Lab 3 report

Group FT_I

Dominik Safaric, Nian Liu, Guy Rombaut, Bas Meesters

1. Time spent: At least 3 hours of trying.

   The biggest problem here was to get Sudoku be accepted by property so not only one random example is tested but multiple. This unfortunately did not work so we had to settle with using just one random example. We also believe that it is not possible, or in any case quite hard, to use QuickCheck on Sudoku. Sudoku is a function type and since functions cannot be an instance of Eq, they cannot be compared. This makes it hard to make it an instance of Arbitrary and therefore we did not succeed in this.

2. Time spent: 30 minutes

   It was quite easy actually to test if the solutions were really minimal. We just had to implement the two post conditions given in the assignment as invariants. That is, we needed to check if the solution only has one possible solution and if any of the other positions was erased, if it then got multiple solutions. Which it should, otherwise it was not minimal. Together these form the invariant of testing a minimal solution. We also created a formal specification in Hspec for this.

3. Time spent: 1.5 hours / 2 hours

   For this assignment we wrote a function which deletes one random block. And a function which uses that to delete n random blocks. It was possible to delete three or four random blocks while still having a unique solution. Five random blocks did not terminate in the time we had it compute, meaning it could not find a Sudoku in which five random blocks were deleted while still having a unique solution. We also submitted another solution of deleting three blocks. This is because we did it separately and it seemed a shame to delete it. You don't have to explicitly look at this.

4. Time spent: 45 minutes / 1 hour (with addition 2 hours)

   This assignment was not really hard, but we had to be careful to be precise. The new constraints which are added is that the new sub grids also had only unique values. This is formalized and implemented in freeInSubGrid2 in Smodified.hs.

   All the other functions we had to build were basically the same as they were before only we had to use the extra constraints in all the solving functions. Like consistent, prune, extendNode, etc. All can be found in Smodified.hs.

   Edit: We wanted to make the Sudoku function more general so it was easier to make a variation of the Sudoku without having to duplicate a lot of code each time. Unluckily this as a bit harder and more error prone than we hoped and we could not totally finish it in time. We did however manage to make it a bit more abstract so functions could be reused.

5. Time spent: 5 / 10 minutes

   This was very easy after 4 was succeeded. Almost all the functions could stay the same only changing that they used the functions made in assignment 4. Minimalizing, extending node, check unique solution was all the same. The only difference was pruning, taking into account the new constraints, and since that function as already made in assignment 4 we were done.

6. According to the paper associated with the assignment the best way to create difficulty in Sudokus is by having the user need to use more complex techniques. The way this can be checked is by calculating the amount of computations needed to arrive at the result. Having more computations means having more logical steps needed. Of course, this only applies when computations are efficient and work like humans. To implement this was a bit out of the scope of this assignment so we tried simpler ways to create difference in difficulty in Sudoku's.

   We made a simple and hard Sudoku generator which both have 30 given values and both have unique solutions. The easy one makes sure every sub grid has at least one value and makes sure that every number is at least three times given in the problem. Though, we don't have mathematical prove or correct test functions we assume - and part of the internet as well- that less logical steps are needed to complete these Sudokus. The hard ones delete two blocks first and thereafter make sure that two random values are represented maximal twice each in the problem. This will probably force the user to use more complex techniques to solve the Sudoku. Though not a very strong test case, we made count functions which counts the amount of constraints on both hard and easy Sudokus. The results gave that hard Sudokus have on average about 20 constraints more. This means that in the start situation the hard Sudokus have on 59 free positions there are on average 20 more possibilities to enter in these positions. More possibilities means that there is more "freedom", which probably means that more logical steps are needed to solve the puzzle.

Bonus: Time spent: A lot, few hours for sure.

Actually the assignment itself was not really hard at all. We could duplicate the normal Sudoku code again and change it a bit to handle the new constraints. Only we found out we duplicated a lot of code, so we wanted to make the normal Sudoku functions more general in which we partly succeeded. In the SS.hs file the normal functions are there, but also the more general ones which take constraints and possible ways to check them into account. This way we can easily create new Sudoku's with additional constraints with lesser code than before. Only because of a lack of time we partly managed to do this. We didn't get to the random functions and the normal ones could also probably be more efficient. Anyway the code for the bonus is in CrossSudoku.hs