Functional Programming With Lists

Amitabha Sanyal

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

as@cse.iitb.ac.in

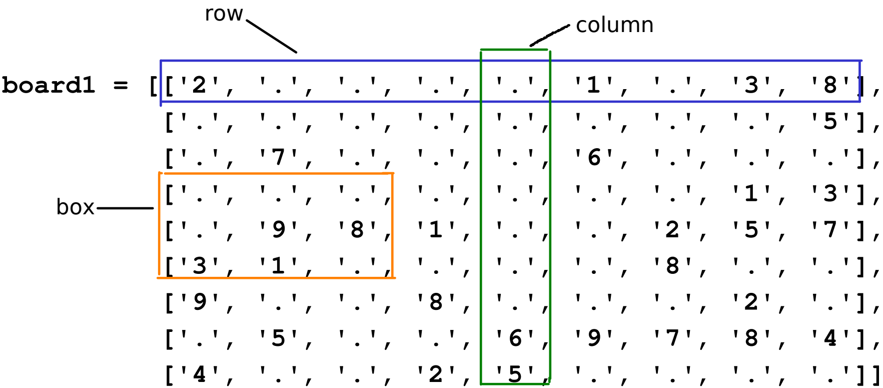
November 2024

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 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
```

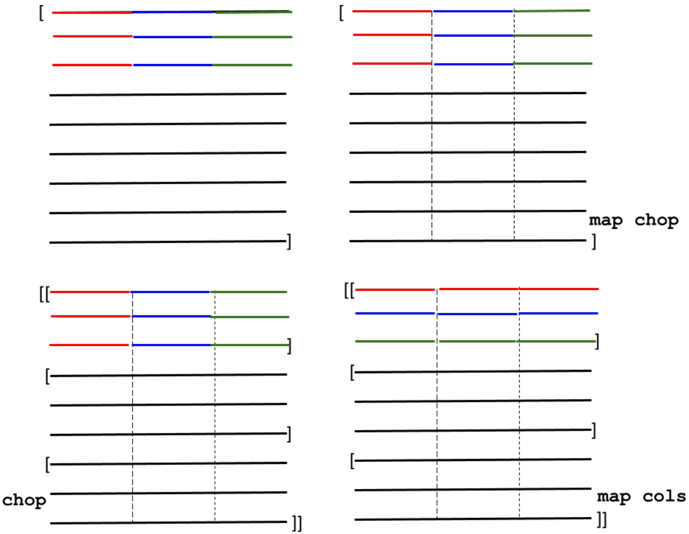
cols makes rows out of columns

```
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**

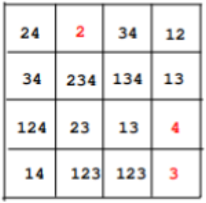
A sudoku solver takes a board and returns a list of correct solutions.

```
sudokusolver1 :: Board -> Board
sudokusolver1  = head . filter correct . mcp . initialChoices
ghci> sudokusolver1 board1
```

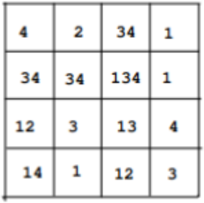
Go for a coffee while it runs. In fact go for several coffees.

Pruning the search space

We would like to prune the search space:



24	2	34	12
34	234	134	13
124	23	13	4
14	123	123	3



4	2	34	1
34	34	134	1
12	3	13	4
14	1	12	3

This is one time pruning.

`pruneList` takes a row of choices, collects all the fixed choices, and removes them from the non-fixed choices.

```
ghci> pruneList [[2,4],[2],[3,4],[1,2]]
[[4],[2],[3,4],[1]]
```

Pruning the search

```
fixed cr = [d | [d] <- cr]

remove fs [x] = [x]
remove fs cs =  [c | c <- cs, c ‘notElem‘ fs]

pruneList css = [(remove fs cs) | cs <- css]
  where
    fs = fixed css
```

Pruning

rows can be pruned by:

```
rows . map pruneList . rows
```

columns and boxes can be pruned by:

```
cols . map pruneList . cols
boxes . map pruneList . boxes
```

Abstract!

