Programowanie funkcyjne 2014
Lista 3

kzi

28 października 2014

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

Zadania w OCaml'u

Semnatyka funkcji w zadaniach 1-7 jest taka sama jak funkcji o tych samych nazwach w Haskell'u, tyle że gorliwa. We wszytkich zadaniach nie wolno korzystać z funkcji bibliotecznych na listach (tylko dopasowanie wzorca i konstruktory).

1. Zdefiniuj ogonowo funkcję `foldl : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`.
   Funkcja `foldl` od $f\ a\ [e_1, e_2, \ldots, e_n]$ zwraca $f\ (\ldots(f\ (f\ a\ e_1)\ e_2)\ldots)\ e_n$. `(3pkt)`

2. Zdefiniuj funkcję `foldr : ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b`.
   Funkcja `foldr` od $f\ a\ [e_1, e_2, \ldots, e_n]$ zwraca $f\ e_1\ (f\ e_2\ (\ldots(f\ e_n\ a)\ldots))$. `(3pkt)`

3. Zdefiniuj funkcję `unfoldr : ('a -> ('a * 'b) option) -> 'a -> 'b list`.
   Jeśli $f\ a =$ `Some` (a_1, b_1), $f\ a_1 =$ `Some` (a_2, b_2), $\ldots$, $f\ a_n =$ `None`, to `unfoldr` od $f\ a$ zwraca $[b_1, \ldots, b_n]$, wpp nie zwraca nic (może rzucać wyjątek albo się zapętlać). `(3pkt)`

Wszystkie funkcje od tego miejsca definiuj za pomocą `fold`-ów.

4. Zdefiniuj funkcję `reverse : 'a list -> 'a list`,
   która odwraca listę. `(1pkt)`

5. Zdefiniuj funkcję `map : ('a -> 'b) -> 'a list -> 'b list`.
   Funkcja `map` od $f\ [e_1, e_2, \ldots, e_n]$ zwraca $[f\ e_1, f\ e_2, \ldots, f\ e_n]$. `(2pkt)`

6. Zdefiniuj funkcję `all : ('a -> bool) -> 'a list -> bool`,
   która sprawdza, czy predykat zachodzi dla wszystkich elementów listy. `(1pkt)`

7. Zdefiniuj funkcję `any : ('a -> bool) -> 'a list -> bool`,
   która sprawdza, czy predykat zachodzi dla jakiegoś elementu listy. `(1pkt)`

   a) `+1pkt` jeśli `any` zaaplikowana do predykatu p i listy $[e_1, \ldots, e_n]$ będzie miała złożoność

   $$O\left(min_i\left(p\ e_i \vee i = n\right)\right).$$

8. Zdefinuj typ `aritm_literal` z zeroarnymi konstruktorami `Neg`, `Add`, `Sub`, `Mul`, `Div`, `Pow` i jednoarnym `Num` opakowywującym `float`-a. `(1pkt)`

9. Zdefiniuj funkcję `eval_rpn : aritm_literal list -> float`, która bierze listę reprezentującą wyrażenie w odwrotnej notacji polskiej (Reverse Polish Notation) i zwraca wynik obliczenia tego wyrażenia. Wartości `Neg`, `Add`, `Sub`, `Mul`, `Div`, `Pow` reprezentują odpowiednio $\sim$, $+$, $-$, $\cdot$, $:$, $\wedge$, gdzie $\sim$ to unarny minus, a $\wedge$ to potęga. Wartość postaci `Num x` reprezentuje liczbę `x`.

   Przykład: Lista `[Num 2.0; Neg; Num 3.0; Num 0.5; Mul; Add; Num (-1.0); Pow]` reprezentuje wyrażenie

   $$2 \quad \sim \quad 3 \quad \frac{1}{2} \quad \cdot \quad + \quad -1 \quad \wedge$$

   które oblicza się do -2.

   Zadbaj o to, żeby `eval_rpn` rzucała wyjątki `TooFewOperations`, `TooManyOperations`, gdy wyrażenie ma nieprawidłową składnię (np. `[Num 2.0; Neg; Num 3.0]`) i `IllegalOperation (`op`, `a`, `b`)`, gdy próbujemy wykonać operację op na niedozwolonych argumentach a b (np. $op =$ `Pow`, $a = -2.0$, $b = 0.5$). `(4pkt)`