Paradygmaty programowania - ćwiczenia
Lista 4

W poniższych zadaniach funkcje należy napisać w obu językach: OCaml i Scala (wykorzystując mechanizm dopasowania wzorców!).

1. Zdefiniuj funkcje *curry3* i *uncurry3*, przeprowadzające konwersję między zwiniętymi i rozwiniętymi postaciami funkcji od trzech argumentów. Podaj ich typy.

2. Przekształć poniższą rekurencyjną definicję funkcji s*umProd*, która oblicza jednocześnie sumę i iloczyn listy liczb całkowitych na równoważną definicję nierekurencyjną z jednokrotnym użyciem funkcji *fold_left* (Scala – *foldLeft lub /:*).

OCaml

```
let rec sumProd l =
  match l with
     h::t -> let (s,p)= sumprod t
             in (h+s,h*p)
   | []   -> (0,1);;
```

Scala

```
def sumProd(xs:List[Int]):(Int,Int) =
   xs match {
    case h::t => {val (s,p)=sumProd(t)
                  (h+s,h*p)
                 }
     case Nil  => (0,1)
   }
```

3. Poniższe dwie wersje funkcji *quicksort* działają niepoprawnie. Dlaczego?

OCaml

```
a) let rec quicksort = function
          []  -> []
       |  [x] -> [x]
       |  xs  -> let small = List.filter (fun y -> y < List.hd xs ) xs
                 and large = List.filter (fun y -> y >= List.hd xs ) xs
                 in quicksort small @ quicksort large;;

b) let rec quicksort' = function
            [] -> []
     |  x::xs -> let small = List.filter (fun y -> y < x ) xs
                 and large = List.filter (fun y -> y > x ) xs
                 in quicksort' small @ (x :: quicksort' large);;
```

4. Zdefiniuj funkcje sortowania

a) przez wstawianie z zachowaniem stabilności i złożoności $O(n^2)$
insertionsort : (‘a->‘a->bool) -> ‘a list -> ‘a list .

b) przez łączenie (scalanie) z zachowaniem stabilności i złożoności $O(n \lg n)$
mergesort : (‘a->‘a->bool) -> ‘a list -> ‘a list .

Pierwszy argument jest funkcją, sprawdzającą porządek. Podaj przykład testu sprawdzającego stabilność.

Uwaga! Przypominam, że funkcje *List.append* i *List.length* mają złożoność liniową!