

Laboratorul 8: Testare folosind QuickCheck

În Haskell avem la dispoziție o librărie care, în anumite situații, generează teste automate. Modificați fișierul `lab8.hs` astfel:

1. Importați modulul `Test.QuickCheck`, i.e. adăugați la începutul fișierului `import Test.QuickCheck`. Dacă acest modul nu este instalat, puteți instala folosind următoarele comenzi direct în terminal (cmd, powershell):

```
cabal update
cabal install QuickCheck
```

Dacă folosiți Windows și `chocolatey`, e posibil să trebuiască să faceți configurare locală în directorul în care veți folosi `QuickCheck`, astfel:

```
c:\Users\...\ex1haskell> cabal install --lib --package-env . QuickCheck
```

Pentru mai multe detalii consultați răspunsul de pe Stack Overflow.

2. Definiți următoarele funcții care calculează dublul, triplul, respectiv, de cinci ori numărul dat ca parametru.

```
double :: Int -> Int
double = undefined
triple :: Int -> Int
triple = undefined
penta :: Int -> Int
penta = undefined
```

3. Observați în fișier următoarea funcție test:

```
test x = (double x + triple x) == (penta x)
```

4. Ce tip are funcția `test`?
5. În interpretor evaluați

```
*Main> quickCheck test
```

și observați rezultatul.

6. Scrieți un alt test care să verifice o proprietate falsă, verificați cu `quickCheck` și observați rezultatul.
7. Scrieți o funcție

```
myLookup :: Int -> [(Int,String)]-> Maybe String
myLookup = undefined
```

care caută un element întreg într-o listă de perechi cheie-valoare și întoarce valoarea găsită folosind un răspuns de tip `Maybe String`.

Scrieți un test

```
testLookup :: Int -> [(Int,String)] -> Bool
testLookup = undefined
```

care verifică faptul că funcția `myLookup` are aceleași rezultate ca funcția `lookup` predefinită.

QuickCheck cu constrângeri

Să încercăm să testăm că `myLookup` este echivalentă cu `lookup` predefinită doar pentru chei pozitive și divizibile cu 5.

```
-- testLookupCond :: Int -> [(Int,String)] -> Property
-- testLookupCond n list = n > 0 && n `div` 5 == 0 ==> testLookup n list
```

Observați faptul că `testLookupCond` are ca rezultat `Property`. Constrângerea din stânga „implicației” `==>` selecționează din valorile de intrare generate doar pe acelea care satisfac condiția dată. Evaluați în interpretor `quickCheck testLookupCond` și observați rezultatul.

8. (a) Scrieți o funcție `myLookup'` cu aceeași semnătură ca `myLookup` care atunci când găsește valoarea, capitalizează prima literă.
- (b) Scrieți un predicat care testează că `myLookup'` este echivalentă cu `lookup` pentru listele care conțin doar valori care încep cu majusculă. Verificați-l folosind `quickCheck`.
9. Se dă următoarea implementare a metodei de sortare quicksort

```
quicksort :: Ord a => [a] -> [a]
quicksort [] = []
quicksort (x:xs) = smalls ++ [x] ++ bigs
    where smalls = quicksort [n | n <- xs, n <= x]
          bigs   = quicksort [n | n <- xs, n > x]
```

Scrieți câte un predicat pentru a verifica următoarele proprietăți pentru liste de întregi:

- a) `quicksort(quicksort(l)) = quicksort(l)`
- b) prin aplicarea sortării pe o listă și pe lista inversată se obțin două liste identice
- c) prin sortarea listei `[1..n]` nu se modifică lista
- d) prin sortare nu se modifică lungimea listei

Să se ruleze cu `quickCheck` predicatele respective.

Sa se implementeze aceleași proprietăți pentru programul urmator (care are o eroare). Sa se ruleze cu quickCheck si sa se observe rezultatele.

```
quicksortBuggy :: Ord a => [a] -> [a]
quicksortBuggy []      = []
quicksortBuggy (x:xs) = smalls ++ [x] ++ bigs
                        where smalls = quicksortBuggy [n | n <- xs, n < x] -- oops
                              bigs  = quicksortBuggy [n | n <- xs, n > x]
```

Testare pentru tipuri de date algebrice

Definiți o instanță a clasei Arbitrary pentru tipul de date ElemIS (instanțe similare în **cursul 7**)

```
data ElemIS = I Int | S String
  deriving (Show,Eq)
```

9. Definiți o funcție myLookUpElem care funcționează similar cu lookUp, doar ca funcționează pentru chei de tip întreg și valori de tip ElemIS.

```
myLookUpElem :: Int -> [(Int,ElemIS)]-> Maybe ElemIS
myLookUpElem = undefined
```

Scrieți un test

```
testLookUpElem :: Int -> [(Int,ElemIS)] -> Bool
testLookUpElem = undefined
```

Și rulați în consolă quickCheck testLookUpElem.