A Sudoku Solver – Expanding (4A)

- Richard Bird Implementation

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Based on

Thinking Functionally with Haskell, R. Bird

https://wiki.haskell.org/Sudoku

http://cdsoft.fr/haskell/sudoku.html

https://gist.github.com/wvandyk/3638996

http://www.cse.chalmers.se/edu/year/2015/course/TDA555/lab3.html

: and ++, and concat

: cons an element onto a list
a -> [a] -> [a]

1 : [2, 3, 4] ==> [1, 2, 3, 4]

++ concatenates two lists
[a] -> [a] -> [a]

[1] ++ [2, 3, 4] ==> [1, 2, 3, 4]

concat : concatenate a list of lists
[[a]] -> [a]

concat [[1, 2], [3, 4, 5]] ==> [1, 2, 3, 4, 5]

http://stackoverflow.com/questions/1817865/haskell-and-differences

any

```
any :: (a -> Bool) -> [a] -> Bool
any p = or . map p

or :: [Bool] -> Bool
or [] = False
or (x:xs) = x || or xs
```

span

```
span :: (a -> Bool) -> [a] -> ([a], [a])
span p [] = ([], [])
span p (x:xs) = if p x then (x:ys, zs)
                       else ([], x:xs)
                       where (ys, zs) = span p xs
```

```
span (< 3) [1,2,3,4,1,2,3,4] == ([1,2],[3,4,1,2,3,4])
span (< 9) [1,2,3] == ([1,2,3],[])
span (< 0) [1,2,3] == ([],[1,2,3])
```

break

```
break :: (a -> Bool) -> [a] -> ([a], [a])
break p = span (not . p)

break even [1,3,7,6,2,3,5] == ([1,3,7], [6,2,3,5])
```

single

```
single :: [a] -> Bool
single [ _ ] = True
single  _  = False
or . map
```

```
(row1, cs:row2)        = break (not . single) row

(rows1, row:rows2)     = break (any (not . single)) rows
                       = break (or . map (not . single)) rows
```

Matrix Choices Example

(row1, cs:row2) = **break** (not . single) **row**

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

Matrix Choices = [Row Choices] ➡ [[Choices]] ➡ [[[Digit]]]

Matrix Choices Example

(row1, cs:row2) = **break** (not . single) **row**

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

rows1

row

rows2

Matrix Choices = [Row Choices] → [[Choices]] → [[[Digit]]]

Matrix Choices Example

(row1, cs:row2) = **break** (not . single) **row**

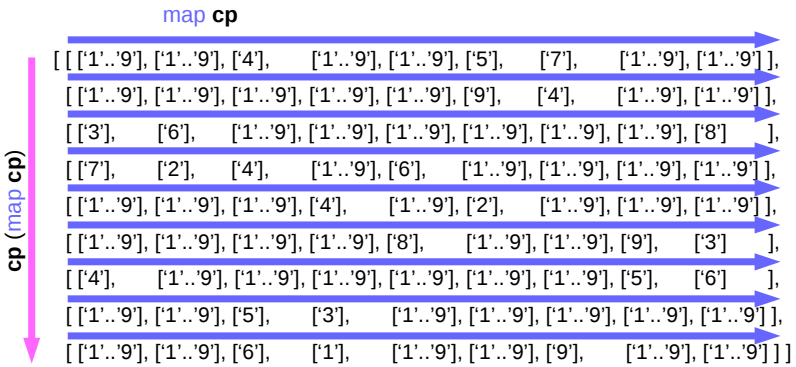
```
[ [ ['1'],       ['2'],       ['3'],       ['4'],       ['5'],       ['6'],       ['7'],       ['8'],       ['9'],       ],
  [ ['9'],       ['8'],       ['7'],       ['6'],       ['5'],       ['4'],       ['3'],       ['2'],       ['1'],       ],
  [ ['4'],       ['1'..'9'],  ['1'..'9'],  ['1'..'9'],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

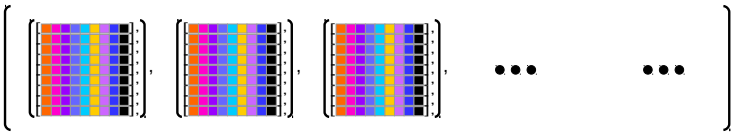
rows1

rows2

```
  [ ['4'],       ['1'..'9'],  ['1'..'9'],  ['1'..'9'],  ['1'..'9'],  ['1'..'9'],  ['1'..'9'],  ['5'],       ['6']        ],
    row1         cs                                     row2
```

expand





expand :: Matrix Choices -> [Grid]
expand = **cp** . map **cp**
cp . map **cp** = [[[a]]] -> [[[a]]]

Matrix Choices

Matrix [Digit]

expand

[Grid]

[Matrix Digit]

expand = concat . map expand . expand1

expand1 :: Matrix [Digit] -> [Matrix [Digit]]
expand1 rows = [rows1 ++ [row1 ++ [c]:row2] ++ rows2 | c <- cs]

Single-Cell Expansion

