Programowanie funkcyjne 2014
Lista 2

kzi

21 października 2014

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

Zadania w Haskell'u

1. Zdefiniuj funkcję `gen_int_list :: System.Random.RandomGen g => g -> Int -> Int -> Int -> Int -> Int -> [Int]`, która zaaplikowana do parametrów *seed* m n_1 n_2 *spec* k zwróci pseudolosową listę długości m wygenerowaną z ziarna *seed* złożoną z liczb $i \in [n_1, n_2] \cup \{spec\}$, w której oczekiwana odległość pomiędzy dwoma wystąpieniami znaku *spec* jest równa k. `(3pkt)`

   Uwaga: W tym zadaniu nie trzeba rozumieć, co znaczy `System.Random.RandomGen g =>` i można myśleć, że `gen_int_list` jest typu `RandomGen -> Int -> Int -> Int -> Int -> Int -> [Int]`, gdzie `RandomGen` to typ ziaren do generowania liczb pseudolosowych.

2. Zdefiniuj typ `Aut a = a -> Int -> Int`.

Definicja. Mówimy, że automat *aut* `:: Aut a` akceptuje ciąg $x_1, \ldots, x_n$ wartości typu `a` wtw gdy *aut* x_1 $1 = q_1$, *aut* x_2 $q_1 = q_2$, ..., *aut* x_n $q_{n-1} = 0$ dla pewnych nieujemnych $q_1, \ldots, q_{n-1}$.

Intuicja jest taka. Nasze automaty to maszynki, które idą sobie po liście od lewej do prawej i zmieniają jakiś stan w zależności od czytanych kolejno elementów. Stan 1 jest stanem początkowym. Przejście do stanu 0 na końcu listy oznacza akceptację tej listy. Przejście do stanu ujemnego oznacza, że nie da się zaakceptować listy, po której idziemy.

2.*a*) Zdefiniuj nieogonowo funkcję `matches_count_nt :: Aut a -> [a] -> Int`, która zaaplikowana do *aut* l zwraca ilość podlist listy l, które są akceptowane przez automat *aut* — przy czym podlisty rozpoczynające się na różnych pozycjach liczymy jako różne (nawet jeśli ich zawartość jest taka sama). `(5pkt)`

Przykład. Dla automatu aut_{ma}, który akceptuje wszystkie listy postaci $(ma)+$ i nic więcej, to jest:

$$ma,\ mama,\ mamama,\ \ldots$$

`matches_count_nt` zaaplikowane do aut_{ma} i listy *tamamama* powinno zwrócić 6, a do listy *tamabamama*, 4. (Automat aut_{ma} będzie zamieszczony na mojej stronie.)

2.*b*) Zdefiniuj funkcję `matches_count_t` , która robi to samo co poprzednia, tyle że ogonowo. `(3pkt)`

2.*c*) Wygeneruj skończoną losową listę, taką że `matches_count_nt` się na niej wywali, a `matches_count_t` nie. `(1pkt)`

Zadania w OCaml'u

3. Zrób zadania 1, 2 w OCaml'u wykorzystując standardowe listy i bibliotekę Random. `(7pkt)`

Zadania "i w jednym i w drugim"

4. Wygeneruj duże losowe listy i zaaplikuj do nich funkcje `matches_count...` . Porównaj wydajność Haskell'a i OCaml'a. `(1pkt)`

Ciąg dalszy nastąpi

Wskazówka: Można korzystać z funkcji w bibliotekach Data.List (Haskell) i List (OCaml), np. `map`, `filter`, `length`, `fold...`, *itp.*