja dla punktu pierwszego jest naturalna. Już tłumaczę o co chodzi.

Weźmy dwie rodziny zbiorów A, B, C oraz A', B', C'.

Mamy dwie bijekcje:

$$p : ((A \times B) \to C) \to (A \to (B \to C)))$$
$$p' : ((A' \times B') \to C') \to (A' \to (B' \to C')))$$

Załóżmy że mamy bijekcja między zbiorami z primami i bez primów (uwaga na kolejność primów!):

$$a : A' \to A$$
$$b : B' \to B$$
$$c : C \to C'$$

Jest oczywista konstrukcja bijekcji między $A' \times B'$ i $A \times B$ (bierzemy funkcję a na pierwszej współrzędnej i b nar drugiej). Jest też analogiczna konstrukcja bijekcji między $A \to C$ oraz $A' \to C'$. Mianowicie trzeba wziąć $f \in A \to C$ i zwrócić $c \circ f \circ a$. Zwróć uwagę na to że potrzebowaliśmy by funkcje a i c szły w odwrotnym kierunku (matematycy mówią że strzałka jest kontra-wariantna ze względu na argument i zgodnie-wariantna ze względu przeciwdziedzinę).

Stosując te konstrukcję wielokrotnie możemy dostać bijekcję między: $(A \times B) \to C$ oraz $(A' \times B') \to C'$. Oznaczmy ją k.

Możemy też dostać bijekcję między $A \to (B \to C))$ oraz $A' \to (B' \to C'))$. Oznaczmy ja l.

W naturalności chodzi o to by pokazać że: $p' \circ k = l \circ p$. Wskazówka jest taka, że trzeba uwierzyć, że jedyna trudność w tym zadaniu jest mentalna - przebicie się przez wysoki poziom abstrakcji.

Pogadanka o funkcjach

W (chyba) każdym języku programowania można zdefiniować funkcję. Ja będę używał notacji:

nazwaFunkcji (argument1 : jegoTyp) (argument2 : jegoTyp) : typWyniku.

Jakiego typu jest funkcja pobierająca coś typu A i zwracająca coś typu B? Z pewnością nie jest to ani typ A ani typ B. Typ takiej funkcji będę oznaczał przez $A \to B$.

Zastanówmy się jednak jak typować funkcję dwuargumentową. Wydaje się, że potrzeba jakaś bogatszej notacji. Jednak my ominiemy problem następująco: tak naprawdę funkcji dwuargumentowych typu $A \times B \to C$ nie ma. Są tylko funkcje, które pobierają jeden argument i zwracają funkcję typu $(B \to C)$, tj. funkcję która pobierze drugi argument i zwróci całościowy wynik (por. podpunkt 1 w zad 2).

Zad 3. Dziwne wektory

Dana jest biblioteka implementująca wektory. Udostępnia ona dwa interfejsy - jeden służący do tworzenia i czytania obiektów, a drugi do ich modyfikowania (i tworzenia podstawowych obiektów). Ty jednak dysponujesz tylko tym drugim. Masz do dyspozycji jakiś tajemniczy typ *wektor(n)* i jedyne co możesz z nim robić, to używać dwóch funkcji, które na nim operują:

1. *opakuj (a : A) (n : int) : wektor(A)* zwraca wektor długości n przechowujący rzeczy typu A (czyli obiekt typu wektor(A)), który na wszystkich współrzędnych ma wartość a.
2. *zastosuj (f : wektor(A->B)) (w : wektor(A)) : wektor(B)* Ta funkcja jest chytra. Pobiera ona dwa wektory tej samej długości. Pierwszy z nich zawiera funkcje typu $A \to B$ (pobierającą A i zwracającą B), drugi z nich zawiera obiekty typu A. Wynikiem zastosuj jest wektor, na którego i-tej pozycji jest wynik zastosowania i-tej funkcji z f do i tej pozycji w.