```
single :: [a] -> Bool
single [_] = True
single _ = False

expand1 :: Matrix [Digit] -> [Matrix [Digit]]
expand1 rows
= [rows1 ++ [row1 ++[c]:row2] ++ rows2 | c <- cs]
  where
  (rows1, row:rows2) = break (any (not . single)) rows
  (row1, cs:row2) = break (not .single) row

break (any (not . single)) rows = [rows, []]
```

Single-Cell Expansion

```
expand1 : Matrix [Digit] -> [Matrix [Digit]]

expand = concat . map expand . expand1

rows = rows1 ++ [row] ++ rows2
row = row1 ++ [cs] ++ row2

expand1 :: Matrix [Digit] -> [Matrix [Digit]]
expand1 rows
= [rows1 ++ [row1 ++ [c]:row2] ++ rows2 | c <- cs]
```

Single-Cell Expansion

```
break :: (a -> Bool) -> [a] -> ([a], [a])
break p = span (not . p)

break even [1,3,7,6,2,3,5]

==> ([1,3,7], [6,2,3,5])

any :: (a -> Bool) -> [a] -> Bool
any p = or . map p

or :: [Bool] -> Bool
or [] = False
or (x:xs) = x || or xs
```

Single-Cell Expansion

```
single :: [a] -> Bool
single [_] = True
single _ = False

expand1 :: Matrix [Digit] -> [Matrix [Digit]]
expand1 rows
= [rows1 ++ [row1 ++[c]:row2] ++ rows2 | c <- cs]
  where
  (rows1, row:rows2) = break (any (not . single)) rows
  (row1, cs:row2) = break (not .single) row

break (any (not . single)) rows = [rows, []]
```

Single-Cell Expansion

```
expand1 :: Matrix [Digit] -> [Matrix [Digit]]
expand1 rows
= [rows1 ++ [row1 ++ [c]::row2] ++ rows2 | c <- cs]
  where
  (rows1, row:rows2) = break (any samllest) rows
  (row1, cs:row2) = break smallest row
  smallest cs = length cs == n
  n = minimum (counts rows)


counts = filter (/= 1) . map length . concat
```

Single-Cell Expansion

```
complete :: Matrix [Digit] -> Bool
complete = all (all single)

safe :: Matrix [Digit] -> Bool
safe m = all ok (rows cm) &&
         all ok (cols cm) &&
         all ok (boxs cm) &&
ok row = nodups [x | [x] <- row]
```

Single-Cell Expansion

```
extract :: Matrix [Digit] -> Grid
extract = map (map head)

filter valid (expand m) = [extract m]

filter valid . expand
= filter valid . concat . map expand . expand1

filter p . concat = concat . map (filter p)

concat . map (filter p . expand) .expand1

concat . map (filter p . expand . prune) . expand1
```

Single-Cell Expansion

```
search = concat . map search . expand1 . prune

solve = search . choices
search cm
| not (safe pm) = []
| complete pm = [extract pm]
| otherwise = concat (map search (expand1 pm))
where pm = prune cm
```

Single-Cell Expansion

```
solve = filter valid . expand . prune . choices

many :: (eq a) => (a -> a) -> a -> a
many f x = if x == y then x else many f y
    where y = f x

solve = filter valid . expand . many prune . choices
```

Single-Cell Expansion

```
expand1   :: Matrix Choices -> [Matrix Choices]
expand1 rows =
  [rows1 ++ [row1 ++ [c]:row2] ++ rows2 | c <- cs]
  where
  (rows1,row:rows2) = break (any smallest) rows
  (row1,cs:row2)    = break smallest row
  smallest cs       = length cs == n
  n                 = minimum (counts rows)

 counts             = filter (/=1) . map length . concat
```

Single-Cell Expansion

```
> solve2 :: Grid -> [Grid]
> solve2 =  search . choices

> search :: Matrix Choices -> [Grid]
> search cm
>  |not (safe pm)  = []
>  |complete pm    = [map (map head) pm]
>  |otherwise      = (concat . map search . expand1) pm
>  where pm = prune cm

> complete :: Matrix Choices -> Bool
> complete = all (all single)

> single [_] = True
> single _   = False
```

Single-Cell Expansion

```
> solve2 :: Grid -> [Grid]
> solve2 =  search . choices

> search :: Matrix Choices -> [Grid]
> search cm
>  |not (safe pm)  = []
>  |complete pm    = [map (map head) pm]
>  |otherwise      = (concat . map search . expand1) pm
>  where pm = prune cm

> complete :: Matrix Choices -> Bool
> complete = all (all single)

> single [_] = True
> single _   = False
```

Single-Cell Expansion

```
> safe :: Matrix Choices -> Bool
> safe cm = all ok (rows cm) &&
>           all ok (cols cm) &&
>           all ok (boxs cm)

> ok row = nodups [d | [d] <- row]
```

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf