Functional Programming With Lists

Amitabha Sanyal

Department of Computer Science and Engineering
IIT Bombay.
Powai, Mumbai - 400076

`as@cse.iitb.ac.in`

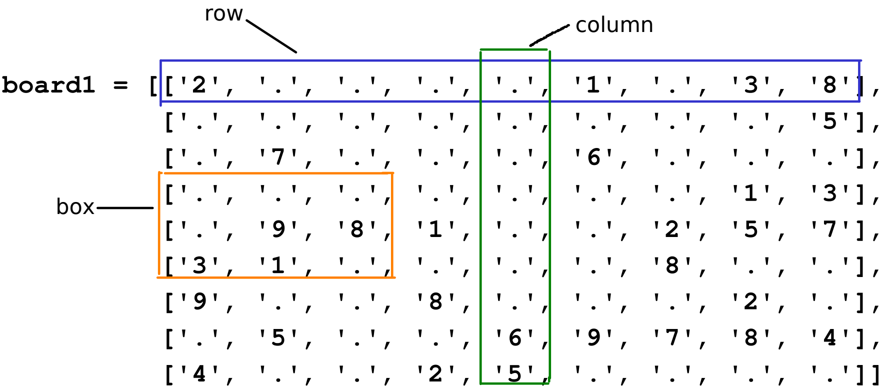
November 2024

A Sudoku solver

As an example of:

1. List processing in Haskell. Use of list comprehensions.
2. Wholemeal programmimg: Transforming lists as a whole. Never look at individual elements.
3. Backtracking in lazy languages.

The Board



```
board1 = [['2', '.', '.', '.', '.', '1', '.', '3', '8'],
          ['.', '.', '.', '.', '.', '.', '.', '.', '5'],
          ['.', '7', '.', '.', '.', '6', '.', '.', '.'],
          ['.', '.', '.', '.', '.', '.', '.', '1', '3'],
          ['.', '9', '8', '1', '.', '.', '2', '5', '7'],
          ['3', '1', '.', '.', '.', '.', '8', '.', '.'],
          ['9', '.', '.', '8', '.', '.', '.', '2', '.'],
          ['.', '5', '.', '.', '6', '9', '7', '8', '4'],
          ['4', '.', '.', '2', '5', '.', '.', '.', '.']]
```

```
type Matrix a = [[a]]
type Board = Matrix Char
```

Characterizing a correct solution

Some constants

```
boxsize = 3:: Int
allvals = "123456789"
blank c = c == '.'
```

A Board is correct, if each row, each column and each box is free of duplicates.

```
correct :: Board -> Bool

correct b = all nodups (rows b) &&
            all nodups (cols b) &&
            all nodups (boxes b)

nodups [] = True
nodups (x:xs) = notElem x xs && nodups xs
```

Characterizing a correct solution

```
rows = id
```

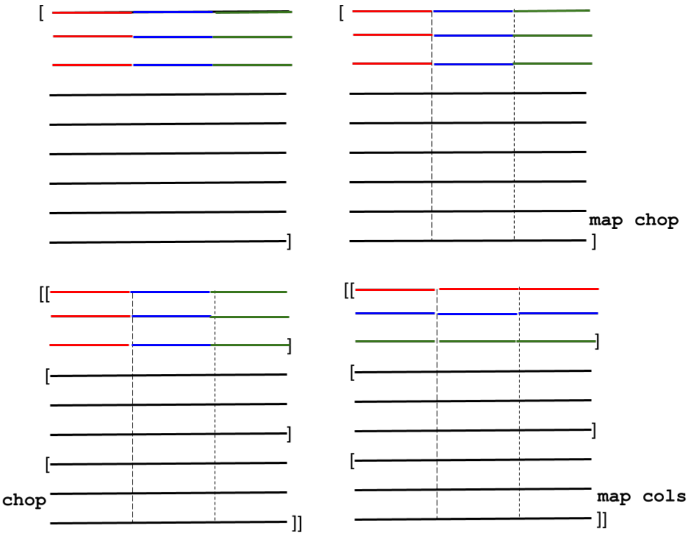
cols makes rows out of columns

```
cols [] = replicate 9 []
cols (x:xs) = zipWith (:) x (cols xs)
```

boxes makes rows out of columns

```
boxes = ?
```

boxes in pictures



Characterizing a correct solution

```
boxes = map unchop . unchop . map cols . chop . map chop

chop = chopBy boxsize
    where chopBy n [] = []
          chopBy n l = take n l : chopBy n (drop n l)

unchop = concat
```

Notice that `rows`, `cols` and `boxes` done twice give the identity function

```
rows . rows = id
cols . cols = id
boxes . boxes = id
```

Choices

The type Choices is a list of possible values for a cell.

1. Most online sudoku apps provide them as hints.
2. Initially:
   - The choices for a blank cell are all possible characters in allvals.
   - The choices for a filled cell is the singleton list containing the cell.

```
fillin :: Char -> [Char]
fillin c
  | blank c = allvals
  | otherwise = [c]
initialChoices b = map (map fillin) b
```

All possible boards

cp is the Cartesian product of a list of lists.

```
cp [] = [[]]
cp (xs:xss) = [x:ys | x <- xs, ys <- cp xss]
```

Given cp how can one define the matrix cartesian product of all rows.

```
mcp = cp . map cp
```

`map` cp converts a matrix of choices to:

[list of all possible first rows,
list of all possible second rows,
. . . ,
list of all possible ninth rows]

cp then gives all possible boards.

sudokusolver version 1

A sudoku solver takes a board and returns a list of correct solutions.

```
sudokusolver1  = filter correct . mcp . initialChoices
```

Go for a coffee while it runs. In fact go for several coffees.