A Sudoku Solver - Specifications (1A)

- 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

Basic Data Types

```
type Choices = [Digit]

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

type Grid     = Matrix Digit
type Digit    = Char

digits  :: [Digit]
digits  =  ['1'..'9']

blank   :: Digit -> Bool
blank   =  (== '0')
```

[[a]]

[[Digit]] 9x9 matrix of digits

The valid digits are ‘1’ to ‘9’
A list of non-zero characters
(‘1’ to ‘9’)

‘0’ standing for blank

Matrix Digit & Matrix Choices

```
type      Digit       =   Char
type      Choices     =   [Digit]

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

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

type      Grid        =   Matrix Digit     ➡  [Row Digit]  ➡  [[Digit]]
```

Sudoku

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

```
[ [ '0', '0', '4', '0', '0', '5', '7', '0', '0' ],
  [ '0', '0', '0', '0', '0', '9', '4', '0', '0' ],
  [ '3', '6', '0', '0', '0', '0', '0', '0', '8' ],
  [ '7', '2', '4', '0', '6', '0', '0', '0', '0' ],
  [ '0', '0', '0', '4', '0', '2', '0', '0', '0' ],
  [ '0', '0', '0', '0', '8', '0', '0', '9', '3' ],
  [ '4', '0', '0', '0', '0', '0', '0', '5', '6' ],
  [ '0', '0', '5', '3', '0', '0', '0', '0', '0' ],
  [ '0', '0', '6', '1', '0', '0', '9', '0', '0' ] ]
```

type Grid = Matrix Digit ➡ [Row Digit] ➡ [[Digit]]

Specification (0)

```
solve1    ::             Grid             -> [Grid]
choices   ::             Grid             -> Matrix Choices
expand    ::             Matrix Choices   -> [Grid]
cp        ::             [[ a ]]          -> [[ a ]]
valid     ::             Grid             -> Bool
nodups    :: Eq a =>     [ a ]            -> Bool
rows      ::             Matrix a         -> [Row a]
cols      ::             Matrix a         -> [Row a]
boxs      ::             Matrix a         -> [Row a]

ungroup       = concat
group []      = []
group (x:y:z:xs) = [x,y,z] : group xs
```

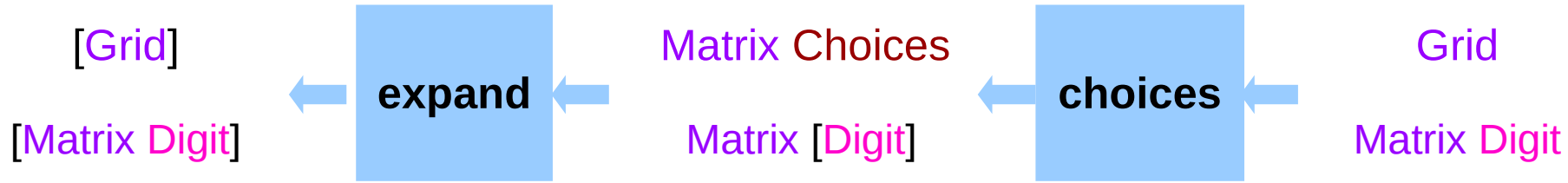
Function Types : choices, expand

```
solve1    ::   Grid           ->   [Grid]
               Matrix Digit        [Matrix Digit]

choices   ::   Grid           ->   Matrix Choices
               Matrix Digit        Matrix [Digit]

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

```
type  Digit      =   Char
type  Choices    =   [Digit]
type  Row    a   =   [a]
type  Matrix a   =   [Row a]

      Matrix Digit      [Row Digit]      [[Digit]]
      Matrix Choices    [Row Choices]    [[Choices]]     [[[Digit]]]

type  Grid       =   Matrix Digit       [Row Digit]      [[Digit]]
```

Function completions

```
solve :: Grid ->  [Grid]
solve  = filter valid . completions

completions :: Grid -> [Grid]
valid :: Grid -> Bool

completions = expand . choices
```



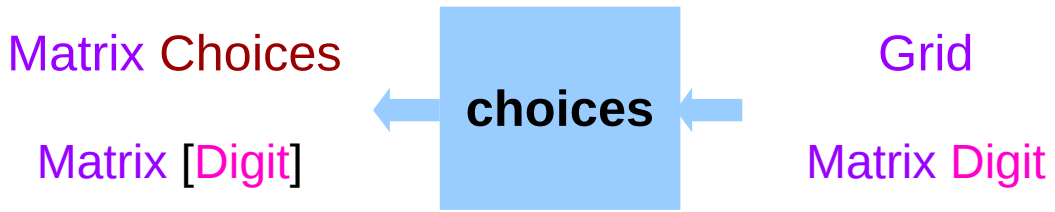
Function: choices

```
choices :: Grid -> Matrix Choices
choices = map (map choice)
  where choice d      | blank d     = digits
                      | otherwise   = [d]
```

```
choices :: Grid -> Matrix [Digit]
choices = map (map choice)
choice d  = if     blank d       then digits    else  [d]
```

