Software Testing Assignment 4

Cindy Berghuizen, Omar Pakker , Chiel Peter, Maria Gouseti

September 27, 2013

1: Book Exercise

Chapter 4

- Prove of Theorem 4.38.2 on page 135. I don't understand the last steps of both parts.

Chapter 5

2: Random data generator

```
genSetMax = 100
genSetMaxEntries = 10

genSet :: IO (Set Int)
genSet =  do
   n  <- getRandomInt genSetMaxEntries
   ns <- genSet' genSetMax n
   return ns

genSet' :: (Eq a, Num a) => Int -> a -> IO (Set Int)
genSet' _ 0 = return (Set [])
genSet' d c = do
         n <- getRandomInt d
         ns <- genSet' d (c-1)
         return (insertSet n ns)
```

Time spent: 10 minutes

Set intersectiom, union, difference

Haskell Program

```
intersectSet :: (Ord a) => Set a -> Set a -> Set a
intersectSet (Set [])     set2  =  (Set [])
intersectSet (Set (x:xs)) set2  | inSet x set2 = insertSet x (intersectSet (Set xs) set2)
                                | otherwise = (intersectSet (Set xs) set2)

-- Pre: no duplicates, it is sorted
-- Post: no duplicates, it is sorted
differenceSet :: (Ord a) => Set a -> Set a -> Set a
differenceSet (Set [])     set2  =  (Set [])
differenceSet (Set (x:xs)) set2  | inSet x set2 = (differenceSet (Set xs) set2)
                                 | otherwise = insertSet x (differenceSet (Set xs) set2)

--Union is already implemented in SetOrd.hs
```

Union was already implemented in SetOrd.hs

Intersection Test

The property of an Intersection set is the following:
Property: A set I is an intersection of sets A and B if the elements of I are in A and B. $A \cap B := \{x : x \in A \wedge x \in B\}$

We test by calling function automatedI. This function generates two random sets and calculates the intersection of those sets. To see if the intersection set indeed is the intersection of A and B we check if it is an subset of A and a subset of B in the function testIntersect. This function returns True is that is the case. The function generateIntersectionTest is used to test multiple cases, this function returns True if all testcases return True.

```
-- A set I is an intersection of A and B if I is an subset of A and B
testIntersect :: (Ord a) => Set a -> Set a -> Set a -> Bool
testIntersect a b i = subSet i a && subSet i b

automatedI :: IO Bool
automatedI = do
    a <- genSet
    b <- genSet
    return $ testIntersect a b (intersectSet a b)

automatedI' :: Int -> IO [Bool]
automatedI' 0 = return []
automatedI' c = do
     d <- automatedI
     ds <- automatedI'(c-1)
     return (d:ds)

generateIntersectionTest :: Int -> IO String
generateIntersectionTest c = do
    ps <- automatedI' c
    return ("All Checks Valid: " ++ (show (all (\x -> x) ps)))
```

Difference Test

The property from a Difference set is the following:
An set D is the difference of A and B if its element are in A and it has no elements in common with B. Also written as: $A \setminus B := \{x : x \in A \wedge x \notin B\}$

The testing is done by inserting two random sets in the function automatedD and calculating the difference of those two sets. Next, we will check that the differenceset is indeed a subset of set A and that the difference set has no elements in common with set B. If this is the case the differenceSet is correct and the function automatedD will return True.
For testing multiple sets we can make use of the function generateDifferenceTest. This will result in True if all the testcases return true in atomatedD.

```
-- Property: An set D is the difference of A and B if it is an subset
--  of A and has no elements in common with B
testDifference :: (Ord a) => Set a -> Set a -> Set a -> Bool
testDifference a b d = subSet d a && noElement d b

noElement :: (Ord a) => Set a -> Set a -> Bool
noElement (Set[]) _ = True
noElement (Set(x:xs)) set | inSet x set = False
                          | otherwise = noElement (Set xs) set

automatedD :: IO Bool
automatedD = do
    a <- genSet
    b <- genSet
    return $ testDifference a b (differenceSet a b)

automatedD' :: Int -> IO [Bool]
automatedD' 0 = return []
automatedD' c = do
```

```
        d <- automatedD
        ds <- automatedD'(c-1)
        return (d:ds)

generateDifferenceTest :: Int -> IO String
generateDifferenceTest c = do
    ps <- automatedD' c
    return ("All Checks Valid: " ++ (show (all (\x -> x) ps)))
```

Union Test

The property to test of a union set is the following:
Every element in either of the sets should be an element of the union set. $A \cup B := \{x : x \in A \vee x \in B\}$

To test this we use the function isElementOf. isElementOf gets a set A, a set B and the presumable union set of A and B. It then tests if every element in the unionset is an element in set A or in set B. If this is the case the function will return True. To check multiple cases testUnion can be used. This returns the list of all the results of the test cases.

```
--PROPERTY : Every element in either of the sets should be an element of the union
testUnion :: Int -> IO [Bool]
testUnion 0 = return []
testUnion a = do        n <- testUnion1
                        ns <- testUnion (a-1)
                        return(n : ns)

testUnion1 :: IO Bool
testUnion1 = do         n <- randomIntSet
                        m <- randomIntSet
                        return(isElementOf n m (unionSet n m))

isElementOf :: (Ord a) => Set a -> Set a -> Set a -> Bool
isElementOf (Set a) (Set b) (Set c) = all (\x -> elem x c) (a++b)
```

```
Lab4> generateDifferenceTest 10
All Checks Valid: True"
Lab4> generateIntersectionTest 10
All Checks Valid: True"
Lab4> generateDifferenceTest 10
All Checks Valid: True"
Lab4> testUnion 10
True,True,True,True,True,True,True,True,True,True]
Lab4>
```

Time spent: 75 minutes

Transitive Closure

```
trClos :: (Ord a) => Rel a -> Rel a
trClos x = trClos2 x []

-- If the closure n+1 is a subset of n than all closures are found
-- because no new elements were found
trClos2 :: (Ord a) => Rel a -> Rel a -> Rel a
trClos2  = fix (\ f x y ->
           if subSet (list2set x) (list2set y) then (nub x)
           else f ((x @@ x)++x) x)
```

Time spent: 1.5 hours

Testing Closure

For testing the closure we use the following property:
If the relations (x,y) and (y,z) exist in a transitiveclosure, then the relation (x,z) should exist in the transitive closure set. This can also be written as:
$\{(x,y),(y,z) \in R^+ \rightarrow (x,z) \in R^+\}$

The function testTrClos gets two relations for the input. These inputs should be the same. The function testTrClos checks if the ??????

```
testTrClos :: (Ord a) => Rel a -> Rel a -> Bool
testTrClos [] _ = True
testTrClos (x:xs) z = (all (\y -> elem y z) ([x] @@ z)) && (testTrClos xs z)

--RANDOM TESTING
randomTestsTrClos :: Int -> IO [Bool]
randomTestsTrClos 0 = return []
randomTestsTrClos a = do        n <- randomTestTrClos
                                m <- randomTestsTrClos (a-1)
                                return (n:m)

randomTestTrClos :: IO Bool
randomTestTrClos = do   n <- randomRelation maxSize
                        return( testTrClos (trClos n) (trClos n))

randomRelation :: (Eq a, Num a) => a -> IO [(Int,Int)]
randomRelation 0 = return []
randomRelation a = do   n <- getRandomInt range
                        m <- getRandomInt range
                        l <- randomRelation (a-1)
                        return((n,m):l)
```

```
Lab4> randomTestsTrClos 10
True,True,True,True,True,True,True,True,True,True]
Lab4>
```

Time spent: 2 hours