Laboratorul 4: Funcția 'foldr'

RECOMANDARE Înainte de a începe să lucrați exercițiile din acest laborator finalizați exercițiile din laboratoarele precedente.

I. Funcția `foldr`.

Funcția `foldr` este folosită pentru agregarea unei colecții. O definiție intuitivă a lui `foldr` este:

```
foldr op unit [a1, a2, a3, ... , an] == a1 `op` a2 `op` a3 `op` .. `op` an `op` unit
```

Vom exersa folosirea funcției `foldr` scriind câteva funcții, mai întâi folosind recursie, apoi folosind foldr.

Exercițiul 1

(a) Scrieți o funcție recursivă care calculează produsul numerelor dintr-o listă.

```
produsRec :: [Integer] -> Integer
produsRec = undefined
```

(b) Scrieți o funcție echivalentă care folosește `foldr` în locul recursiei.

```
produsFold :: [Integer] -> Integer
produsFold = undefined
```

Exercițiul 2

(a) Scrieți o funcție recursivă care verifică faptul că toate elementele dintr-o listă sunt `True`.

```
andRec :: [Bool] -> Bool
andRec = undefined
```

(b) Scrieți o funcție echivalentă care folosește `foldr` în locul recursiei.

```
andFold :: [Bool] -> Bool
andFold = undefined
```

Exercițiul 3

(a) Scrieți o funcție recursivă care concatenează o listă de liste.

```haskell
concatRec :: [[a]] -> [a]
concatRec = undefined
```

(b) Scrieți o funcție echivalentă care folosește `foldr` în locul recursiei.

```haskell
concatFold :: [[a]] -> [a]
concatFold = undefined
```

Exercițiul 4

(a) Scrieți o funcție care elimină un caracter din șir de caractere.

```haskell
rmChar :: Char -> String -> String
rmChar = undefined
```

(b) Scrieți o funcție recursivă care elimină toate caracterele din al doilea argument care se găsesc în primul argument.

```haskell
rmCharsRec :: String -> String -> String
rmCharsRec = undefined

test_rmchars :: Bool
test_rmchars = rmCharsRec ['a'..'l'] "fotbal" == "ot"
```

(c) Scrieți o funcție echivalentă cu cea de la (b) care folosește `foldr` în locul recursiei.

```haskell
rmCharsFold :: String -> String -> String
rmCharsFold = undefined
```