```
digits  :: [Digit]
digits  =  ['1'..'9']
blank   :: Digit -> Bool
blank   =  (== '0')
```

Installs the available digits for each cell
If the cell is blank, then all digits for possible choices
else there is only one choice and a singleton is returned



Function: choices

choices = map (map **choice**)

map **choice**

map (map **choice**)

[[‘0’, ‘0’, ‘4’, ‘0’, ‘0’, ‘5’, ‘7’, ‘0’, ‘0’],
[‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘9’, ‘4’, ‘0’, ‘0’],
[‘3’, ‘6’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘8’],
[‘7’, ‘2’, ‘4’, ‘0’, ‘6’, ‘0’, ‘0’, ‘0’, ‘0’],
[‘0’, ‘0’, ‘0’, ‘4’, ‘0’, ‘2’, ‘0’, ‘0’, ‘0’],
[‘0’, ‘0’, ‘0’, ‘0’, ‘8’, ‘0’, ‘0’, ‘9’, ‘3’],
[‘4’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘5’, ‘6’],
[‘0’, ‘0’, ‘5’, ‘3’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’],
[‘0’, ‘0’, ‘6’, ‘1’, ‘0’, ‘0’, ‘9’, ‘0’, ‘0’]]

choice ‘0’ → [‘1’..‘9’]

Matrix Choices Example

[[[‘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’]],
[[‘3’], [‘6’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘8’]],
[[‘7’], [‘2’], [‘4’], [‘1’..’9’], [‘6’], [‘1’..’9’], [‘1’..’9’], [‘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’]],
[[‘4’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘5’], [‘6’]],
[[‘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]]]

Cartesian Product (**cp**)

cp [[1, 2, 3], [2], [1, 3]]

[[1, 2, 3] x [2] x [1, 3]]

[[1, 2, 1], [1, 2, 3], [2, 2, 1], [2, 2, 3], [3, 2, 1], [3, 2, 3]]

cp [ ] = [[ ]]

cp [[1], [2], [3]] => [[1, 2, 3]]

cp [[1], [ ], [4, 5]] => [ ]

Cartesian Product (**cp**)

cp [[1, 2, 3], [2], [1, 3]]

[[1, 2, 1], [1, 2, 3], [2, 2, 1], [2, 2, 3], [3, 2, 1], [3, 2, 3]]

cp [[2], [1, 3]] = [[2, 1], [2, 3]]

cp ([1, 2, 3] **:** [[2], [1, 3]] **)** = [[1, 2, 1], [1, 2, 3],
[2, 2, 1], [2, 2, 3],
[3, 2, 1], [3, 2, 3]]

[1, 2, 3] x **cp** [[2], [1, 3]] = [1, 2, 3] x [[2, 1], [2, 3]]

list comprehension

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

Cartesian Product (**cp**)

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

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

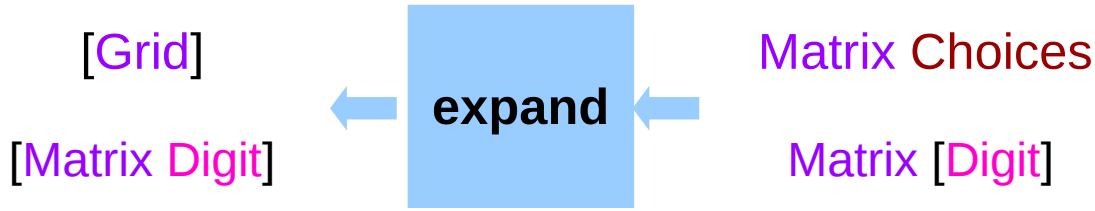
cp [xs] = **cp** (x:[ ])
= [x:ys | x <- xs, ys <- **cp** [ ]] if **cp** [ ] = [ ]
= [x:ys | x <- xs, ys <- [ ]]
= [ ] contradict

cp [ ] = [ ] results in **cp** xss = [ ] therefore **cp** [] = [[ ]]

Expand

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

cp . map cp = [ [[a]] ] -> [ [[a]] ]
```

```
digits  :: [Digit]
digits  =  ['1'..'9']
blank   :: Digit -> Bool
blank   =  (== '0')
```

