Sudoku Solver
Informatics 1 – Introduction to Computation
Functional Programming Tutorial 9

Melkonian, Sannella, Vlassi-Pandi, Wadler

Week 10
due 4pm Tuesday 22 November 2022
tutorials on Thursday 24 and Friday 25 November 2022

You will not receive credit for your coursework unless you attend the corresponding tutorial session. Please email kendal.reid@ed.ac.uk if you cannot join your assigned tutorial.

Good Scholarly Practice: Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct.

This also has links to the relevant University pages. Please do not publish solutions to these exercises on the internet or elsewhere, to avoid others copying your solutions.

Sudoku Solver

This tutorial will help you to write a solver for Sudoku puzzles. The exercise is adapted from Chapter 5 of *Thinking Functionally with Haskell* by Richard Bird (Cambridge Univ. Press, 2015).

Sudoku is played on a 9×9 grid, as shown in the diagram. The goal is to fill in each empty cell with a digit between 1 to 9, so that each row, column, and 3×3 box contains the numbers 1 to 9—hence no number should appear twice in any row, column, or box. Usually, Sudoku puzzles are designed to have a unique solution, but we will write a solver (in the optional section) that returns a list of all possible solutions.

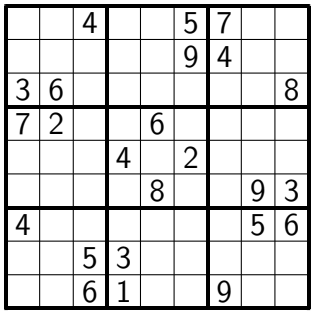

Figure 1: An easy Sudoku puzzle

1 Representing and printing Sudoku puzzles

We will represent a Sudoku puzzle as a matrix, where a matrix is a list of rows and each row is itself a list.

```
type Row a     =  [a]
type Matrix a  =  [Row a]
```

By convention, a row will always have nine elements and a matrix will always have nine rows, so a matrix will always represent a 9×9 grid. (We could use abstract data types to enforce this convention.)

We introduce a type to represent the contents of a cell.

```
type Digit     =  Char
```

By convention, a digit will be one of the characters from `'1'` to `'9'` or a blank. A Sudoku problem has type `Matrix Digit`, with blanks representing cells that haven't yet been filled.

An online Sudoku puzzle generator can be found at sudoku.com. It can generate puzzles at five levels of difficulty: easy, medium, hard, expert, and evil. Here's an easy puzzle:

```
easy :: Matrix Digit
easy =  ["    345  ",
         "  89   3 ",
         "3    2789",
         "2 4  6815",
         "    4    ",
         "8765  4 2",
         "7523    6",
         " 1   79  ",
         "  942    "]
```

A function `put` is provided for printing out puzzles in a more readable format:

```
> put easy
-------------
|   | 34|5  |
|  8|9  | 3 |
|3  |  2|789|
-------------
|2 4|  6|815|
|   | 4 |   |
|876|5  |4 2|
-------------
|752|3  |  6|
| 1 |  7|9  |
|  9|42 |   |
-------------
```

We will represent a partially solved Sudoku puzzle by a matrix of type `Matrix [Digit]` where each cell contains a list of the digits that might appear in that cell.

Exercise 1

Write a function

```
choice :: Digit -> [Digit]
```

that converts a space to a list of all the digits from `'1'` to `'9'`, and otherwise converts a digit to a list containing just that digit.

Use `choice` to write a function

```
choices :: Matrix Digit -> Matrix [Digit]
```

