Lab 4 - Week 6. Invariant and Relations

Aim

Apply the theory of Invariants and relations to practical problems using Haskell.

Prerequisites

- Read or reread Chapters 4 and 5 of *The Haskell Road*.
- Make a list of questions on specific points that cause difficulty in understanding.

Submission Guidelines

- Submit one Haskell file per exercise (only Haskell files allowed).
- Name each file as `ExerciseX.hs` for exercises and `EulerX.hs` for Euler problems.
- Ensure that the respective module of each file has the same naming format (`module ExerciseX where`).
- Follow the exercise naming conventions closely. Some exercises go through automated testing, so it is important not to change the indicated declarations.
- Do not include any personally identifiable information in the submissions.
- If using additional dependencies, indicate so in a comment at the top of the file.
- Indicate the time spent on each exercise like so `Time Spent: X min`. Make sure it is **in minutes.**
- Codegrade: For each assignment, you need to create a new group with all teammates. Please note that once you submit, you cannot change the team structure, so be cautious.

Imports

```
import Data.List
import System.Random
```

```
import Test.QuickCheck
```

Exercise 1

Implement a random data generator for the datatype `Set Int`, where `Set` is as defined in SetOrd.hs. First do this from scratch, next give a version that uses *QuickCheck* to random test this datatype.

Deliverables: two random test generators, indication of time spent.

Exercise 2

Implement operations for set intersection, set union and set difference, for the datatype `Set` defined in `SetOrd.hs`. Next, use automated testing to check that your implementation is correct. First use your own generator, next use QuickCheck.
Use the following declarations:

```
setIntersection :: Ord a => Set a -> Set a -> Set a setUnion :: Ord a => Set a -> Set
a -> Set a setDifference :: Ord a => Set a -> Set a -> Set a
```

Deliverables: implementations, test properties, short test report, indication of time spent.

Exercise 3

Suppose we implement binary relations as list of pairs, Haskell type `[(a,a)]`. Assume the following definition:

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

Use the following declaration:

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

to define a function that gives the symmetric closure of a relation, where the relation is represented as an ordered list of pairs. E.g., `symClos [(1,2),(2,3),(3,4)]` should give `[(1,2),`

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

Deliverables: Haskell program, indication of time spent.

Exercise 4

A relation R is serial on a domain A if for any x $\in$ A there is an y $\in$ A such that xRy. Suppose relations are represented as lists of pairs:

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

1. Write a function for checking whether a relation is serial:

```
isSerial :: Eq a => [a] -> Rel a > Bool
```

1. Test your implementation with two QuickCheck properties.
2. Consider the relation R = {(x, y) | x = y(mod n)}, where (mod n) is the modulo function in modular arithmetic and n > 0. Discuss whether (and when) R is serial. How can you test whether R is serial? How can you prove that R is serial?

Deliverables: Haskell program, QuickCheck properties, short test report (including the proof), indication of time spent.

Exercise 5

Use the datatype for relations from the previous exercise, plus

```
infixr 5 @@

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

to define a function:

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

that gives the transitive closure of a relation, represented as an ordered 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)]`.

Deliverables: Haskell program, indication of time spent.

Exercise 6

Test the functions `symClos` and `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.

Exercise 7

Is there a difference between the symmetric closure of the transitive closure of a relation R and the transitive closure of the symmetric closure of R ?

Hint: If your answer is that these are the same, you should give an argument, if you think these are different you should give an example that illustrates the difference.

Deliverables: Haskell file with the answer in comment form, indication of time spent

---

Bonus 1

In the lecture notes, `Statement` is in class `Show`, but the `show` function for it is a bit clumsy. Write your own `show` function for imperative programs. Next, write a `read` function, and use `show` and `read` to state some abstract test properties for how these functions should behave. Next, use `QuickCheck` to test your implementations.

Deliverables: implementation, `QuickCheck` properties, test report, indication of time spent.

Bonus 2

If this was all easy for you, you might wish to throw in a solution to a difficult problem from Project Euler.
