Programowanie funkcyjne 2016

Grupa kzi
Lista 2
(18.10.2016)

Uwaga†: Jeśli zadanie jest niedospecyfikowane, to braki w specyfikacji można uzupełnić w dowolny sensowny(!) sposób.

Wstęp do zadań w OCamlu

Na tej liście powinniśmy przećwiczyć sobie pisanie funkcji ogonowych. Do tego celu mogą posłużyć drzewa binarne, które implementowaliśmy na poprzedniej liście. Niestety tamta implementacja kiepsko nadaje się do pokazania, że funkcje nieogonowe są złe. Problemem jest nasze fałszywe założenie o tym, że typ `int` zawiera reprezentacje wszystkich pozycji wierzchołków w takich drzewach. Dlatego tą listę zaczniemy od zaprogramowania biblioteki do obsługi dowolnie dużych liczb naturalnych, które już to założenie spełniają. Część tej biblioteki jest dostępna w pliku `nat.ml` na stronie pracowni dostępnej z mojej strony domowej.

Zadania w OCamlu

Zakładamy, że mamy zrobioną Listę 1 i możemy korzystać z rozwiązań znajdujących się tam zadań. Jeśli ktoś nie zrobił zadań z Listy 1, niech weźmie rozwiązania od kolegi/koleżanki.

1. W przypadku porównywania równych liczb naturalnych udostępniona implementacja funkcji `nless_eq` skanuje te liczby dwa razy. Wzorując się na funkcji `nless`, popraw tą funkcję, tak żeby skanowała swoje argumenty tylko raz. Czy ta funkcja jest ogonowa? (4 pkt)

2. Na bazie funkcji `nadd` napisz funkcję `nsub : nat -> nat -> nat`, która odejmuje dwie liczby naturalne. Odjęcie liczby większej od mniejszej powinno zwracać zero. Podobnie jak w poprzednim zadaniu ta funkcja powinna przebiegać po swoich argumentach tylko raz. (Czyli dodatkowe ich porównanie nie wchodzi w grę.) Czemu ta funkcja powinna być ogonowa? (3 pkt)

3. Popraw zadania z Listy 1 tak, żeby pozycje były reprezentowane przez wartości typu `nat`. (1 pkt)

4. Napisz funkcje `sum_nt` i `sum_t` typu `int flbt -> int`, które liczą sumę wszystkich wartości przechowywanych w skończonym drzewie. Pierwsza z tych funkcji powinna być nieogonowa, a druga ogonowa. (4 PKT)

5. Poniższa funkcja `t_right1` generuje pewne drzewa. Złożoność pobrania wartości wierzchołka takiego drzewa jest równa $\mathcal{O}(d)$, gdzie d jest argumentem tej funkcji. Popraw tą funkcję tak, żeby ta złożoność była liniowa względem głębokości wierzchołka.

```
let rec only_right p =
    p = root_pos || nmod2 p = 1 && only_right (ndiv2 p)

let t_right1 (d :  int) :  int flbt =
    (fun p -> if nless p (nilsl (i2n 1) d) && only_right p
              then 1
              else raise NIT)
              (* zakladamy, ze mamy zdefiniowany taki wyjatek;
                 NIT oznacza Not In Tree, analogicznie do
                 NIL, który oznacza Not In List *)
```

(3 PKT)

6. Pokaż, że niektóre funkcje powinny być ogonowe. W tym celu napisz dwa programy, które za pomocą `t_right1` generują pewne drzewo i wypisują na standardowe wyjście sumę przechowywanych przez nie wartości. Pierwszy program powinien liczyć tą sumę za pomocą `sum_nt`, a drugi za pomocą `sum_t`. Skompiluj te programy do bytecodu OCamla (polecenie `ocamlc`) i wykonaj je (polecenie `ocamlrun`) z na tyle małym stosem, żeby zwróciły różne wyniki (i policzyły się przed końcem zajęć). (3 PKT)

Wstęp do zadań w Haskellu

(Mówiąc bardzo nieformalnie) automat na słowach to taka maszynka, która czyta słowa złożone z liter pewnego alfabetu i po przeczytaniu każdej litery zmienia zamontowany w niej stan. Automat rozpoczyna pracę w pewnym początkowym stanie i gdy skończy pracę, mówi „tak", jeśli stan, w którym skończył należy do pewnego zbioru stanów akceptujących, i „nie" w.p.p.

Na potrzeby tej listy zakładamy, że automaty czytają słowa od lewej do prawej i ich stanami są wartości typu `Int`.

W Haskellu za definicję automatu możemy przyjąć krotkę typu

```
type Aut = ([Char], Int, Int -> Char -> Int, Int -> Bool),
```

gdzie

- pierwszym elementem jest alfabet,
- drugim jest stan początkowy,

- trzecim jest funkcja σ taka, że jeśli $\sigma\ i\ c = j$, to automat po przeczytaniu litery c będąc w stanie i zmieni stan na j,
- czwartym jest funkcja α taka, że $\alpha\ i$ stwierdza, czy i należy do zbioru stanów akceptujących.

Zadania w Haskellu

7. Napisz funkcję `aut_words`, która zaaplikowana do definicji automatu zwraca listę typu `[(String,Bool)]` taką, że zrzutowana na pierwszą oś daje listę wszystkich słów nad alfabetem automatu, i jeśli para (w, b) jest na tej liście, to b jest odpowiedzią automatu dla słowa w. Np. dla definicji automatu

   ```
   (['a', 'b'], 0, \_ c -> if c == 'a' then 0 else 1, \i-> i == 1)
   ```

   zwracaną listą może być

   ```
   [("",false),("a",false),("b",true),("aa",false),("ab",true),
           ("ba",false),("bb",true),("aaa", false),...].
   ```

   Zakładając, że n pierwszych elementów tej listy zostało już wymuszone, wymuszenie kolejnego powinno być obliczane w stałym czasie. Czy ta funkcja może być ogonowa? Jeśli nie, to dlaczego?

   Korzystając z powyższej funkcji napisz funkcję `accepted_words`, która zaaplikowana do automatu zwraca listę wszystkich słów, dla których automat mówi „tak". (8 pkt)