Programowanie Funkcyjne 2018

Lista zadań nr 1

10 października 2018

Zadanie 1 (2p). Jaki jest typ wyrażenia `fun x -> x`? Napisz wyrażenie anonimowe reprezentujące funkcję identycznościową, którego typem w OCamlu jest typ `int -> int`. Napisz wyrażenia typu `'a -> 'b` oraz typu `('a -> 'b) -> ('c -> 'a) -> 'c -> 'b`.

Zadanie 2 (2p). Napisz dwie wersje funkcji (w tym jedną za pomoca rekursji ogonowej), która oblicza n-ty wyraz ciągu zdefiniowanego wzorem:

$$a_0 = 0$$
$$a_{n+1} = 2a_n + 1$$

a następnie porównaj ich działanie dla dużych n.

Zadanie 3 (2p). Napisz funkcję definiującą złożenie dwóch funkcji oraz funkcję iterowania wywołania funkcji wykorzystującą funkcję złożenia. Za pomocą iteracji zdefiniuj mnożenie (skorzystaj z faktu, ze operator infiksowy `+` może być potraktowany jak funkcja dwóch argumentów) i potęgowanie (przy okazji zdefiniuj odpowiedni operator infiksowy).

Zadanie 4 (8p). Strumień (tj. nieskończony ciąg) elementów typu `t` możemy reprezentować za pomocą funkcji typu `int -> t` w taki sposób, że dla dowolnej takiej funkcji `s`, `s 0` oznacza pierwszy element strumienia, `s 1` następny, itd. Używając powyższej reprezentacji zdefiniuj następujące funkcje na strumieniach (tam gdzie to możliwe, funkcje te powinny być polimorficzne, tj. powinny działać na strumieniach o dowolnych elementach):

- `hd`, `tl` — funkcje zwracające odpowiednio głowę i ogon strumienia
- `add` — funkcja dodawania zadanej stałej do każdego elementu strumienia i zwracająca powstały w ten sposób strumień
- `map` — funkcja, która dla zadanej operacji 1-argumentowej przetwarza elementy zadanego strumienia za pomocą tej operacji i zwraca powstały w ten sposób nowy strumień (tak, jak `map` na listach skończonych)
- `map2` — jak wyżej, ale dla zadanych: funkcji 2-argumentowej i 2 strumieni
- `replace` — funkcja, która dla zadanego indeksu `n`, wartości `a` i strumienia `s` zastępuje co `n`-ty element strumienia `s` przez wartość `a` i zwraca powstały w ten sposób strumień
- `take` — funkcja, która dla zadanego indeksu `n` i strumienia `s` tworzy nowy strumień złożony z co `n`-tego elementu strumienia `s`
- `scan` — funkcja, która dla zadanej funkcji `f: 'a -> 'b -> 'a`, wartości początkowej `a: 'a` i strumienia `s` elementów typu `'b` tworzy nowy strumień, którego każdy element jest wynikiem „zwinięcia" początkowego segmentu strumienia `s` aż do bieżącego elementu włącznie za pomocą funkcji `f`, tj. w strumieniu wynikowym element o indeksie `n` ma wartość `(f (. . . (f (f a (s 0)) (s 1)) . . . ) (s n))`

- `tabulate` — funkcja tablicowania strumienia, której wartością powinna być lista elementów strumienia leżąca w zadanym zakresie indeksów.

Zdefiniuj przykładowe strumienie i przetestuj implementację. W definicji funkcji `tabulate` wykorzystaj możliwość definiowania parametrów opcjonalnych dla funkcji (niech początek zakresu indeksów będzie opcjonalny i domyślnie równy `0`).

Przykład. Pisząc `let f ?(x=0) y = x + y` deklarujemy, że pierwszy argument funkcji `f` o etykiecie `x` jest opcjonalny, a jego wartość domyślna wynosi `0`. Funkcję `f` można zatem wywołać za pomocą wyrażenia `f 3` `(= 3)` lub jawnie podając wartość parametru opcjonalnego, za pomocą składni `f ~x:42 3 (= 45)`.

Zadanie 5 (3p). Okazuje się że wartości boole'owskie można reprezentować przy pomocy funkcji polimorficznych typu `'a -> 'a -> 'a`. Zdefiniuj wartości o poniższych sygnaturach odpowiadające prawdzie i fałszowi, operatorom koniunkcji i alternatywy, a także funkcje konwersji między naszą reprezentacją a wbudowanym typem wartości logicznych.

- `ctrue, cfalse: 'a -> 'a -> 'a`
- `cand, cor: ('a -> 'a -> 'a) -> ('a -> 'a -> 'a) -> 'a -> 'a -> 'a`
- `cbool_of_bool: bool -> 'a -> 'a -> 'a`
- `bool_of_cbool: (bool -> bool -> bool) -> bool`

Zastanów się czy (i dlaczego) typy znalezione przez OCamla mogą się różnić od podanych powyżej.

Wskazówka: Zastanów się ile funkcji typu `'a -> 'a -> 'a` które nie wywołują innych funkcji i nie wykonują efektów ubocznych (tj. zawsze kończą działanie, nie wywołują wyjątków, nie używają wejścia/wyjścia, itp.) istnieje.

Zadanie 6 (3p). Podobnie jak w poprzednim zadaniu, jako funkcje polimorficzne możemy reprezentować również liczby naturalne. Ideą takiej reprezentacji jest żeby liczbie n odpowiadała funkcja $f, x \mapsto f^n(x)$. W tej reprezentacji typem liczb naturalnych jest `('a -> 'a) -> 'a -> 'a`. Zdefiniuj liczbę zero, operację następnika, operacje dodawania i mnożenia, funkcję sprawdzającą czy dana liczba jest zerem, a także konwersje między naszą reprezentacją a wbudowanym typem liczb całkowitych (nie przejmuj się liczbami ujemnymi).

- `zero : ('a -> 'a) -> 'a -> 'a`
- `succ : (('a -> 'a) -> 'a -> 'a) -> ('a -> 'a) -> 'a -> 'a`
- `add, mul : (('a -> 'a) -> 'a -> 'a) -> (('a -> 'a) -> 'a -> 'a) -> ('a -> 'a) -> 'a -> 'a`
- `isZero : (('a -> 'a) -> 'a -> 'a) -> 'a -> 'a -> 'a`
- `cnum_of_int : int -> ('a -> 'a) -> 'a -> 'a`
- `int_of_cnum : ((int -> int) -> int -> int) -> int`

Zastanów się czy (i dlaczego) typy niektórych z powyższych funkcji znalezione przez algorytm inferencji mogą się różnić od podanych powyżej.

Uwaga: Jeśli uważałeś na algebrze, możesz zauważyć że powyższa reprezentacja jest ściśle związana z twierdzeniem Cayleya o funkcyjnej reprezentacji monoidów.