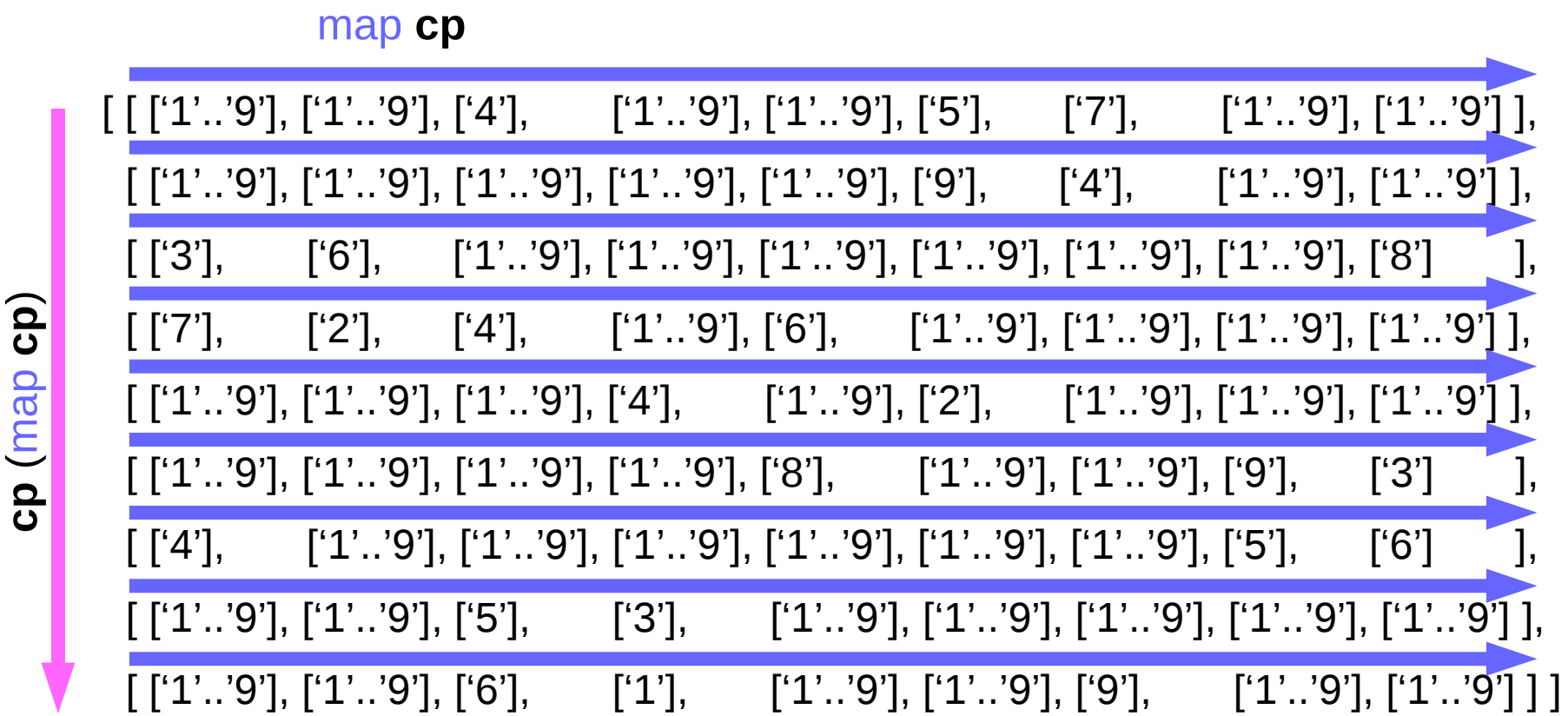


Expand Operations

expand :: Matrix Choices -> [Grid]

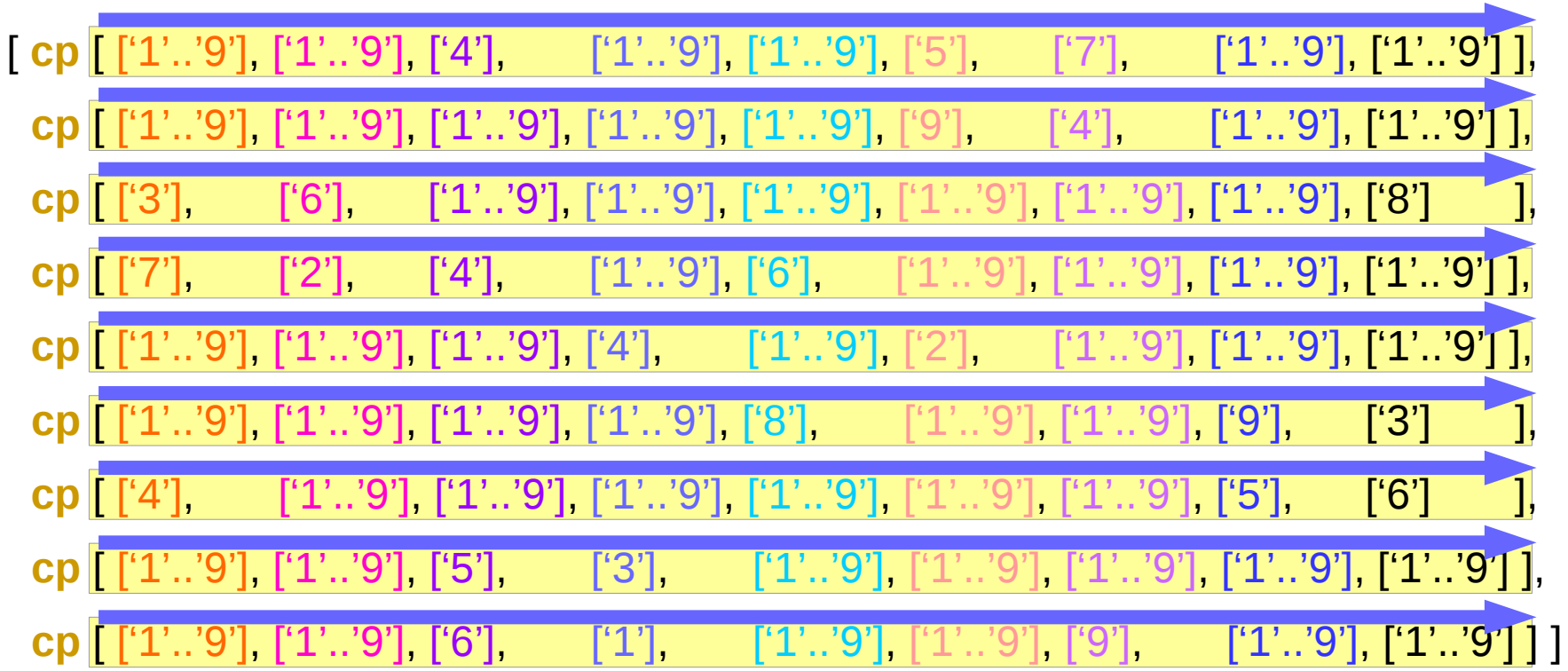
[[[a]]] -> [[[a]]]

