EDAF40: Lab 3
Building your own Sudoku puzzle, ver. 1.0

Adrian Roth and Jacek Malec

April 15, 2019

The goal of this lab is to continue on the Sudoku board (see labs 1 and 2) and, in addition, work with I/O so that you can automatize some functionality, in particular related to testing. Remember that all functions written in this lab form one possible way of solving Assignment 1, but there are definitely many other approaches. If you find an approach which you think is more logical and easier to understand than our version, we would be happy to hear about it!

Communicating with the World Outside

In lab 2 you have implemented

```
verifySudoku :: Board -> Bool
```

that would check whether a given Sudoku board is valid, either just checking for conflicts or, in the second step, also for blocking situations. (Can you tell what type is `Board` actually?

Today we shall let your program read Sudoku board(s) from some input stream (usually a file) and inform the user which boards were OK and which were not.

Consider the following piece of code (should be available from the lab web page as `Main.hs`):

```
module Main where

import Sudoku

bool :: a -> a -> Bool -> a
bool x _ False = x
bool _ y True = y

boolSum :: [Bool] -> Int
boolSum = sum . map (bool 0 1)

prep, prep' :: String -> [String]
```

```
prep' [] = []
prep' s = take 81 s : prep' (drop 81 s)

prep = prep' . filter (not . flip elem "\n=")

files :: [String]
files = ["easy50.txt", "inconsistent20.txt"]

main :: IO ()
main =
  mapM_ (\s -> do
    sGrids <- readFile s
    let grids = prep sGrids
    let verified = map verifySudoku grids
    --print verified
    let n = boolSum verified
    putStrLn $
      s ++": "++ show n ++"/"++ show (length grids) ++" consistent"
    ) files
```

(BTW, you have probably already seen it.)

Questions:

1. How should the file with your `verifySudoku` be named, alternatively, what do you need to modify in the above code?
2. What does function `boolSum` do?
3. What does function `prep` do?
4. What does function `mapM_` do (look it up in the documentation)?

Task 1. Make sure that your Sudoku verifier works with the input files provided. Use the above `Main.hs` or your own one.

Task 2. Inform the user about the verification success, or the kind of conflict detected, if any, for each analyzed board.

Task 3. Visualize on the screen the place of detected conflict in case when verification was negative: mark the dubious cell somehow and print out the board in a legible way.

`Task 4.` (Optional) Prepare for an interactive determination of the Sudoku size, so that your program can verify Sudoku of the sizes 4x9, 9x9 (and maybe 16x16: how would you code the subsequent digits?) where size is given as a parameter.