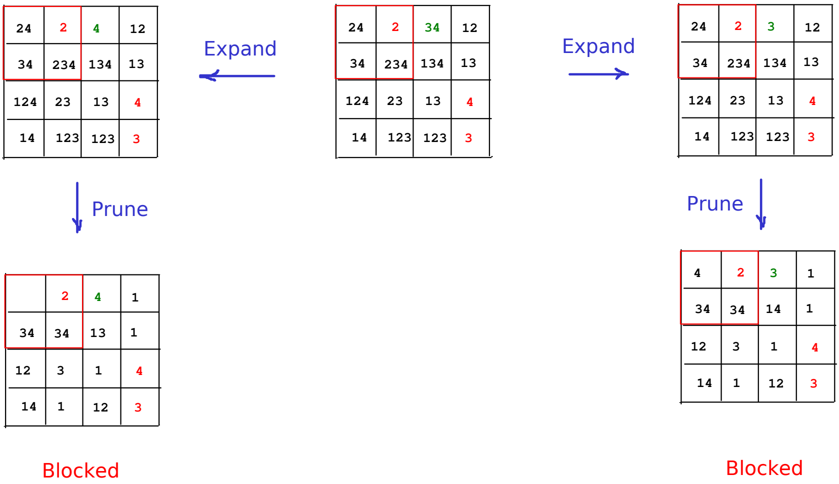
```
pruneBy f = f . map pruneList . f
pruneMatrix = pruneBy rows . pruneBy cols . pruneBy boxes
```

Sudoku solver version 2

```
sudokusolver2 :: Board -> Board
sudokusolver2 = head . filter correct . pruneMatrix . mcp .
                    initialChoices
ghci> sudokusolver2 board1
```

Is the coffee shack still open?

Expand → Prune → Expand → Prune



24	2	34	12
34	234	134	13
124	23	13	4
14	123	123	3

Expand

24	2	4	12
34	234	134	13
124	23	13	4
14	123	123	3

Prune

	2	4	1
34	34	13	1
12	3	1	4
14	1	12	3

Blocked

Expand

24	2	3	12
34	234	134	13
124	23	13	4
14	123	123	3

Prune

4	2	3	1
34	34	14	1
12	3	1	4
14	1	12	3

Blocked

Expand → Prune → Expand → Prune

Expand: Take a choice matrix that has a cell with at least two (say x) choices, and replace it with x choice matrices each containing one of the choices.

1. This enables the possibility of pruning.
2. We can repeat the expand → prune cycle, till:
    1. All cells in the choice matrix have only one choice.
    2. The choice matrix is blocked because of the *void* or the *unsafe* condition.
    3. Blocked matrices are discarded.

Expand → Prune → Expand → Prune

```
blocked cm = void cm || not (safe cm)

void cm = any (any null) cm

safe cm = all (nodups . fixed)  (rows cm) &&
          all (nodups . fixed) (cols cm) &&
          all (nodups . fixed) (boxes cm)
```

Expand → Prune → Expand → Prune

To expand a choice matrix, we select a cell that has a minimum of all cells that have more than one choice.

```
minchoice cm = minimum [length c | c <- concat cm,
                                        length c > 1]
```

A choice list is a candidate for expansion if its length is the same as minchoice

```
expand cm = [rows1 ++ (row1 ++ ([c]:row2)):rows2 | c<-cs]
   where isCandidate c = length c == minchoice cm
         (rows1, row:rows2) = break (any isCandidate) cm
         (row1, cs:row2) = break isCandidate row
```

The Final Solution

```
expandprune cm
  | blocked cm = []
  | all (all single) cm = cm
  | otherwise = [cm2 | cm1 <- expand cm,
                      cm2 <- expandprune (pruneMatrix cm1)]

sudokusolver3 :: Board -> Board
sudokusolver3 = map (map head). head . expandprune .
                initialChoices
```

```
ghci> sudokusolver3 board1
["249571638","861432975","573986142","725698413",
"698143257","314725869","937814526","152369784",
"486257391"]
```