expand = **cp** . map **cp**



Matrix Choices Example

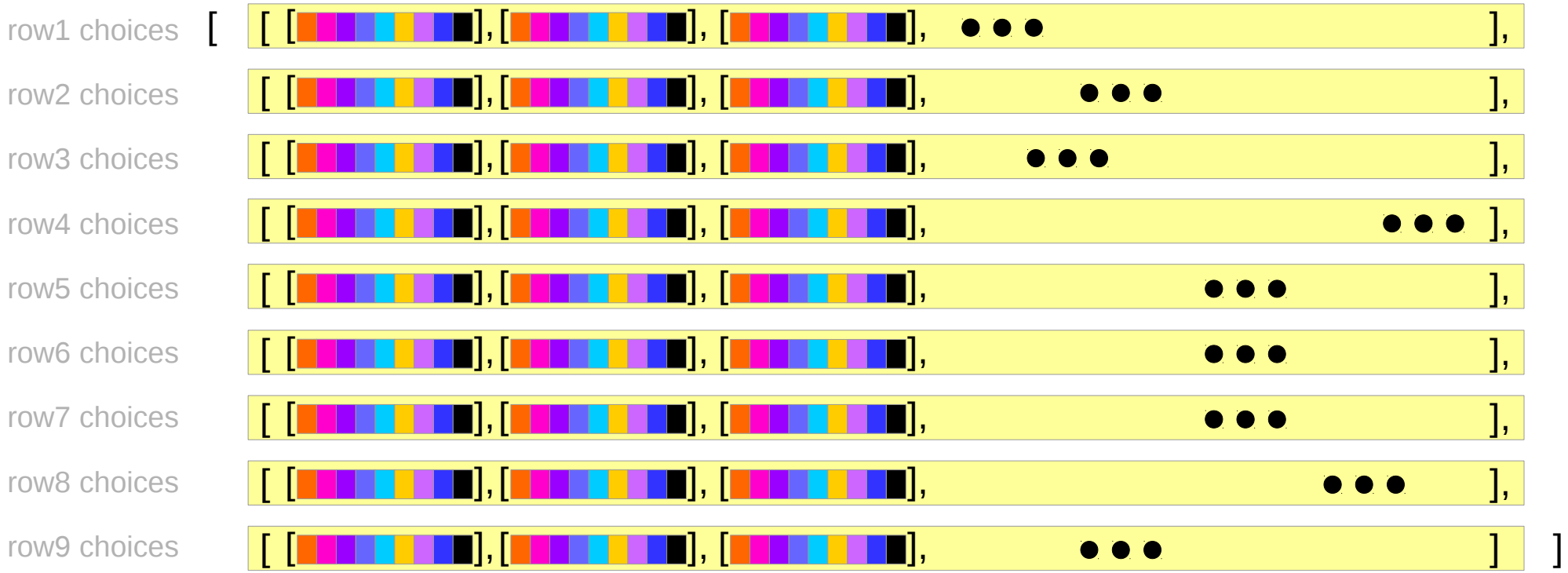
```
expand :: Matrix Choices -> [Grid]
expand = cp . map cp
```




```
cp [[1, 2, 3], [2], [1, 3]]
[[1, 2, 1], [1, 2, 3], [2, 2, 1], [2, 2, 3], [3, 2, 1], [3, 2, 3]]
```

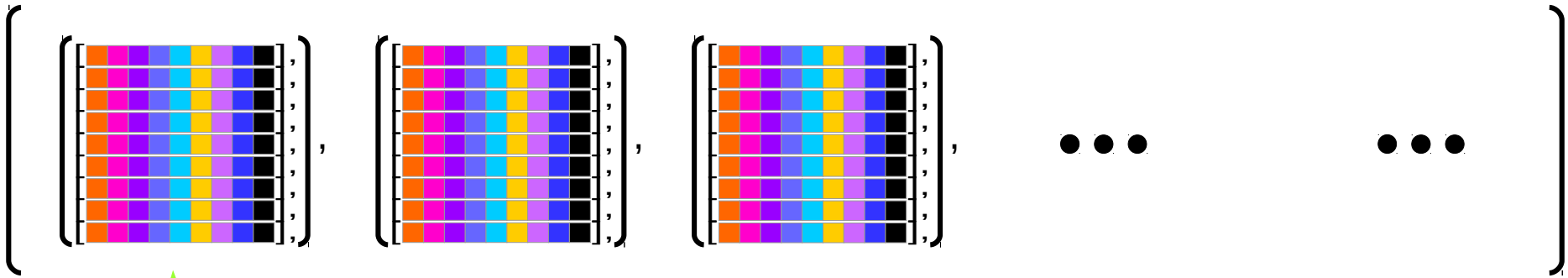
Matrix Choices Example

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

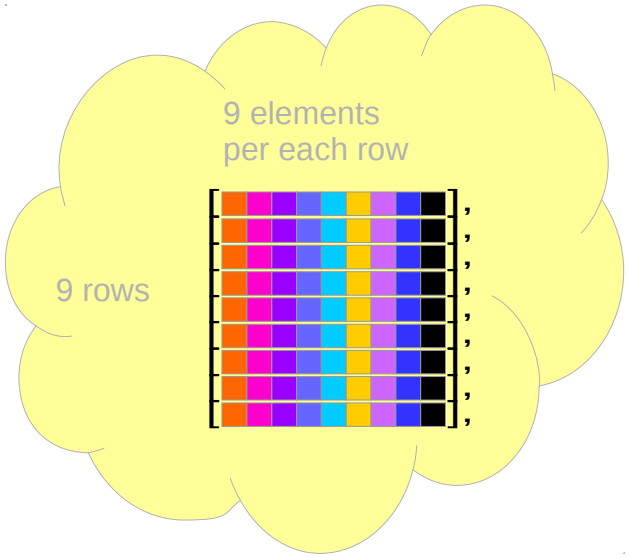



```
cp [[1, 2, 3], [2], [1, 3]]
[[1, 2, 1], [1, 2, 3], [2, 2, 1], [2, 2, 3], [3, 2, 1], [3, 2, 3]]
```

[Grid]



type Grid =
Matrix Digit
➡ [Row Digit]
➡ [[Digit]]



Expand

```
> solve1 :: Grid -> [Grid]
> solve1 = filter valid . expand . choices

> type Choices = [Digit]

> choices :: Grid -> Matrix Choices
> choices = map (map choice)
>  where choice d | blank d   = digits
>                 | otherwise = [d]

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

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

