METROPOLITAN STATE UNIVERSITY OF DENVER
Department of Mathematical and Computer Sciences

CS 3810 - Principles of Programming Languages (Fall 2019)

Programming Assignment 02

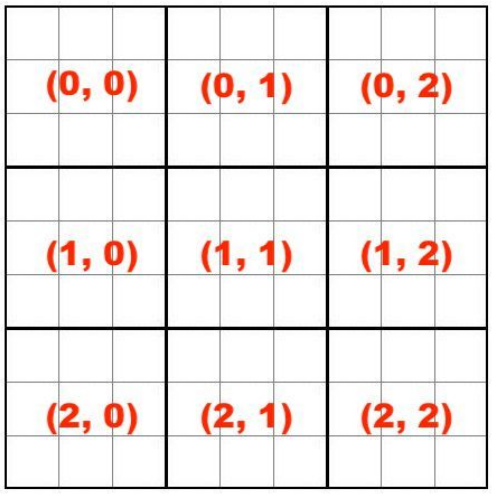
Deadline: November 3rd 11:59pm

Introduction

In this activity you are asked to write an implementation for a Sudoku solver using the Haskell programming language. Wikipedia has an article that explains the rules of the game if you never played Sudoku. The solution for this problem is going to be built in incremental steps to illustrate function programming capabilities. The picture below shows a typical Sudoku board and its boxes (you should use the suggested numbering).

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A Sudoku Board (source: Wikipedia)



Sudoku Boxes (w/ identifications)

Get the code template (`sudoku.hs`) from here.

Types

Type synonyms in Haskell as we discussed in class allow assigning names to pre-existent types. In this assignment you are required to use the following types (note that types are capitalized):

```
type Sequence = [Int]
type Board    = [Sequence]
```

A `Sequence` is just a list of integers. You can think of a `Sequence` as a row, column, or box. In Sudoku, a valid `Sequence` has `9` digits. We will use digit `0` in a `Sequence` to denote an

empty cell. Also, Sudoku requires that no `Sequence` should have repeated digits (with the exception of digit `0`).

A `Board` is a list of `Sequence`. A typical Sudoku board is a list of 9 `Sequence` values, each representing a row and with size 9. The Sudoku board of the example used in the Introduction section of this document could be modeled in Haskell in the following way:

```
board = [  [ 5, 3, 0, 0, 7, 0, 0, 0, 0],
           [ 6, 0, 0, 1, 9, 5, 0, 0, 0],
           [ 0, 9, 8, 0, 0, 0, 0, 6, 0],
           [ 8, 0, 0, 0, 6, 0, 0, 0, 3],
           [ 4, 0, 0, 8, 0, 3, 0, 0, 1],
           [ 7, 0, 0, 0, 2, 0, 0, 0, 6],
           [ 0, 6, 0, 0, 0, 0, 2, 8, 0],
           [ 0, 0, 0, 4, 1, 9, 0, 0, 5],
           [ 0, 0, 0, 0, 8, 0, 0, 7, 9] ]
```

Helper Functions

Begin your project studying the code (and examples) of the two helper functions `toInt` and `toIntList` described and implemented in the provided code template.

Getter Functions

Implement the following "getter" functions based on their descriptions and examples written as comments in the provided code template.

- `getBoard :: [Char] -> Board`
- `getNRows :: Board -> Int`
- `getNCols :: Board -> Int`
- `getBox :: Board -> Int -> Int -> Sequence`
- `getEmptySpot :: Board -> (Int, Int)`

Predicate Functions

Implement the following "predicate" functions based on their descriptions and examples written as comments in the provided code template.

- `isGridValid :: Board -> Bool`
- `isSequenceValid :: Sequence -> Bool`

- `areRowsValid :: Board -> Bool`
- `areColsValid :: Board -> Bool`
- `areBoxesValid :: Board -> Bool`
- `isValid :: Board -> Bool`
- `isCompleted :: Board -> Bool`
- `isSolved :: Board -> Bool`

Setter Functions

Implement the following "setter" functions based on their descriptions and examples written as comments in the provided code template.

- `setRowAt :: Sequence -> Int -> Int -> Sequence`
- `setBoardAt :: Board -> Int -> Int -> Int -> Board`
- `buildChoices :: Board -> Int -> Int -> [Board]`

Main Function

Below are the three to do's left for you to implement in `main`:

- validate the command-line and get the file name containing the board;
- read the contents of the board file into a string;
- create a board from the contents string board;
- use solve to find the solutions, disconsidering the ones that are [[]]; and
- print the solutions found.

Testing

Five sudoku boards are given for you to test here:

- "sudoku0.txt" is the configuration shown in the Introduction section of this document and it has only one solution
- "sudoku1.txt", "sudoku2.txt", and "sudoku3.txt" also have only one solution;
- "sudoku4.txt" has two solutions
- "sudoku5.txt" is the hardest one and it has at least one solution;
- "sudoku6.xt" is invalid.

Submission

For this assignment you just need to submit the `sudoku.hs` file through Blackboard.

Rubric

- `getBoard :: [Char] -> Board`
- `getNRows :: Board -> Int`
- `getNCols :: Board -> Int`
- `getBox :: Board -> Int -> Int -> Sequence`
- `getEmptySpot :: Board -> (Int, Int)`
- `isGridValid :: Board -> Bool`
- `isSequenceValid :: Sequence -> Bool`
- `areRowsValid :: Board -> Bool`
- `areColsValid :: Board -> Bool`
- `areBoxesValid :: Board -> Bool`
- `isValid :: Board -> Bool`
- `isCompleted :: Board -> Bool`
- `isSolved :: Board -> Bool`
- `setRowAt :: Sequence -> Int -> Int -> Sequence`
- `setBoardAt :: Board -> Int -> Int -> Int -> Board`
- `buildChoices :: Board -> Int -> Int -> [Board]`
- `main` 5 to do's

+5 points for each function (max +80)
+20 points for main

Extra Credit

If you are done with the implementation, I am willing to give you UP TO 10 extra points on this assignment if you also submit a second implementation OF YOUR OWN (not just grabbed and adapted with small modifications from the Internet) that is more *haskellian* than the suggested implementation. By *haskellian* I mean an implementation that uses specific features found in Haskell such as function composition, laziness of evaluation, immutability of variables and purity of functions. The second source code that you will have to submit in order to get the extra points must be named `my_sudoku.hs` and it should have a header comment explaining your approach to the problem and why do you think your solution is more *haskellian* than the one provided. The instructor will use his own discretion to decide how much extra points your solution deserves, based on the quality of the code, approach and reasoning.