Lab Session Software Testing 2014, Week 4 With each deliverable, indicate the time spent.

1. If the course topic of this week was difficult for you, you should spend some time to read up on relations in Chapter 5 of "The Haskell Road".

   (Deliverable: List with specific points that cause difficulty, with a precise indication of what you do not understand, and why.)

2. Teach yourself `Hspec` and `QuickCheck` using the tutorials and links given in the course slides for Week 4.

   (Deliverable: honest indication of the time spent.)

3. Implement a random data generator for the datatype `Set Int`, where `Set` is as defined in `http://homepages.cwi.nl/~jve/rcrh/SetOrd.hs`. First do this from scratch, next use QuickCheck to random test this type.

   (Deliverables: Haskell program, indication of time spent.)

4. Implement operations for set intersection, set union and set difference, for the datatype `Set` defined in `http://homepages.cwi.nl/~jve/rcrh/SetOrd.hs`. Next, use automated random testing to check that your implementation is correct. First use your own random generator, next use QuickCheck.

   (Deliverables: Haskell program, test code, short test report, indication of time spent.)

5. Suppose we implement binary relations as list of pairs, Haskell type `[(a,a)]`.

   Assume the following definitions:

```
type Rel a = [(a,a)]

infixr 5 @@

(@@) :: Eq a => Rel a -> Rel a -> Rel a
r @@ s =
  nub [ (x,z) | (x,y) <- r, (w,z) <- s, y == w ]
```

Use this to implement a function

```
trClos :: Ord a => Rel a -> Rel a
```

that gives the transitive closure of a relation, where the relation is represented as a list of pairs.

E.g., `trClos [(1,2),(2,3),(3,4)]` should give

`[(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)]`.

(Deliverable: Haskell program, indication of time spent.)

6. Give a reasonable Hspec specification for the function you defined in the previous exercise.

   (Deliverable: Haskell program, indication of time spent.)

7. Test the function `trClos` from the previous exercises. Devise your own test method for this. Try to use random test generation. Define reasonable properties to test. Can you use QuickCheck? How?

   (Deliverables: test code, short test report, indication of time spent.)

8. Bonus question. Consider the following function:

```
module Lab4 where

fp :: Eq a => (a -> a) -> a -> a
fp f = \ x -> if x == f x then x else fp f (f x)
```

Your task is to explain the following behaviour:

```
*Lab4> fp (\ x -> ((x + 2/x)/2)) 2
1.414213562373095
*Lab4> 1.414213562373095^2
1.9999999999999996
*Lab4> fp (\ x -> ((x + 3/x)/2)) 3
1.7320508075688772
```

```
*Lab4> 1.7320508075688772^2
2.9999999999999996
*Lab4> fp (\ x -> ((x + 4/x)/2)) 4
2.0
*Lab4> fp (\ x -> ((x + 5/x)/2)) 5
2.23606797749979
*Lab4> 2.23606797749979^2
5.000000000000001
```

Deliverable: explanation of why `fp (\ x -> ((x + a/x)/2)) a` computes the square root of a.