```
digits  :: [Digit]
digits  =  ['1'..'9']
blank   :: Digit -> Bool
blank   =  (== '0')
```

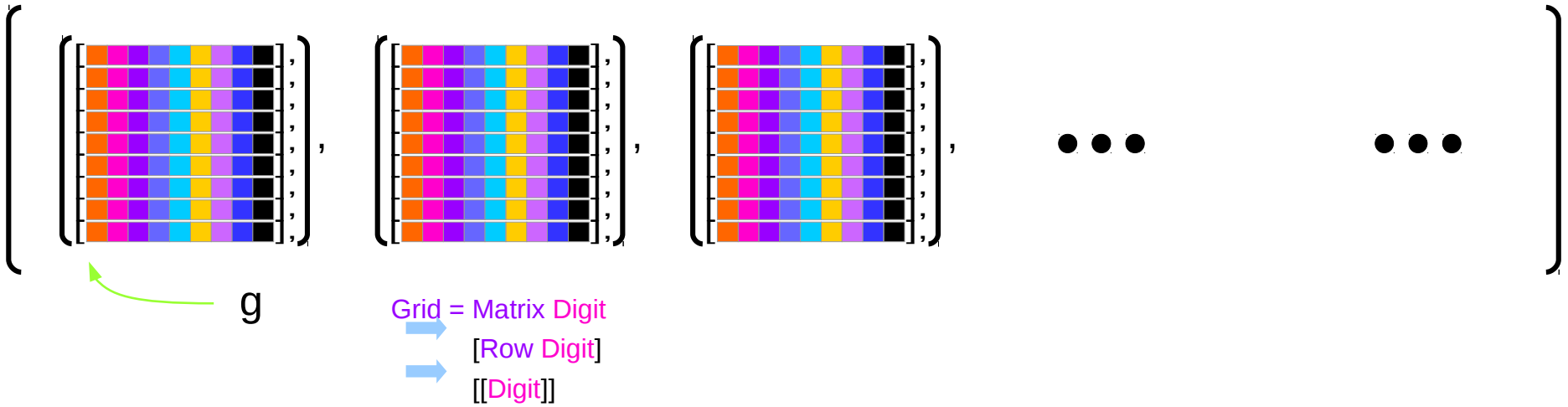
Specification (1)

```
solve1 :: Grid -> [Grid]
solve1 = filter valid . expand . choices

type Choices = [Digit]

choices :: Grid -> Matrix Choices
choices = map (map choice)
  where choice d | blank d   = digits
                 | otherwise = [d]

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

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

```
digits  :: [Digit]
digits  =  ['1'..'9']
blank   :: Digit -> Bool
blank   =  (== '0')
```

Specification (2)

```
valid  :: Grid -> Bool
valid g = all nodups (rows  g) &&
          all nodups (cols  g) &&
          all nodups (boxs g)

nodups :: Eq a => [a] -> Bool
nodups []       = True
nodups (x:xs) = x `notElem` xs && nodups xs
```



Specification (3)

```
rows :: Matrix a -> [Row a]
rows = id

cols :: Matrix a -> [Row a]
cols [xs]       = [[x] | x <- xs]
cols (xs:xss)   = zipWith (:) xs (cols xss)

boxs :: Matrix a -> [Row a]
boxs =    map ungroup . ungroup .
          map cols .
          group . map group
```

group and ungroup

ungroup = concat

group [] = []
group (x:y:z:xs) = [x,y,z] : **group** xs

[x, y, z, xs] ➡ [[x, y, z], **group** xs]

rows and cols

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

```
rows :: Matrix a -> [Row a]
rows :: Matrix a -> Matrix a
rows = id
```

***id** : identity function*

If a matrix is given by a list of its rows

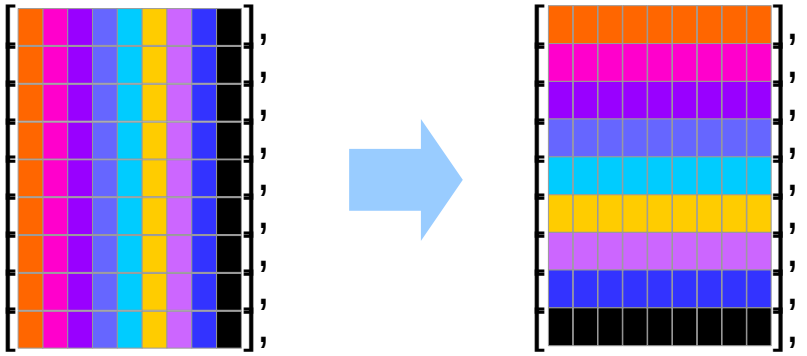
tt returns the same matrix

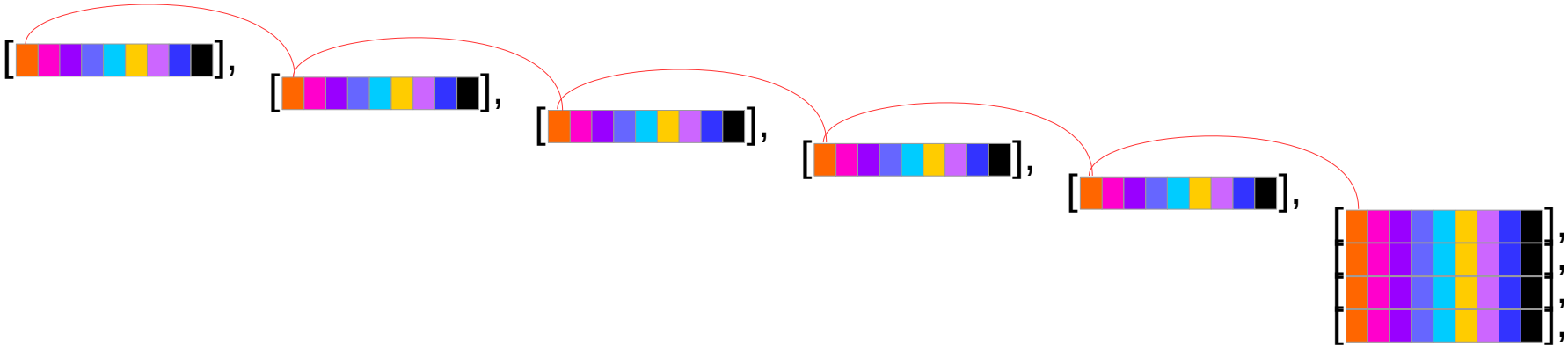
```
cols          :: Matrix a -> [Row a]
cols          :: Matrix a -> Matrix a
cols [xs]     = [[x] | x <- xs]
cols (xs:xss) = zipWith (:) xs (cols xss)
```

transpose of a matrix

cols

```
cols          :: Matrix a -> [Row a]
cols          :: Matrix a -> Matrix a
cols [xs]      = [[x] | x <- xs]
cols (xs:xss) = zipWith (:) xs (cols xss)
```





