

Paradygmaty programowania - ćwiczenia

Lista 5

Na wykładzie zostały zdefiniowane listy leniwe.

OCaml: `type 'a llist = LNil | LCons of 'a * (unit -> 'a llist);;`

1. (OCaml i Scala)

Zdefiniuj funkcję `repeat : int -> 'a llist -> 'a llist`, która dla danej dodatniej liczby całkowitej k i listy leniwej (strumienia w Scali) $[x_0, x_1, x_2, x_3, \dots]$ zwraca listę leniwą (strumień w Scali), w której każdy element jest powtórzony k razy, np.

`repeat 3 [x0, x1, x2, x3, ...] => [x0, x0, x0, x1, x1, x1, x2, x2, x2, x3, x3, x3, ...]`

Uwaga. Dla zwiększenia czytelności zastosowano tu notację dla zwykłych list.

2. Zdefiniuj (w inny sposób, niż na wykładzie) leniwą listę liczb Fibonacciego

a) (OCaml) `lfib : int llist`

b) (Scala) `lfib : Stream[Int]`.

3. (OCaml i Scala) Polimorficzne leniwe drzewa binarne można zdefiniować następująco:

OCaml: `type 'a lBT = LEmpty | LNode of 'a * (unit -> 'a lBT) * (unit -> 'a lBT);;`

Scala:

```
sealed trait lBT[+A]
case object LEmpty extends lBT[Nothing]
case class LNode[+A](elem:A, left:()=>lBT[A], right:()=>lBT[A]) extends lBT[A]
```

a) Napisz funkcję `lTree`, która dla zadanej liczby naturalnej n konstruuje nieskończone leniwe drzewo binarne z korzeniem o wartości n i z dwoma poddrzewami `lTree(2*n)` oraz `lTree(2*n+1)`. To drzewo jest przydatne do testowania funkcji z następnego podpunktu.

b) Napisz funkcję, tworzącą leniwą listę w OCamlu (strumień w Scali), zawierającą wszystkie wartości węzłów leniwego drzewa binarnego.

Wskazówka: zastosuj obejście drzewa wszerz, reprezentując kolejkę jako zwykłą listę.