Zaimplementuj następujące funkcje na wektorach:

1. dodawanie dwóch wektorów po współrzędnych. Wsk. Użyj *zastosuj* dla typu $B = (C \to D)$.
2. funkcję, która dostawszy wektory $w^1, w^2, \cdots w^k$ (dostaje je np. w tablicy, liście, czy inne zwykłej strukturze danych) ma zwrócić wektor tv, zawierający średnią kwadratową liczoną po współrzędnych. Mówić wprost ma zachodzić (zwróć uwagę że wszystkie indeksy dolne są takie same):

$$v_l = \sqrt{\frac{\sum_{i=0}^{k} (w_l^i)^2}{n}}$$

3. Czy umiesz zrobić powyższy podpunkt, zakładając globalnie ustalone $k = 7$, tak by wywołać tylko k razy funkcję zastosuj? A jeśli k nie jest ustalone? Jaką funkcjonalność powinien mieć system typów by to się udało?

W tym zadaniu polecam pisać kod w pseudokodzie - nie jest moim celem sprawdzenie, czy umiesz używać wskaźników na funkcje i innych zaawansowanych funkcjonalności w Twoim ulubionym języku (być może w ogóle ich w nim nie ma, np. Pascal). Warto jednak zamieścić krótką dyskusję, jakich funkcjonalności używasz w pseudokodzie i jaka jest ich składnia. Jeśli jednak wskaźniki na funkcje i lambda abstrakcje (Java 8, C++11) nie są Ci obce, to próba implementacji powyższego zadania może być pouczającym ćwiczeniem (nieocenianym w tej kwalifikacji).

Zad 4. Logika i inne przypadłości

Na początku znowu będzie trochę matematyki. Udowodnij następujące tożsamości (ϕ, ψ, α - dowolne zdanie logiczne):

1. $\phi \to (\psi \to \alpha)$ jest równoważne $(\phi \wedge \psi) \to \alpha$. Dalej zakładam że implikacja wiąże w lewo (czyli mogę w pierwszej części pominąć nawiasy).
2. $\forall_\alpha (\psi \to \alpha) \to (\phi \to \alpha) \to \alpha$ jest równoważne $\psi \vee \phi$
3. Powyższa formuła wyraziła alternatywę tylko przy pomocy implikacji i uniwersalnej kwantyfikacji. Podaj analogiczny sposób wyrażenia koniunkcji tylko przy pomocy implikacji i uniwersalnej kwantyfikacji.
4. Tutaj przyjmuję że w ψ występuje x jako wolna zmienna.
   $\forall_\alpha (\forall_x \psi(x) \to \alpha) \to \alpha$ jest równoważne $\exists_x \psi(x)$

Pogadanka o polimorfiźmie

Wyobraź sobie funkcję, która bierze x i zwraca x (matematycy nazywają taką funkcję identycznością). Jaki typ przypisać takiej funkcji? Może to być równie dobrze Int $\to$ Int, jak i Double $\to$ Double, jak i cokolwiek innego. Typ polimorficzny to typ używający kwantyfikatora ogólnego: $\forall_\alpha \alpha \to \alpha$. Taki typ aż sam się prosi by przypisać go identyczności. Znaczenie kwantyfikatora ogólnego opisują dwie zasady:

- Reguła wprowadzania. Aby móc powiedzieć że funkcja ma typ $\forall_x \phi(x)$ (ϕ to typ w którym może występować literka x), musi zachodzić że dla dowolnego typu β funkcja ma typ $\phi[x = \beta]$ (podstawienie pod x typu β).
- Reguła użycie. Aby użyć funkcji polimorficznej (zdjąć kwantyfikator) należy podać jej typ, którym ma się ona ukonkretnić. Czyli polimorficzna identyczność pobiera dwa argumenty: najpierw bierze jakiś typ, a potem jakiś element tego typu.

Przykład widzieliśmy powyżej: funkcja identyczność ma zarazem typ Int $\to$ Int, jak i String $\to$ String, jak i każdy analogiczny. Możemy więc powiedzieć, że ma typ $\forall_\alpha \alpha \to \alpha$. W tym przykładzie $\phi := \alpha \to \alpha$.