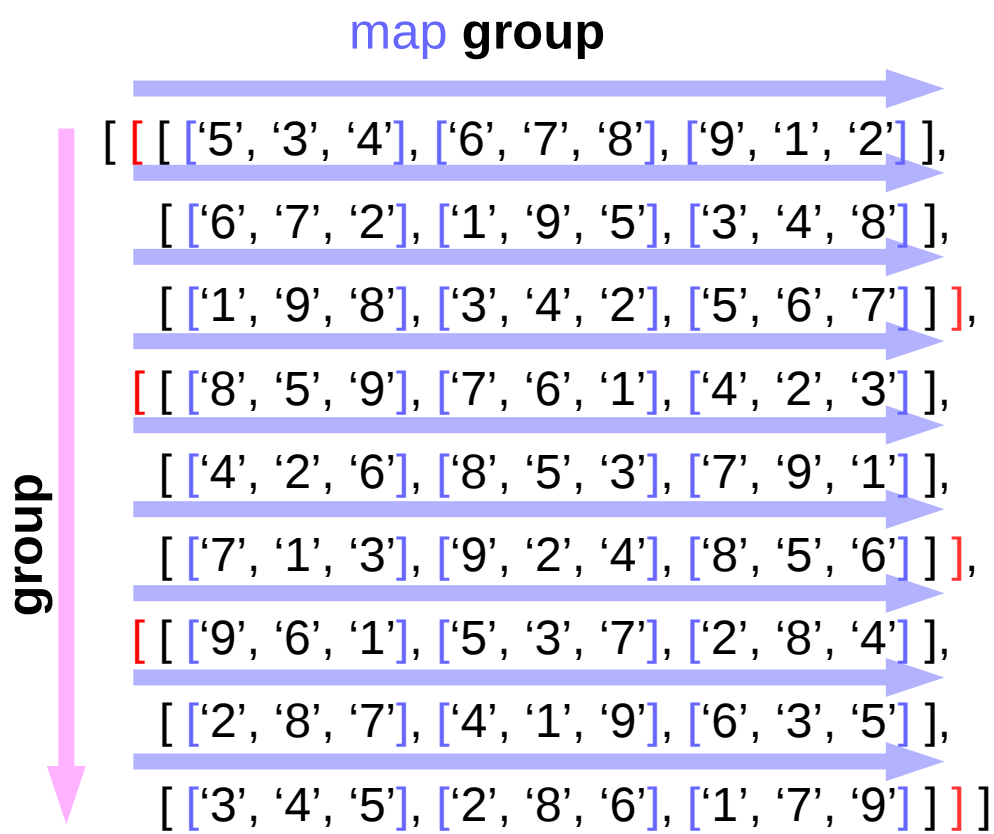
boxs

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

```
boxs :: Matrix a -> [Row a]
boxs :: Matrix a -> Matrix a
boxs =      map ungroup .
            ungroup .
            map cols .
            group .
            map group

ungroup :: [[a]] -> [a]
ungroup       = concat              takes a grouped list and ungroups it

group []      = []
group (x:y:z:xs) = [x,y,z]:group xs     splits a list into groups of three
group xs = take 3 xs : group (drop 3 xs)
```

group.map group

```
[ [ ‘5’, ‘3’, ‘4’, ‘6’, ‘7’, ‘8’, ‘9’, ‘1’, ‘2’ ],
  [ ‘6’, ‘7’, ‘2’, ‘1’, ‘9’, ‘5’, ‘3’, ‘4’, ‘8’ ],
  [ ‘1’, ‘9’, ‘8’, ‘3’, ‘4’, ‘2’, ‘5’, ‘6’, ‘7’ ],
  [ ‘8’, ‘5’, ‘9’, ‘7’, ‘6’, ‘1’, ‘4’, ‘2’, ‘3’ ],
  [ ‘4’, ‘2’, ‘6’, ‘8’, ‘5’, ‘3’, ‘7’, ‘9’, ‘1’ ],
  [ ‘7’, ‘1’, ‘3’, ‘9’, ‘2’, ‘4’, ‘8’, ‘5’, ‘6’ ],
  [ ‘9’, ‘6’, ‘1’, ‘5’, ‘3’, ‘7’, ‘2’, ‘8’, ‘4’ ],
  [ ‘2’, ‘8’, ‘7’, ‘4’, ‘1’, ‘9’, ‘6’, ‘3’, ‘5’ ],
  [ ‘3’, ‘4’, ‘5’, ‘2’, ‘8’, ‘6’, ‘1’, ‘7’, ‘9’ ] ]
```



type Grid = Matrix Digit → [Row Digit] → [[Digit]]

