Software Testing Assignment 4

Cindy Berghuizen, Omar Pakker , Chiel Peter, Maria Gouseti

September 24, 2013

1: Book Exercise

-

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
```

Intersection Test

```
-- 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

```
-- 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

```
--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)
```

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

```
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)
```

Time spent: 2 hours