Zad 5. Polimorfizm

W matematyce jest operacja rozłącznej sumy zbiorów oznaczanej $A \oplus B$. Działa ona tak, że bierze dwa zbiory A i B i zwraca zbiór, który składa się z rozłącznych kopii A i B. Istnieje naturalna operacja włożenia - włożenie lewe : WŁÓŻLEWO : $A- > A \oplus B$ i analogiczne prawe.

Analogiczna operacja ma sens w językach programowania: można chcieć np. mieć obiekt typu Int *vee* String - czyli coś co jest albo liczbą, albo napisem. Taka alternatywa na typach zachowuje się dokładnie jak suma rozłącza - są naturalne włożenia. Analogia dotyczy sumy rozłącznej, a nie zwykłej - bowiem w typ Int $\vee$ Int można włożyć liczbę 5 na dwa sposoby (na lewo i na prawo) i wyjdą różne rzeczy.

W tym zadaniu przypuśćmy, że mamy dwie funkcje - jedna umie wypisać Int, a druga String. Obie te funkcje zwracają void/unit lub coś podobnego.

Nasz język nie dysponuje alternatywą na typach bezpośrednio - my jednak będziemy chytrzy i zdefiniujemy sobie notację (por zadanie 4.2):

$$\phi \vee \psi := \forall_\alpha (\psi \to \alpha) \to (\phi \to \alpha) \to \alpha$$

1. Opisać jak działa funkcja wkładania w tak rozumianą alternatywę - tj funkcja WŁÓŻLEWO która bierze coś typu ϕ i zwraca coś typu $\phi \vee \psi$. WŁÓŻLEWO powinno być typu $\forall_{\phi,\psi} \phi \to \phi \vee \psi$.

2. Opisać jak działa funkcja, która bierze coś typu Int $\vee$ String i wypisuje to co dostała na ekran.

3. Czym różni się typ $\forall_\alpha(\alpha \to \alpha) \to Int$ od typu $(\forall_\alpha \alpha \to \alpha) \to Int$? Wskazówka - spróbuj wymyślić funkcję każdego z tych typów. Warto nawiązać do zad 4.4.

Zad 6. Magiczne wróżki kontratakują

W tym zadaniu będziemy wczytywać i przekazywać grafy. Przyjmujemy, że graf nie jest dany w formie tekstowe na wejściu, lecz jako funkcja, która mówi czy między dwoma wierzchołkami jest krawędź czy nie. A tak naprawdę to jako parę - liczba wierzchołków w grafie i powyższą funkcję (na liczbach naturalnych).

Mamy do dyspozycji magiczną wróżkę, która dostawszy graf (jako funkcję) umie odpowiedzieć czy ten graf jest pełnym drzewem binarnym (doprecyzowanie: *skierowane* od ojców do synów i bez wyróżnionej kolejności dzieci). Wróżka robi to bardzo szybko - w czasie stałym, niezależnie od wielkości grafu.

Nas jednak nie interesuje czy graf, który dostaliśmy od króla (jako funkcję działającą w czasie stałym) jest pełnym drzewem binarnym - król zawsze daje poddanym pełne drzewa binarne, a królewskiego słowo możemy być pewni. Interesuje nas jednak, czy konkretne dwa wierzchołki w tym grafie są na tej samej wysokości czy nie. Otrzymany graf jest niebotycznie wielki, więc aby poznać odpowiedź na frapujące nas pytanie chcemy skorzystać jakoś ze zdolności magicznej wróżki.

Jak to zrobić by działać w czasie stałym?

Wzmocnienie tezy będące jednocześnie podpowiedzią: można zadać wróżce tylko jedno pytanie, i to o funkcji, która działa w czasie stałym.

Changelog

- 0.4.1 - poprawiono typy w zad 3 (zamieniono B na A).
- wcześniej - dodano podpunkt do zad 1, rozwinięto opis polimorfizmu, poprawiono literówki.