Lab 3 report

Group FT_I

Dominik Safaric, Nian Liu, Guy Rombaut, Bas Meesters

1. Since we are with four people, naturally, each of us finds different things difficult. Below is a list of the things we found individually.
    a. The thing we all find difficult the most is the proving of certain properties. Even the examples don't really make sense to us. We certainly would like to practice this more in the workshops.
    b. Also, what can be quite difficult is to use all the different properties of sets easily. They can be a bit abstract and there are a lot of them so it takes some time to get to know them well.
    c. Equivalence classes are not clear at all. We don't understand their purpose, what to do with it and how to do it.
    d. We don't know what the exact difference is between a Relation R and R^2 and how you can get R^2 from R. Relation composition is also not understood completely.
2. Time spent on Hspec and QuickCheck: 1 hour for Hspec and 30 minutes on QuickCheck.
3. Time spent on test generator: Took about 1 hour and half to write a data generator and an automatable test generator for sets. Time spent on functions & testable properties: 30 minutes. Since the test generators were already finished the testing was done in minutes.

4. Union was already there in SetOrd.hs. We could use that function and change it a bit to create the intersection and the difference of two sets. Testable properties were not that difficult to find. For union, all the elements of both lists should be in the union set. No other elements should be in the union set (functions tpUnion and tpUnion2). For intersection, only when an element form set 1 is also in set 2 it should be in intersection set. This goes the same for set 2. . No other elements should be in the intersection set (functions tpIntersection and tpIntersection2). For difference only if an element is in set 1 and not in set 2 it should be in the difference set (function tpDifference).

    We used our own random sets generator to test the testable properties which gave the correct results (after trial and error though, difference was implemented wrong in first instance). The Quick check could use the same properties. For QuickCheck to work, we had to make the type Set an instance of Arbitrary. This took quite some time. After Vadim pointed out we should use other instances (for example lists) of Arbitrary it was a bit easier. Still did take something around an hour or so.

    Since we tried the exercises separately we also included alternatives to union, intersection and difference. They are included below the first versions.

5. Time spent: 45 minutes. The first part, where we figured out that we could use the @@ at the relation itself, was figured out pretty fast. Took a while to notice we should do it recursively because new transitive relations could be found while iterating.
6. Time spend: 30 minutes.

7. Time spend: 45 minutes. Mostly because we had to figure out how to use arbitrary again for the type Rel a. The tests are quite similar to the specification given in 6. We needed functions to create relations from lists and vice versa so random relations could be created more easily. The testable properties we created are: the trClos of a relation already containing all elements in transitive closure should be the same. The empty relation should return the empty relation. The relation with all unique elements should after trClos still be the same.
8. The function fp expects a function and what happens when fp is called with any function and a number is that the function is applied with this number and if the result is the same as applying the same function again then it is terminated. If not, the function is called again with the result of the previous one. So this is an iterative process that repeats until f x == x.

   Now to the function (\x -> (x + a/x)/2). What happens here is that x is an overestimate of the square and a/x is an underestimate. Which in first instance isn't of course since only for numbers under 4 this is the case. Anyway, the average of those two is (x + a /x)/2. When x is an underestimated the new value will be higher than the previous one, since (x + a/x)/2 will be higher than the square root. When x is an overestimate, the new value will be lower than the previous one, since (x + a/x)/2 will be higher than the square root. This way, slowly it will start going to a fixed point, the square root of x (or a number very, very close to it).