map cols. group . map group

```
[ [ [ ‘5’, ‘3’, ‘4’], [‘6’, ‘7’, ‘8’], [‘9’, ‘1’, ‘2’] ],
  [ [‘6’, ‘7’, ‘2’], [‘1’, ‘9’, ‘5’], [‘3’, ‘4’, ‘8’] ],
  [ [‘1’, ‘9’, ‘8’], [‘3’, ‘4’, ‘2’], [‘5’, ‘6’, ‘7’] ] ],
[ [ [‘8’, ‘5’, ‘9’], [‘7’, ‘6’, ‘1’], [‘4’, ‘2’, ‘3’] ],
  [ [‘4’, ‘2’, ‘6’], [‘8’, ‘5’, ‘3’], [‘7’, ‘9’, ‘1’] ],
  [ [‘7’, ‘1’, ‘3’], [‘9’, ‘2’, ‘4’], [‘8’, ‘5’, ‘6’] ] ],
[ [ [‘9’, ‘6’, ‘1’], [‘5’, ‘3’, ‘7’], [‘2’, ‘8’, ‘4’] ],
  [ [‘2’, ‘8’, ‘7’], [‘4’, ‘1’, ‘9’], [‘6’, ‘3’, ‘5’] ],
  [ [‘3’, ‘4’, ‘5’], [‘2’, ‘8’, ‘6’], [‘1’, ‘7’, ‘9’] ] ] ]
```

```
[ [ [ [‘5’, ‘3’, ‘4’], [‘6’, ‘7’, ‘2’], [‘1’, ‘9’, ‘8’] ],
  [ [‘6’, ‘7’, ‘8’], [‘1’, ‘9’, ‘5’], [‘3’, ‘4’, ‘2’] ],
  [ [‘9’, ‘1’, ‘2’], [‘3’, ‘4’, ‘8’], [‘5’, ‘6’, ‘7’] ] ],
[ [ [‘8’, ‘5’, ‘9’], [‘4’, ‘2’, ‘6’], [‘7’, ‘1’, ‘3’] ],
  [ [‘7’, ‘6’, ‘1’], [‘8’, ‘5’, ‘3’], [‘9’, ‘2’, ‘4’] ],
  [ [‘4’, ‘2’, ‘3’], [‘7’, ‘9’, ‘1’], [‘8’, ‘5’, ‘6’] ] ],
[ [ [‘9’, ‘6’, ‘1’], [‘2’, ‘8’, ‘7’], [‘3’, ‘4’, ‘5’] ],
  [ [‘5’, ‘3’, ‘7’], [‘4’, ‘1’, ‘9’], [‘2’, ‘8’, ‘6’] ],
  [ [‘2’, ‘8’, ‘4’], [‘6’, ‘3’, ‘5’], [‘1’, ‘7’, ‘9’] ] ] ]
```

type Grid = Matrix Digit ➡ [Row Digit] ➡ [[Digit]]

ungroup . map cols . group . map group

ungroup

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

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

type Grid = Matrix Digit ➡ [Row Digit] ➡ [[Digit]]

map ungroup . ungroup . map cols . group . map group

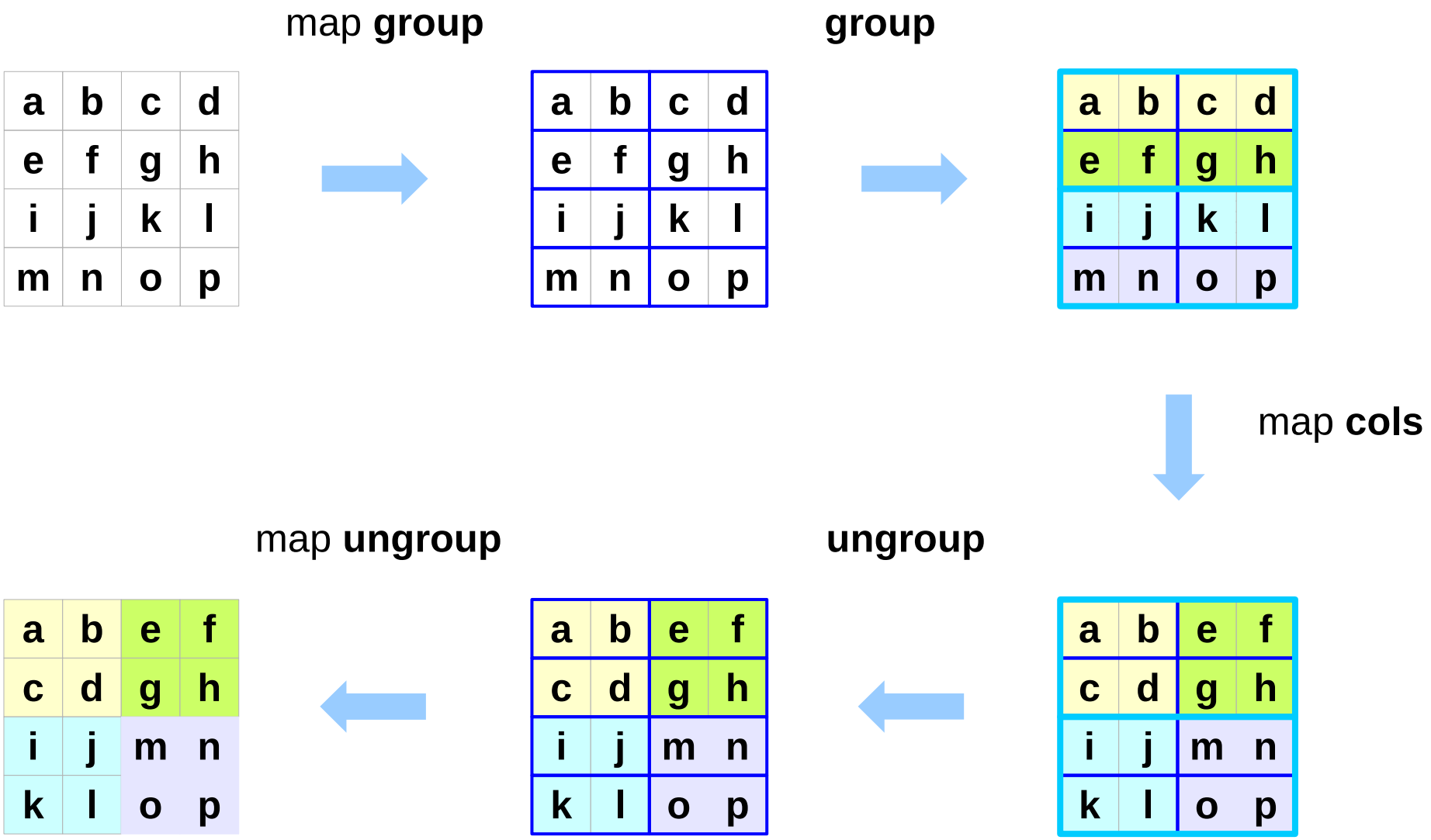
map ungroup

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

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

type Grid = Matrix Digit ➡ [Row Digit] ➡ [[Digit]]

