Programowanie funkcyjne 2016

Grupa kzi
Lista 5
(22.11.2016)

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

Zadania w OCamlu

1. Zdefiniuj typ `'a ltree` dla leniwych drzew, w których wierzchołki mogą mieć dowolną ilość dzieci. Skorzystaj z biblioteki Lazy. Każdy wierzchołek powinien przechowywać wartość typu `'a`. Wszystkie wierzchołki, poza korzeniem, i wszystkie krawędzie w drzewie powinny być odroczone. W szczególności, ani zdefiniowanie dowolnego nieskończonego drzewa, ani pobranie któregokolwiek z jego wierzchołków nie powinno się zapętlać, chyba, że sama procedura wyliczająca wierzchołki się zapętla. (Dobrze się przyjrzyj swojemu rozwiązaniu, bo tego zadania nie będzie można poprawiać.) (3 ptk)

2. Zdefiniuj funkcję `ltree2llist: 'a ltree -> 'a llist`, która "wylistowuje" wszystkie elementy leniwego drzewa. `'a llist` to typ zdefiniowany na wykładzie – wersja z użyciem biblioteki Lazy (jest na slajdach). (5 ptk)

Definicja. Niech $R = \{(w_1, v_1), \ldots, (w_n, v_n)\}$ będzie zbiorem par napisów. Nazwijmy go sobie zbiorem reguł przepisywania. Proces przepisywania napisów względem R to ciąg (skończony albo nie) napisów $u_1, u_2, \ldots$, taki że dla każdych dwóch sąsiednich elementów u_i, u_{i+1} istnieją takie k, u', u'', że $u_i = u'w_ku''$ i $u_{i+1} = u'v_ku''$.

3. Zdefiniuj funkcję `rewrite_tree : (string * string) list -> string -> string ltree`, która zaaplikowana do $[(w_1, v_1); \ldots; (w_n, v_n)]$, u_1 zwraca takie leniwe drzewo, że

   > ciąg u_1, u_2, ... jest procesem przepisywania napisów względem $\{(w_1, v_1), \ldots, (w_n, v_n)\}$

   wtw

   > u_1, u_2, ... jest ścieżką w drzewie (dokładniej, ścieżką etykiet).

Inaczej mówiąc, `rewrite_tree` buduje drzewo wszystkich procesów przepisywania napisów względem $\{(w_1, v_1), \ldots, (w_n, v_n)\}$ startujących z u_1. (12 PTK)

4. Wykorzystaj wcześniejsze zadania i zdefiniuj funkcję, która stara się rozwiązać problem słów dla semi systemów Thue'go. Tj. twoja funkcja zaaplikowana do listy reguł przepisywania R i napisów u, u' powinna zwracać `true` wtw gdy istnieje proces przepisywania napisów względem R startujący z u i kończący się na u'. Jeśli takiego procesu nie ma funkcja może zwrócić `false`, zapętlić się, albo rzucić wyjątek, że zabrakło pamięci. (2 PTK)

   Uwaga: Ten problem jest nierozstrzygalny, więc nie da się napisać programu, który zawsze potrafi go rozwiązać. Mimo wszystko czasem możemy chcieć spróbować. W takim wypadku funkcja, która się zapętla, kiedy nie może znaleźć rozwiązania, nie jest zbyt dobra i powinniśmy użyć jakieś wersji ograniczonego BFS'a.