Universiteit van Amsterdam

Assignment 2

Erik Kooistra, Ana Oprescu and Damian Frölich

Translated by: Daan Kruis

Sudoku

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

Deadline: Individual: Wednesday 12 February 2020, 16:00
Team: Sunday 16 February 2020, 18:00

1 Introduction and Puzzles (Individual)

Before you begin with the bigger Haskell-program you will have to write this week, there are a few smaller puzzles that you will have to solve. This is not meant to annoy you or to test how well you can find code snippets on the internet, but because it will help you develop some finesse and understanding about Haksell. As mentioned in the lecture there are a few standard functions in Haskell that make the language stronger and you will now actually use them. Below are eleven functions that can be defined using these standard functions. Do this and use the gained knowledge for the rest of the assignment.

1. Define `length` in terms of `foldr` and call it `length'`.
2. Find out what `or` does. Define your own version in terms of `foldr` and call it `or'`.
3. Define `elem x` in terms of `foldr`[1] and call this `elem'`.
4. Define `map f` in terms of `foldr` and call it `map'`.
5. Define `(++)` in terms of `foldr` and call it `plusplus`.
6. Define `reverse` in terms of `foldr` and call it `reverseR`.
7. Define `reverse` in terms of `foldl` and call it `reverseL`.
8. (expert) Define `(!!)` in terms of `foldl`.
9. Create a function `isPalindrome` that determines if a string (or list) is a palindrome.
10. Create a function, called `fibonacci`, that returns and infinite list with the Fibonacci sequence in terms of `scanl`.
11. (expert)[2] Given the following type definition `type List = (Int) -> Int`, write a function that enables you to add elements to this type.

Hand these functions in as the module `Puzzles`, by creating a file called "`Puzzles.hs`" with at the top:

```
module Puzzles

where
```

followed by your answers, **with the right names!** Otherwise the tests will fail and you will receive an insufficient grade. On canvas, under modules, there is a `tar` file, called: `HaskellAssignment.tar.gz`, which contains the tests to use locally. Don't forget to read Section 3 and Section 5.

[1] To get this to work you have to tell Haskell that the parameter is part of the EQ-typeclass, for example: `elem' :: Eq a => a -> [a] -> Bool`

[2] Very useful exercise for the team assignment.

2 Sudoku Team

The Sudoku is a well-known puzzle; background information and rules can easily be found online. Your goal is to implement a general Sudoku solver in the file `SudokuSolver.hs`. To do this you'll need the `Sudoku.hs` file, which can be foun in the `tar`file, called: `HaskellAssignment.tar.gz`, found on canvas under modules.
Important: you are not allowed to add anything to or remove anything from the `Sudoku.hs` file, as we will use our own version for testing. Furthermore you should not define the types in `Sudoku.hs` in your own file, as it will create a 'naming collision' and cause the tests to fail. If everything goes well locally, it will go well on Codegrade.

```haskell
module SudokuSolver

where

import Sudoku
import Data.List
import Data.Maybe


positions, values :: [Int]
positions = [1..9]
values    = [1..9]

blocks :: [[Int]]
blocks = [[1..3], [4..6], [7..9]]

showDgt :: Value -> String
showDgt 0 = " "
showDgt d = show d

showSubgridRow:: [Value] -> String
showSubgridRow = unwords . map showDgt

{-- The chunksOf n xs ++ chunksOf n ys == chunksOf n (xs ++ ys)
    property holds.
--}
chunksOf :: Int -> [a] -> [[a]]
chunksOf _ [] = []  -- Models the Data.List.Split implementation
chunksOf n ls = fst splitted : (chunksOf n . snd) splitted
    where splitted = splitAt n ls


showRow :: [Value] -> String
showRow sr =
  "| "
  ++
  intercalate " | " (map showSubgridRow $ chunksOf 3 sr)
  ++
  " |"


showGrid :: Grid -> String
showGrid grid =
  "+-------+-------+-------+\n"
```

```
  ++
  intercalate "\n+-------+-------+-------+\n" rows
  ++
  "\n+-------+-------+-------+"
  where rows = map (intercalate "\n") $ chunksOf 3 $ map showRow grid


sud2grid :: Sudoku -> Grid
sud2grid s =
  [ [ s (r, c) | c <- [1..9] ] | r <- [1..9] ]


grid2sud :: Grid -> Sudoku
grid2sud gr = \(r, c) -> pos gr (r, c)
  where pos :: [[a]] -> (Row, Column) -> a
        pos gr (r, c) = (gr !! (r - 1)) !! (c - 1)


printSudoku :: Sudoku -> IO()
printSudoku = putStrLn . showGrid . sud2grid
```

The defined types can be found in the file: `Sudoku.hs`.

The goal of the solver is to accept a sudoku as a grid and return, as a grid, the solved sudoku. An example of such a grid is as follows.

```
example1 :: Grid
example1 =
    [ [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]
    ]
```

The first step is to determine the possibilities, given a row, column and subgrid. Furthermore we will need a list of all empty positions in the sudoku. To do this you will need the following functions.

1. `extend :: Sudoku -> (Row,Column,Value) -> Sudoku`
2. `freeInRow :: Sudoku -> Row -> [Value]`
3. `freeInColumn :: Sudoku -> Column -> [Value]`
4. `freeInSubgrid :: Sudoku -> (Row,Column) -> [Value]`
5. `freeAtPos :: Sudoku -> (Row,Column) -> [Value]`
6. `openPositions :: Sudoku -> [(Row,Column)]`

Now that we can calculate the possibilities for every empty space in the sudoku we will also have to check if the sudoku is valid for a given solution.

1. `rowValid :: Sudoku -> Row -> Bool`
2. `colValid :: Sudoku -> Column -> Bool`

3. `subgridValid :: Sudoku -> (Row,Column) -> Bool`
4. `consistent :: Sudoku -> Bool`

2.1 Solving the sudoku

With all these different function we now have enough information to start solving a sudoku. We will do this using a search tree. But before we can do this we must define a few types. A Constraint has as a value a x, y coordinate in the sudoku and a list of all options for that position. The needed types have been defined in the `Sudoku.hs` file and look like this:

```
type Constraint = (Row,Column,[Value])
type Node = (Sudoku,[Constraint])
```

The next function can be useful during debugging:
`printNode :: Node -> IO() printNode = printSudoku . fst`

The first step in the solver is generating a list of all possible Constraints, sorted from the least possibilities to the most possibilities. The type signature of the function is as follows
`constraints :: Sudoku -> [Constraint]`

Using this information you can now write the solve function, whose definition is:
`solveSudoku :: Sudoku -> Maybe Sudoku`

The best way to implement this is to start by creating a list of initial constraints and then per possibility look for new possibilities until you can go no further.

It is useful to write a helper that looks like this:
`solveAndShow :: Grid -> IO()`

3 Important

Stick to the naming conventions as described in the assignment, otherwise your tests will fail and you will get an insufficient grade. Furthermore, you cannot use Haskell compiler 'directives' (these are comments written between `{-# #-}`). Lastly it is important that you do not place a `main` function in your `SudokuSolver.hs` or `Puzzles.hs` files. Testing your program can be done using `ghci` or with the provided tests.

4 Tools

The correctness of your work will be graded automatically. Similarly, part of your coding style and idiomaticity will be graded automatically. For style and idiomaticity: 'suggestions' will be graded by the TAs and for all other mistakes points will be deducted. Except for a few cases it is simply necessary to correct the 'suggestions'.

We do this using the following programs:

- Tests
  - tasty
  - tasty-hunit
  - tasty-quickcheck
  - tasty-smallcheck
- Style & idiomaticity
  - hlint

To install `Tasty` and friends you will need `stack`, see: `https://docs.haskellstack.org/en/stable/README/#how-to-install`. If you have installed stack you can install Tasty and friends using the following shell commands:

```
stack install tasty-1.2.3
stack install tasty-hunit-0.10.0.2
stack install tasty-smallcheck-0.8.1
stack install tasty-quickcheck
```

Hlint can be installed via Ubuntu: `sudo apt install hlint`. Not using ubuntu 18.04? You can also install hlint using stack. The version that is used is **2.0.11**.

5 Hand-in

For the individual assignment you will **only** have to hand in the `Puzzles.hs` file. For the team assignment you will **only** have to hand in the `SudokuSolver.hs` file! Once again, stick to the indicated filenames and naming conventions in the assignment!