boxs


map group
group
map cols
map ungroup
ungroup
a b c d
e f g h
i j k l
m n o p
a b e f
c d g h
i j m n
k l o p

boxs

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

boxs

a	b	e	f
c	d	g	h
i	j	m	n
k	l	o	p

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

boxs

a	b	e	f
c	d	g	h
i	j	m	n
k	l	o	p

cols

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

cols

a	e	i	m
b	f	j	n
c	g	k	o
d	h	l	p

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

cols

a	e	i	m
b	f	j	n
c	g	k	o
d	h	l	p

rows, cols, boxs

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

rows

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

cols

a	e	i	m
b	f	j	n
c	g	k	o
d	h	l	p

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

boxs

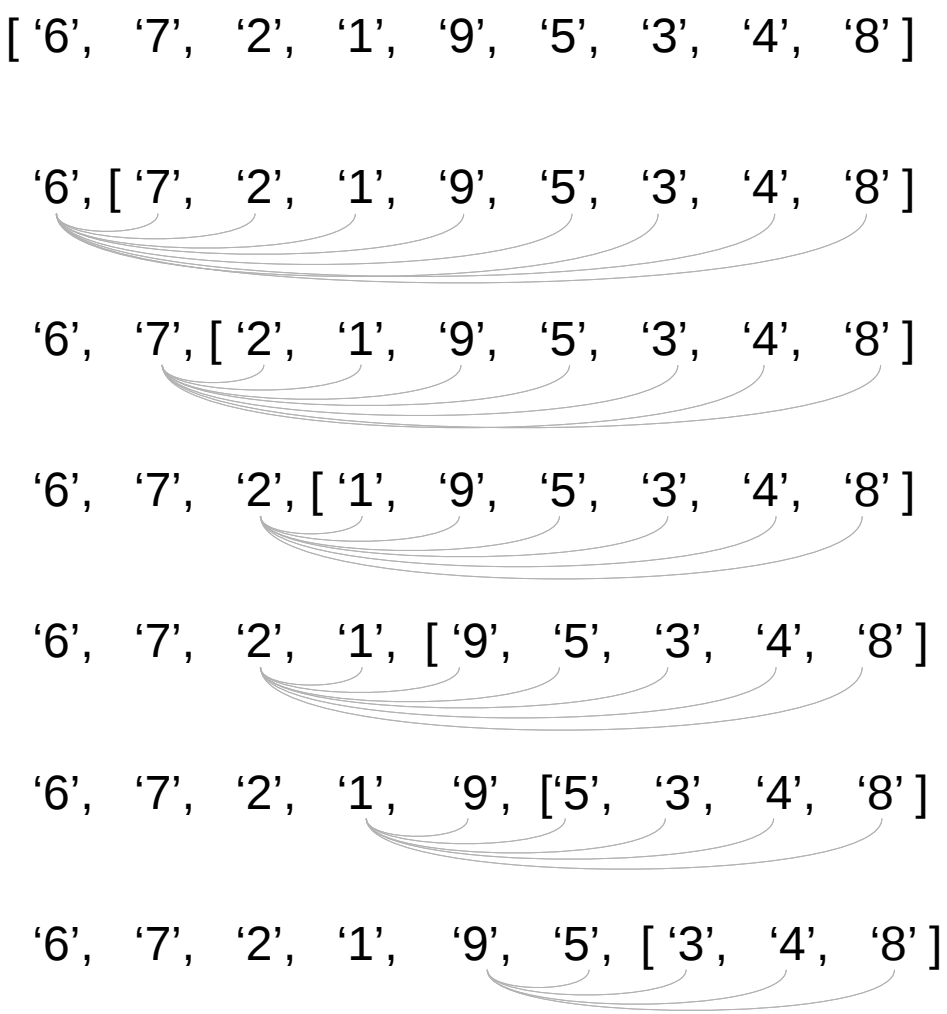
a	b	e	f
c	d	g	h
i	j	m	n
k	l	o	p

nodups

```
nodups  :: (Eq a) =>       [a] -> Bool
nodups []          = True
nodups (x:xs)      = x `notElem` xs && nodups xs

notElem :: (Eq a) =>       a ->  [a] -> Bool
notElem x xs       = all (/= x ) xs

all p = and . map p
```

```
nodups  :: (Eq a) =>       [a] -> Bool
nodups []          = True
nodups (x:xs)      = all (/=x) xs && nodups xs

all p = and . map p
```

nodups




```
nodups (x:xs)    =
      x `notElem` xs && nodups xs

notElem x xs     = all (/= x ) xs

all p = and . map p
```

nodups

[‘6’, ‘7’, ‘2’, ‘1’, ‘9’, ‘5’, ‘3’, ‘4’, ‘8’]

‘6’, ‘7’, ‘2’, ‘1’, ‘9’, ‘5’, [‘3’, ‘4’, ‘8’]

‘6’, ‘7’, ‘2’, ‘1’, ‘9’, ‘5’, ‘3’, [‘4’, ‘8’]

‘6’, ‘7’, ‘2’, ‘1’, ‘9’, ‘5’, ‘3’, ‘4’, [‘8’]

‘6’, ‘7’, ‘2’, ‘1’, ‘9’, ‘5’, ‘3’, ‘4’, ‘8’ []

```
nodups (x:xs)    =
      x `notElem` xs && nodups xs

notElem x xs     = all (/= x ) xs

all p = and . map p
```

References

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