which does that for all of the cells in a Sudoku puzzle. For example,

```
choices easy ==
 [["123456789", "123456789", "123456789",
   "123456789", "3",         "4",
   "5",         "123456789", "123456789"],
  ["123456789", "123456789", "8",
   "9",         "123456789", "123456789",
   "123456789", "3",         "123456789"],
  ["3",         "123456789", "123456789",
   "123456789", "123456789", "2",
   "7",         "8",         "9"],
  ["2",         "123456789", "4",
   "123456789", "123456789", "6",
   "8",         "1",         "5"],
  ["123456789", "123456789", "123456789",
   "123456789", "4",         "123456789",
   "123456789", "123456789", "123456789"],
  ["8",         "7",         "6",
   "5",         "123456789", "123456789",
   "4",         "123456789", "2"],
  ["7",         "5",         "2",
   "3",         "123456789", "123456789",
   "123456789", "123456789", "6"],
  ["123456789", "1",         "123456789",
   "123456789", "123456789", "7",
   "9",         "123456789", "123456789"],
  ["123456789", "123456789", "9",
   "4",         "2",         "123456789",
   "123456789", "123456789", "123456789"]]
```

2 Pruning

Many Sudoku puzzles, including the easy puzzle above, can be solved by the following *pruning* technique.

When a cell in a partially-solved puzzle of type `Matrix [Digit]` contains a singleton list `['`d`']`, we can improve the solution by removing d from the list of possible values in any other cell in the same row, column, or box. Repeatedly applying pruning is sometimes enough to solve a puzzle.

Exercise 2

Write a function

```
pruneRow :: Row [Digit] -> Row [Digit]
```

for pruning a row of a puzzle that leaves any list consisting of a single digit unchanged, but removes from any other list any single digit that appears elsewhere in the row. For example,

```
pruneRow ["139", "269", "17", "1578", "3", "4", "5", "256", "1"]
      == ["9",   "269", "7",  "78",   "3", "4", "5", "26",  "1"]
```

Hint: The function `splits :: [a] -> [(a, [a])]` from p274 of the textbook might come in handy. Its definition is provided.

Code is provided for a function

```
prune :: Matrix [Digit] -> Matrix [Digit]
```

which applies `pruneRow` to perform pruning on each row, column, and box in the matrix.

Exercise 3

Write a function

```
close :: Eq a => (a -> a) -> a -> a
```

that takes a function `g` and value `x` and repeatedly applies `g`, starting with `x`, until the value no longer changes.

Exercise 4

Write a function

```
extract :: Matrix [Digit] -> Matrix Digit
```

that takes a partially-solved Sudoku puzzle where every list contains a single digit into the solved puzzle containing those digits, and is undefined otherwise.

Exercise 5

Using `choices`, `prune`, `close`, and `extract`, write a function

```
solve :: Matrix Digit -> Matrix Digit
```

which solves any Sudoku puzzle that can be solved by repeated pruning. Which of the four given puzzles — `easy`, `medium`, `hard` and `evil` — can be solved in this way?

3 Optional Material: Expanding a single cell

> Following the Common Marking Scheme, a student with good mastery of the material is expected to get 3/4 points. This section is for demonstrating exceptional mastery of the material. It is optional and worth 1/4 points.

Given a partial solution of type `Matrix [Digit]`, there are three possibilities.

1. Every list of digits contains only one element. Then we can use `extract` to get the solution.
2. Some list of digits is empty. Then, there is no possible solution.
3. Some list of digits contains more than one element.

In the third case, consider the shortest such list ds, of length n. (If more than one list is shortest, pick the first one.) Then for each d in ds, we can generate a new partial solution where ds is replaced by $[d]$, and then recursively try to solve each of the n resulting partial solutions.

For example, given the matrix of partial solutions:

```
[["9",     "1578",  "3",     "67",    "167",   "4",     "2",     "15678", "5678"],
 ["4",     "178",   "6",     "5",     "12379", "127",   "138",   "13789", "789"],
 ["57",    "157",   "2",     "8",     "13679", "167",   "135",   "13456", "5679"],
 ["238",   "238",   "9",     "267",   "2678",  "5",     "138",   "13678", "4"],
 ["58",    "6",     "7",     "1",     "4",     "3",     "9",     "2",     "58"],
 ["1",     "23458", "45",    "9",     "2678",  "267",   "358",   "35678", "5678"],
 ["2356",  "12345", "145",   "246",   "1256",  "8",     "7",     "59",    "259"],
 ["2567",  "1257",  "15",    "267",   "12567", "9",     "4",     "58",    "3"],
 ["257",   "24579", "8",     "3",     "257",   "27",    "6",     "59",    "1"]]
```

The earliest, shortest list containing more than one choice is `"67"` (the fourth element of the first row) and expanding it gives two simpler partial solutions.

```
[[["9",     "1578",  "3",     "6",     "167",   "4",     "2",     "15678", "5678"],
  ["4",     "178",   "6",     "5",     "12379", "127",   "138",   "13789", "789"],
  ["57",    "157",   "2",     "8",     "13679", "167",   "135",   "13456", "5679"],
  ["238",   "238",   "9",     "267",   "2678",  "5",     "138",   "13678", "4"],
  ["58",    "6",     "7",     "1",     "4",     "3",     "9",     "2",     "58"],
  ["1",     "23458", "45",    "9",     "2678",  "267",   "358",   "35678", "5678"],
  ["2356",  "12345", "145",   "246",   "1256",  "8",     "7",     "59",    "259"],
  ["2567",  "1257",  "15",    "267",   "12567", "9",     "4",     "58",    "3"],
  ["257",   "24579", "8",     "3",     "257",   "27",    "6",     "59",    "1"]],
 [["9",     "1578",  "3",     "7",     "167",   "4",     "2",     "15678", "5678"],
  ["4",     "178",   "6",     "5",     "12379", "127",   "138",   "13789", "789"],
  ["57",    "157",   "2",     "8",     "13679", "167",   "135",   "13456", "5679"],
  ["238",   "238",   "9",     "267",   "2678",  "5",     "138",   "13678", "4"],
  ["58",    "6",     "7",     "1",     "4",     "3",     "9",     "2",     "58"],
  ["1",     "23458", "45",    "9",     "2678",  "267",   "358",   "35678", "5678"],
  ["2356",  "12345", "145",   "246",   "1256",  "8",     "7",     "59",    "259"],
  ["2567",  "1257",  "15",    "267",   "12567", "9",     "4",     "58",    "3"],
  ["257",   "24579", "8",     "3",     "257",   "27",    "6",     "59",    "1"]]]
```

Exercise 6

Write a function

```
failed :: Matrix [Digit] -> Bool
```

that returns true if any of the lists of choices is empty.

Exercise 7

Write a function

```
solved :: Matrix [Digit] -> Bool
```

that returns true if all of the lists of choices contain exactly one digit.

Exercise 8

Write a function

```
shortest :: Matrix [Digit] -> Int
```

that returns the length of the shortest list of digits that has more than one choice. You may assume that the input is not solved, which means that not all lists of digits are singletons.

The standard Prelude defines the function

```
break :: (a -> Bool) -> [a] -> ([a], [a])
```

that given a predicate and a list containing an element that satisfies the predicate, splits the list into two lists where the first is all the elements up to and not including the element which satisfies the predicate, and the second is the remainder, beginning with the element that satisfies the predicate. For example,

```
break (\ds -> length ds == 2)
      ["9","1578","3","67","167","4","2","15678","5678"]
== (["9","1578","3"],["67","167","4","2","15678","5678"])
```

Exercise 9

Using `shortest` and using `break` twice, write a function

```
expand :: Matrix [Digit] -> [Matrix [Digit]]
```

that performs the operation described at the beginning of this section: find the earliest, shortest list of choices containing more than one choice, and generate a new matrix for each choice in the list.

Hint: We can break the matrix up into pieces using

```
(preMat, row:postMat) =  break (any p) mat
(preRow, ds:postRow)  =  break p row
```

where p is a predicate on lists of choices that returns `True` if its length is equal to a given length. We can then reassemble the array by computing

```
preMat ++ [preRow ++ [[d]] ++ postRow] ++ postMat
```

where d is a chosen digit from ds.

Exercise 10

Using `failed`, `solved`, `extract`, `close`, `prune`, and `expand`, write a search program

```
search :: Matrix Digit -> [Matrix Digit]
```

that executes the `search` procedure described at the beginning of this section. Use it to solve all four sudoku puzzles.