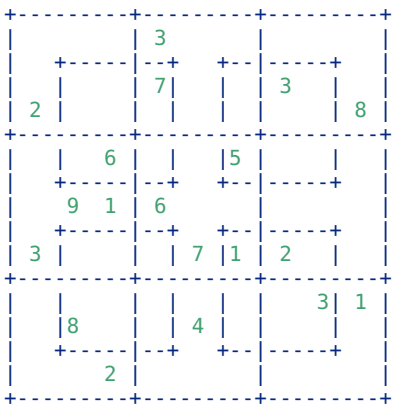
Lab 5 Software Specification and Testing

```
> module Lab5 where
>
> import Data.List
> import System.Random
> import Lecture5
```

Exercise 1



The goal of this exercise is to extend the Sudoku program described in the lecture of this week with functions that can also handle Sudokus of a special kind: the Sudokus that appear in the Dutch evening newspaper NRC-Handelsblad each week (designed by Peter Ritmeester, from Oct 8, 2005 onward). These NRC Sudokus are special in that they have to satisfy a few extra constraints: in addition to the usual Sudoku constraints, each of the 3 × 3 subgrids with left-top corner (2,2), (2,6), (6,2), and (6,6) should also yield a surjective function. The above figure gives an example (this is the NRC sudoku that appeared Saturday Nov 26, 2005).

Your task is to formalize this extra constraint, and to use your formalization in a program that can solve this Sudoku. See also the webpage of Andries Brouwer.

Deliverables: modified Sudoku solver, solution to the above puzzle, indication of time spent.

Exercise 2

When the Sudoku code was presented to an audience of Master of Logic students, a proposal emerged to refactor the code to make the formulation of constraints more uniform. The following definitions were proposed:

```
> type Position = (Row,Column)
> type Constrnt = [[Position]]
```

The regular constraints for Sudoku can now be stated as:

```
> rowConstrnt = [[(r,c)| c <- values ] | r <- values ]
> columnConstrnt = [[(r,c)| r <- values ] | c <- values ]
> blockConstrnt = [[(r,c)| r <- b1, c <- b2 ] | b1 <- blocks, b2 <- blocks ]
```

The generation of the values that are still possible at a given position now takes the following shape:

```
> freeAtPos' :: Sudoku -> Position -> Constrnt -> [Value]
> freeAtPos' s (r,c) xs = let
>    ys = filter (elem (r,c)) xs
>  in
>    foldl1 intersect (map ((values \\) . map s) ys)
```

Refactor the code along the lines of this proposal, and next compare the two versions for extendability and efficiency. Which of the two versions is easier to modify for NRC sudokus, and why? Which of the two versions is more efficient? Devise your own testing method for this, and write a short test report.

Deliverables: Refactored code, test report, indication of time spent.

Exercise 3

A Sudoku problem P is *minimal* if it admits a unique solution, and every problem P' you can get from P by erasing one of the hints admits more than one solution. How can you test whether the problems generated by the code given in the lecture notes are minimal?

Deliverables: testing code, test report, indication of time spent.

Exercise 4

Write a program that generates Sudoku problems with three empty blocks. Is it also possible to generate Sudoku problems with four empty blocks? Five? How can you check this?

Deliverables: generator, short report on findings, indication of time spent.

Exercise 5

Extend the code of the lectures to create a program that generates NRC Sudoku problems, that is, Sudoku problems satisfying the extra constraint explained in the NRC exercise above.

Deliverables: NRC Sudoku generator, indication of time spent.

Exercise 6 (Bonus)

Can you find a way of classifying the difficulty of a Sudoku problem? Can you modify the Sudoku problem generator so that it can generate problems that are minimal, but easy to solve by hand? Problems that are minimal but hard to solve by hand? How can you test whether the problems your program generates satisfy these properties? Consult (Pelánek 2014).

Exercise 7 (Bonus)

Minimal problems for NRC Sudokus need fewer hints than standard Sudoku problems. Investigate the difference. What is the average number of hints in a minimal standard Sudoku problem? What is the average number of hints in a minimal NRC Sudoku problem?

Submission deadline is Sunday evening, October 8th, at 6 pm.

Pelánek, Radek. 2014. “Difficulty Rating of Sudoku Puzzles: An Overview and Evaluation.” arXiv:1403.7373.